On the Assessment of Robustness in Deep Reinforcement Learning

Tobias Allgeier

Master's thesis

On the Assessment of Robustness in Deep Reinforcement Learning

Tobias Allgeier

February 12, 2021

Chair of Data Processing
Technische Universität München

Tobias Allgeier. *On the Assessment of Robustness in Deep Reinforcement Learning.* Master's thesis, Technische Universität München, Munich, Germany, 2021.

Supervised by Prof. Dr.-Ing. Klaus Diepold and Sven Gronauer; submitted on February 12, 2021 to the Department of Electrical and Computer Engineering of the Technische Universität München.

Chair of Data Processing, Technische Universität München, 80290 München, Germany, http://www.ldv.ei.tum.de/.

Abstract

Machine learning and especially reinforcement learning (RL) technologies have not yet been developed to such an extent that they can replace control technologies in large sections of industrial and technological areas. Missing robustness in RL algorithms is one of the issues that hinder the real-world application so far. There are many disturbances, such as noise or manufacturing tolerances in the real world that cause significant problems. In order to become applicable, the robustness of RL algorithms against different disturbances has to increase. Moreover, objective measures have to be developed that allow the robustness evaluation of an algorithm.

This thesis focuses on the field of robustness in RL algorithms. In the first part, a new robust approach of deep RL is introduced and evaluated. Therefore, the state of the art algorithm Proximal Policy Optimization is applied to a problem defined within a robust Markov decision process. To model these processes, the system is defined with uncertainties in the system parameters. In this case, a worst-case assumption of the possible behavior under the uncertainty of the system parameters is taken into account. Additionally, the influence of the distribution type and spread of the uncertainties is tested. In the second part, the research deals with the question about the robustness evaluation of RL algorithms. Therefore, a robustness benchmark of the standard PPO and the new approach, where PPO and RMDP are combined, is performed. To address the different aspects of robustness in this work, several ways to evaluate robustness against different disturbances are considered. The influence of noise, changing environment parameters and transfer learning are metrics to perform the robustness evaluation.

The robustness evaluation of this thesis leads to the result that the state of the art algorithm based on domain randomization still achieves the most robust performance. Comparing all evaluation metrics and environments the robustness performance is very dependent on the different environments. Furthermore, the types of distributions, introduced in the learning process, do not have an impact on the performance. This research shows that the important parameter to achieve more robust algorithms is the spread of the underlying distribution. Additionally, it uncovers that this parameter varies from every environment and algorithm. In the last part, an outlook for future work concerning robustness based on these results is provided.

Contents

1 Introduction

Reinforcement learning (RL) is a fast-growing technology and gains more and more attention and importance. The success of reinforcement learning algorithms is shown in several research fields such as robotics [Kober et al., 2013] and [Kormushev et al., 2013] or playing games such as Atari [Bellemare et al., 2017] or Go [Silver et al., 2016]. Especially the handling of complex problems with these algorithms was previously never expected to be done within this decade. All in all, the number of papers investigating the topic of reinforcement learning have increased strongly [Henderson et al., 2018].

The aim is to establish reinforcement learning algorithms not only in academic surroundings but also to use these approaches in real-world applications. Since in researches of lots experts such as Henderson et al. [2018], Irpan [2018] or Dulac-Arnold et al. [2019] outlined that RL still has many issues, several studies have already been conducted to solve these problems. In order to me more likely to handle real world scenarios, several research studies focused on the topic robustness of algorithms.

To increase the robustness of algorithms different approaches have been used in different researches. One big topic concerning robustness is to handle adversarial attacks with the algorithm as in the researches of [Vinitsky et al., 2020] or [Pinto et al., 2017]. Other important research aspects are the use of RL algorithms in real-world approaches, where several sim-to-real approaches as in [Peng et al., 2018] or [Andrychowicz et al., 2020] have been conducted. Furthermore, several more approaches can be found focusing on the improvement of robustness, some are also described in this thesis. As of today, robustness remains a very important research topic.

1.1 Problem Description

Noisy sensor data or small parameter deviations in the running system in comparison to the learning system are reasons why the agent cannot work properly. Even small differences between real-world scenarios and simulation problems can cause big issues for the agent and can lead to a misbehaviour [Henderson et al., 2018]. This misbehaviour in the industrial context can have big impacts when for example a product line has to stop or an autonomous driving car makes wrong decisions. To avoid this scenario the aim of this project is, to improve and benchmark robustness of RL. In

the future, RL should be applied in large areas to handle real-world problems instead of being mainly used for simulation problems.

Trying to address these problems the idea is to change the definition of the process, where the reinforcement learning agent has to act. Several possibilities to make these changes are described by Feinberg and Shwartz [2012]. In this thesis, the focus is on Markov decision processes with uncertain transitions, that are first introduced by Satia and Lave Jr [1973]. This topic gained more attention in research when, for example Iyengar [2005] and Nilim and El Ghaoui [2004], took it up. The combination of this approach with deep RL, which itself achieves very promising results, might result in a more robust behaviour. Combining these approaches has only been investigated so far in the course of a master thesis, that considers drone control, by Bjarre [2019]. To the best of my knowledge, a benchmark on robustness has never been performed on this approach.

This leads to the next problem addressed in this thesis: the question of how robustness can be measured and what a more robust algorithm in general means. While it is easier to distinguish robustness in cases of classification, for example, it is more complex in other machine learning problems. Therefore several researches, as for example [Baker et al., 2008], [Fernandez et al., 2005] and [Schain and Schain, 2015] have tackled this problem. Since it is difficult to define a single robustness benchmark in terms of RL, in further researches different approaches have been taken into account. To handle the problems in the real world the most important measures of robustness for reinforcement learning are noise handling ([Wang et al., 2019]), the handling different system variations ([Mankowitz et al., 2018]) as well as the quality of a transfer ([Peng et al., 2018]).

1.2 Thesis Outline

The contribution of this thesis is twofold. At first, the state of the art algorithm Proximal Policy Optimization (PPO) is applied to a problem defined within a robust Markov decision process. Therefore, different attempts have been made to identify the best variation for this setup. In the second part, a robustness benchmark is provided, taking the state of the art approach, the new approach and the standard PPO into account. The robustness evaluation is based on different metrics.

The structure of the thesis is structured according to the following outline: The Chapter 2 introduces the foundations of reinforcement learning, including function approximations. policy gradients and Markov decision processes. Afterwards, in Chapter 3, a detailed view of the current state of the art works in cases of robustness is investigated. The methods used for the experiments and the evaluation are described in Chapter 4. A detailed view on the experiments and the corresponding results then is provided in Chapter 5 and Chapter 6. Finally, the thesis ends with a conclusion of the results in Chapter 7.

2 Background

The field of machine learning is distinguished into supervised, unsupervised learning and RL. While in supervised learning the machine always needs labeled input-output data pairs, in unsupervised learning the machine learns without labeled data. This chapter sets the theoretical groundings for RL.

The principle of RL is an agent interacting with its environment, which gets feedback based on actions that lead to the best behavior. Therefore reinforcement learning is a very powerful approach to learn behavior in a not fully known or previously defined state-space or environment. Since RL is a very fast-growing and frequently asked research topic, there are several research studies to given a good overview over the whole topic like as example [Li, 2017], [Arulkumaran et al., 2017] or [Kaelbling et al., 1996].

The first section investigates the background of RL in detail. Starting with introducing the Markov decision processes (MDP) as a framework for reinforcement learning algorithms, more complex parts such as value functions in order to evaluate the agent's performance are followed. Second, several approaches of RL such as tabular methods, function approximation, policy gradient and actor-critic methods are described. In the last section, a special case of the Markov decision process is introduced that is assumed to achieve more robustness for reinforcement learning algorithms.

2.1 Reinforcement learning

In reinforcement learning an agent is learning different behaviors with the best possible performance within a given environment through taking actions in the environment. Based on these actions the agent receives feedback, that is necessary in order to achieve the best possible performance in the environment. By taking an action the agent gets a reward based on the action and the follow-up state. This reward indicates how good the action was. The learning process of the agents is based on the earned rewards and leads the agent to perform the actions in the best possible way to earn the best possible reward over a long sequence of actions.

Reinforcement learning problems can be described in so called MDPs. The MDP is a mathematical formalized tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. The single parts of the tuples are described in Szepesvári [2010]:

- countable set of states $s \in \mathcal{S}$
- countable set of actions $a \in \mathcal{A}$
- a state-transition Rule $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ representing the dynamic of the environment,
- the reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that gives the agent feedback over every state-action pair
- a discount factor $\gamma \in [0, 1]$ that limits the relevance of the future rewards in the trajectory τ

The agent starts in an initial state s_0 at the beginning. Then by taking an action a_t at time step t the agent gets the observation with its follow up state s_{t+1}. The follow-up state is given in the state transition rule $P(s_{t+1}|s_t, a_t)$ and added with a reward r_t. The whole procedure is defined as a sequential decision process. The main goal of the agent is to maximize the earned rewards in the environment over the long run. The decision, which action to take in a particular state, is called the policy $\pi : s \rightarrow a$ of an agent. In RL there are also stochastic policies described by the parameter π. Therefore in each time step the agent is in state s and takes action a according to its policy π, observes a new state s_{t+1} at the next time step and gets the defined reward R. All in all the aim is to find the best policy in order to maximize the reward over all time steps within a trajectory.

A very important assumption for this process is the so called *Markov property*. It says the current state s contains all important information for the agent and also includes all necessary information about the history. Considering this property, the next state is just dependent on the current state and does not need any more information about the history. In this case, the decision process is called a fully observable process. If the Markov property does not hold, meaning that the state s does not contain all true state information, we speak about a *partially observable Markov decision process*. In most basic reinforcement learning approaches a fully observable Markov decision process underlies the whole algorithm.

The MDPs have an infinite horizon behaviour. Therefore, the agent interacts with its environment in a infinite behaviour, or unless it has reached a terminal state. The basic structure of an agent in a MDP is shown in Figure 2.1. Each finite sequence of state action pairs can be described in the path $\tau = (s_1 a_1, s_2 a_2, \dots)$. The probability

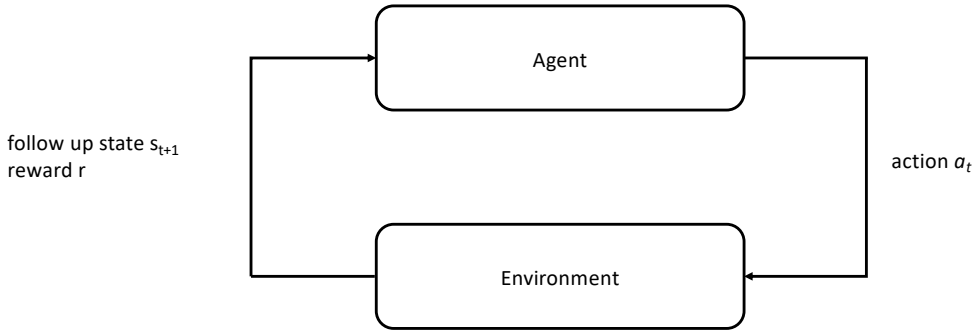

Figure 2.1: The agent interacts with the environment. Every time step t the agent takes an action a_t . This actions causes a change in the environment, which makes the agent receive an reward r and brings it to the next state s_{t+1}

to make this trajectory again can be described with the Markov chain.

$$P(\tau|\pi) = P(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2.1}$$

The probability of all state action pairs, including the initial state are taken into account. Also the overall reward of the trajectory τ according to the policy π can be calculated with

$$R(\tau) = \sum_{t=0}^{T} r_t. \tag{2.2}$$

Where r_t denotes the reward of every state from the trajectory.

The last symbol in the definition of MDPs is the discount factor γ. The discount factor is a value between 0 and 1 and weights the impact of the future expected rewards. As a consequence of the discount factor, every sequence can be bound to a finite horizon. It determines how likely it is to choose a state based on a higher future reward. The discounted return from one point in the sequence can be computed as

$$R_t(\tau) = \sum_{k=t}^{T} \gamma^{k-t} r_k, \tag{2.3}$$

where t represents the current time step.

Based on these future rewards it is possible to perform an optimization of the policy until the end of each trajectory. In contrast to the dynamic programming approach, not all states and trajectories are known in RL but the optimization has to take place based on the current seen trajectories. The performance of the current agent can be

evaluated with value functions. A value function calculates the expected reward at one state with a stationary policy:

$$\begin{aligned} V_\pi(s) &\triangleq \mathbb{E}_\pi[R_t|s_t = s] \\ &= \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t|s_0 = s] \end{aligned} \tag{2.4}$$

In other words, the value function describes the future discounted reward while acting according to a policy. As a next step, the action-value function $Q_\pi(s,a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ describes the expected return. When starting in state s and taking action a, then follow the stationary policy π is followed afterwards. Therefore a similarity between value- and action-value-function exists and is described as follows:

$$\begin{aligned} Q_\pi(s,a) &\triangleq \mathbb{E}_\pi[R_t|s_0 = s, a_0 = a] \\ &= \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t|s_0 = s, a_0 = a] \end{aligned} \tag{2.5}$$

All of the value functions obey special self-consistency equations called Bellman equations. The basic idea behind the Bellman equations is that the value of your starting point is the expected reward from this state plus the expected value-function of the following state s':

$$V_\pi(s) = R(s,a) + \gamma V_\pi(s') \tag{2.6}$$

By maximizing the value-function parts over the actions these equations lead to the optimal value function V^*:

$$V^*(s) = max_a(R(s,a) + \gamma V^*(s')) \tag{2.7}$$

Value functions as well as action-value functions are often estimated by function approximations. A main goal of RL is to learn the optimal value function and therefore to decide in the best possible way within the environment. In order to approximate the value functions, linear and nonlinear function approximators are taken into account. Especially for handling more complex problems or higher dimensional data sets, the calculation of value functions gets more and more complex. The introduction of neural networks as function approximators for complex problems, was the first step of so called deep-reinforcement learning, introduced by Mnih et al. [2013].

In order to optimize the RL problem, there are two different approaches to be considered: on-policy optimization and off-policy-optimization. On-policy methods attempt to evaluate or improve the policy that is used to make decisions. In contrast, off-policy methods usually use generated data and compare the behaviour to the current policy. In the following section, some standard reinforcement learning algorithms are introduced.

2.2 Approaches

After the first introduction to the terminology of RL and its notation, the following different approaches of RL algorithms are described. The first and most important distinction of RL algorithms is whether the agent has access to a model or whether it learns a model of the environment. The main advantage of model-based algorithms is that the agent can plan by thinking ahead. Therefore the upcoming behavior can be taken into account for learning the new policy. Since often there is no environment model available and learning an accurate model is very difficult, this approach is not always applicable. Therefore the agent often has to build up the environment model based on experiences. The quality of the model depends on the algorithm used and therefore might be sub-optimal. Model learning in general is difficult and computationally expensive. Also concerning model-based RL, there are several approaches to improve robustness and to handle these uncertainties as for example described by Zhou et al. [2019]. Anyways, out of these reasons the following approaches focus on the other option of RL algorithms, the model-free learning.

As the name suggests model-free RL algorithms ignore the models, instead they learn based on actions and rewards. The aim of these algorithms is not to create a model, but to find the best policy for an agent that interacts with its environment without having any knowledge over the whole model. To achieve the goal of an ideal policy, the optimization can be performed through different ways such as temporal differences between the steps, policy gradients for optimization, or approximate learning approaches in more complex problems.

2.2.1 Tabular Methods

The simplest approaches to realize model-free reinforcement learning are tabular methods. In order to maximize the Bellman equation through learning the best policy by using tabular methods, an iterative behavior is used. Mainly two different approaches are very common for this kind of algorithms, the Monte Carlo (MC) algorithm and the temporal difference method.

In the case of the MC algorithm, the agent moves through a whole episode. At every state, the agent has a high probability to take the best action based on the experience learned so far. In the beginning, the agent is more likely to take random actions to explore more of the environment than later on. In this manner, the algorithm calculates the value function of each state based on the result of the whole episode. With this behaviour, the agent tries to explore all states and therefore it chooses the policy based on the best overall value.

$$V_\pi(s) = 1/\tau \sum_{s=0}^{\tau} \sum_{t=0}^{T} R(s,a) \tag{2.8}$$

With this behaviour in MC algorithms, the agent can learn the best behavior directly

through the interaction with its environment. Taking also the reward factor into account, this approach can also be used for infinite horizon problems. To get another solution for infinite or large state space problems the temporal difference (TD) algorithm is used. This algorithm works similarly to the MC method. The difference is that this algorithm does not has to wait until the end of an episode in order to know the return. It computes the value of the current state based on estimates of other states. In general, the TD algorithm just looks ahead one step and computes the next value. This optimal value for each state-action entry can be calculated according to the update rule

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha(r + \gamma max_{a'}Q_t(s',a')). \tag{2.9}$$

In other implementations, there is also a TD-λ approach that takes λ steps into account in order to make a link between MC and TD methods. These methods are well suitable for smaller action and state spaces. In larger sets, these methods increase computational cost exponentially. Because of these reasons for larger and more complex problems other methods are taken into account to optimize the Bellman equation.

2.2.2 Function Approximation

For larger and more complex state and action spaces the estimation of value or action-value functions is performed with function approximation. When

$$\begin{aligned} \hat{V}_w(s) &\approx V_\pi(s) \\ \hat{Q}_w(s) &\approx W_\pi(s) \end{aligned} \tag{2.10}$$

holds, function approximation is suitable for the problem. In this case w describes such a parameter vector. Function approximation can be realized by different methods as linear combinations, support vector machines, regression methods or neural networks ([Bertsekas et al., 1995] [Bishop, 2006]).

In this thesis the main focus is on function approximation through neural networks. A multilayered feed forward network is used to estimate value functions and to give a policy. The neural network is built up with non-linear function transformations. Each consists of a linear combination of the inputs ($Wx + b$) with a non-linear activation function $h(\cdot)$. Therefore the output can be described as:

$$y(x,W,b) = \sigma(W_n h(W_{n-1}h...(W_1 x + b_1)... + b_{n-1}) + b_n) \tag{2.11}$$

where σ describes the output activation function.

Function approximators estimate the current value function but do not compute it exactly. To measure the closeness or accuracy of the approximator with the real values the square error between real and predicted value is calculated:

$$J_v(\theta) = \mathbb{E}_\pi[\frac{1}{2}(\hat{V}_w(s) - V_\pi(s))^2] \tag{2.12}$$

To minimize this error $J(\theta)$ has to be fully differentiable with respect to θ. Minimization is performed with a descend of $J(\theta)$ in direction of the gradient $\nabla J(\theta)$. Due to the fact that in practice the correct value functions are not available the gradient descend is performed on the estimated value functions:

$$\nabla_\theta J_v(\theta) = \nabla_\theta \mathbb{E}_\pi[\frac{1}{2}(\hat{V}_w(s) - V_\pi(s))^2] \tag{2.13}$$

After that gradient descend step the parameters are iterative updated according the iteration loop and the learning rate defined for the system.

$$\theta_{k+1} \leftarrow \theta_k + \alpha\nabla\theta \text{ with } \nabla\theta = \nabla_\theta J_v(\theta) \mid_{\theta_k} \tag{2.14}$$

Function approximation is widely used in practice to calculate values of states and actions when similar circumstances occur. For more complex problems function approximators and especially neural networks are a lot more computational efficient than tabular methods ([Gronauer, 2019]).

2.2.3 Policy Gradient

Besides tabular learning methods, policy-based learning methods are often used in RL. Instead of computing the value functions of each state concerning future steps, the optimization is performed by directly changing the policy. The main idea behind policy-based learning algorithms is, to determine which action a at a state s to take in order to maximize the reward. To achieve this objective a vector of parameters called θ parametrizes the policy π to achieve the best action a. By definition, that policies are usually not fully deterministic the policy gives a probability of taking an action in a particular state which leads to this notations:

$$\pi(a \mid s, \theta) = P_r(a_t = a \mid s_t = s) \tag{2.15}$$

Policy-based algorithms are used because they directly optimize the policy instead of the value iterating algorithms. They are very effective to use in high dimensional or continuous action spaces as they need less memory and computation resources. However, the big disadvantage of these methods is the behaviour that it usually converges to a local rather than a global optimum.

The optimization of the policy is fulfilled when the expected rewards are maximized. The maximization is performed by defining an objective $J(\theta)$ with respect to the expected rewards $R(s, a)$ that we get in every time step from zero to the end of the episode T while following the policy $\pi(\theta)$. To get the total reward over one episode we formulate a trajectory τ that starts from zero to time step T and sums up all the expected rewards.

$$J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} R(s_t, a_t)] = \sum_\tau P(\tau \mid \theta)R(\tau) \tag{2.16}$$

After defining the objective function the next step to achieve an optimization is to maximize this function. The maximization is performed as mentioned in the function approximation part with a gradient descend approach. In this case the policy π_θ with respect to the parameters θ is optimized by performing a step in the direction of the gradient. Taking the gradient of the function with respect to θ leads to:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau) \\ &= \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau) \\ &= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)}\nabla_\theta P(\tau;\theta)R(\tau) \\ &= \sum_\tau P(\tau;\theta)\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)}R(\tau)\end{aligned} \tag{2.17}$$

Notice when $\nabla p(x) = p(x)\frac{\nabla p(x)}{p(x)} = p(x)\nabla log p(x)$ holds, the equation can be written as follows:

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau;\theta)\nabla_\theta log P(\tau;\theta)R(\tau) \tag{2.18}$$

The approximation with the empirical estimate for m sample paths under the policy π_θ leads to:

$$\hat{g} = \nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta log P(\tau^{(i)};\theta)R(\tau^{(i)}) \tag{2.19}$$

Now this expression is also valid when R is discontinuous or even unknown and also if the sample space is a discrete set. Starting at this point the the gradient is calculated over the whole trajectory, which leads to:

$$\begin{aligned}\nabla_\theta log P(\tau;\theta) &= \nabla_\theta log\left[\prod_{t=0}^{H} \underbrace{P(s_{t+1} \mid s_t, u_t)}_{\text{dynamics model}} \underbrace{\pi_\theta(u_t \mid s_t)}_{\text{policy}}\right] \\ &= \nabla_\theta\left[\sum_{t=0}^{H} log P(s_{t+1} \mid s_t, u_t) + \sum_{t=0}^{H} log\pi_\theta(u_t \mid s_t)\right] \\ &= \nabla_\theta \sum_{t=0}^{H} log\pi_\theta(u_t \mid s_t) \\ &= \sum_{t=0}^{H} \underbrace{\nabla_\theta log\pi_\theta(u_t \mid s_t)}_{\text{no dynamics model required!}}\end{aligned} \tag{2.20}$$

This equation shows, that the policy gradient can be computed without having access to the environment dynamics. In the following the gradient-log probability of the

trajectory can be put in the policy gradient term and leads to:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t \mid s_t)R(\tau)\right] \tag{2.21}$$

The mean objective can be computed by summing up the gradient over different trajectories. Therefore the expectation of the policy gradient can be estimated as follows:

$$\hat{g} = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t \mid s_t)R(\tau), \tag{2.22}$$

where $|\mathcal{D}| = |\{\tau_1, ..., \tau_N\}|$ shows the number of sampled trajectories. This equation is known as the *Vanilla policy gradient*, which reduces the probability of bad trajectories and makes good trajectories more likely. The advantage of this gradient is that the term $R(\tau)$ has no bias, but on the other hand, the result $\hat{g}$ has a high variance. There are several ways to reduce variances, which are described in in the following section.

2.2.4 Actor Critic Methods

The Equation 2.22 helps to find out how the policy varies following θ. The return is used to amplify those variations. A high return indicates to the neural network that the progress is heading into the right direction and thus boosts $R(\tau)$. Problems may occur in this policy gradient if the sum of the returns are in total 0. In this case, the gradient equals also 0 and the neural network therefore won't learn from this situation. To solve this problem discounted rewards are taken into account for each step, starting from the current state of the trajectory. This leads to the following gradient:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t \mid s_t)R_t \text{ with } R_t = \sum_{t'=t}^{T} \gamma^{t'} r_{t'} \tag{2.23}$$

Problems may then occur if it is not clear what value for R_t is considered as a good value. To solve this problem a baseline is taken into account as

$$\nabla_\theta J(\pi_\theta) = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t \mid s_t)R_t - b_t, \tag{2.24}$$

where b_t represents the average of all actions. This approach is similar to the action value function $Q(s, a)$ and the value function $V(s)$ because each of them considers the average of all rewards that are gained through taking actions at particular states. In this case the equation can be rewritten as:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t \mid s_t)(Q(s_t, a_t) - V_\phi(s_t)) \tag{2.25}$$

Taking a closer look at this equation it can be seen that it consists of two parts. $\pi(a|s)$ is responsible for taking the action, while the second term described by action-value and value functions shows how good the chosen action was. The two parts are called the actor and the critic in each step.

$$\nabla_\theta J(\pi_\theta) = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \underbrace{log\pi_\theta(a_t \mid s_t)}_{Actor} \underbrace{(Q(s_t, a_t) - V_\phi(s_t))}_{Critic} \tag{2.26}$$

As already mentioned in the sections before the critic can be computed in different ways such as tabular methods or for example function approximation including neural networks.

2.3 Robust Markov Decision Process

Markov decision processes as described in Section 2.1 are a common approach to model dynamic optimization problems in many different areas. MDPs are often used because of their good traceability. Since the whole action and state space is defined in the MDP, all transitions are traceable. Taking a look at real world applications of RL problems, these properties are often not applicable and a more robust approach for MDPs is taken into account. More different approaches of MDPs, like a partial observable MDP or non stationary MDPs like in Lecarpentier and Rachelson [2019], therefore are discussed in literature. This thesis mainly focuses on the so called Robust Markov decision process (RMDP) also considers uncertainties in transition probabilities first considered by Bagnell et al. [2001]. In comparison to the usual MDP with fixed reward and transition function r and p, uncertainties are defined. In detail the framework is modeled like :$\mathcal{U}_{s,a} \subseteq R_{s,a} \times P_{s,a}$ where P defines a set of possible next transition distributions and R defines a set of upcoming expected rewards. A transition $(P_t) \in \mathcal{U}_\pi$ is randomly chosen out of the sets and results therefore in a reward r_t at each time step and each policy. With focus on infinite horizon decision processes, in contrast to the approach of robust dynamic programming, the calculation of the values is made with respect to a discount factor γ. Therefore, in robust MDPs the value functions for a policy π can be defined as follows:

$$\tilde{V_\pi} := \min_{(r_t, P_t) \in \mathcal{U}_\pi} \sum_{t=0}^{\infty} (\gamma P_t)^t r_t \tag{2.27}$$

By adding the uncertainties of the Markov model there are in contrast to the usual MDP more possibilities for the next step. Considering this problem and handle all possible transitions within the value function calculation, an additional minimization step is performed to be aware of the possible worst case scenario. Minimizing the value function calculation $\tilde{V_\pi}$ by using a stationary and deterministic policy the assumption:

$$(r_t, P_t) = (r_\pi^*, P_\pi^*) \tag{2.28}$$

holds for all time steps t. To achieve the same approach as in normal Markov decision processes a robust Bellman operator is defined as:

$$\tilde{\mathcal{T}}\tilde{v} := \max_{\pi \in A^{|S|}} \min_{(r,P)\in\mathcal{U}_\pi} r + \gamma P\tilde{v} \tag{2.29}$$

With respect to the robust Bellman operator $\tilde{\mathcal{T}}$ it can be seen, that the optimal robust values satisfy

$$\tilde{v}^* = \tilde{\mathcal{T}}\tilde{v}^* \tag{2.30}$$

Therefore, the optimal robust policy $\tilde{\pi^*}$ with respect to $\tilde{v^*}$can be described as follows:

$$\tilde{\pi^*} := arg^* \max_{\pi \in A^{|S|}} \min_{(r,P)\in\mathcal{U}_\pi} r + \gamma P\tilde{v} \tag{2.31}$$

In the new formulation of the robust Markov decision process the optimization problem turned into a $minmax$ problem. In contrast to usual MDPs, where just a single maximization is needed. Because of this reason the robust MDPs are often leading to very conservative results. However it can lead to better results according robustness and works with changing environment parameters.

3 State of the Art

The whole field of reinforcement learning underlies rapid development. Especially robustness is one of the most important and fastest-growing fields in reinforcement learning. It is often considered by transferring the learned policy to slightly different problem settings in the same environment. Another rising topic is the transfer from the simulation into real-world problems, also called transfer learning. Research in this field can be divided mainly into two different parts. The first block focuses on the learning of adversarial behaviour, whereas the second one focuses on the handling and the modelling of uncertainties as well as noisy sensor data to make the agents behaviour more robust. This chapter mainly focuses on these uncertainties and varying system dynamics to improve robustness. Another milestone was the breakthrough of deep learning methods, such as neural networks and function approximators mentioned in Chapter 2.2.2 and in [Mnih et al., 2013], [Bellemare et al., 2017] and [Bellemare et al., 2019]. Deep RL made a big impact and also pushed the development of robustness. Now the research mainly focuses on deep reinforcement learning algorithms as they have delivered very promising results so far.

This chapter contains actual approaches and state of the art techniques concerning RL with RMDPs. Afterwards, other methods to learn robustness of an agent towards changing system dynamics and also several state of the art approaches concerning transfer learning where the sim-to-real approaches have an important role, are described in this chapter.

3.1 Robust Markov Decision Process

The concept of robust MDPs came up already in the early 2000s where the approach has been considered to get more robust reinforcement learning approaches. The robust MDP has the potential to lead to better results than other approaches due to the handling of uncertainties in the transition matrix, as it has been described in Chapter 2.3.

In the beginning, robust dynamic programming was considered based on RMDPs. The first approaches concerning the robustness of reinforcement learning in robust Markov decision processes led to conservative performances as described in [Xu and Mannor, 2007], [Nilim and El Ghaoui, 2004] and [Iyengar, 2005].

Nilim and El Ghaoui [2004] and Iyengar [2005] considered RMDPs in which they model the uncertainties in transition probabilities. Therefore uncertainty sets, where

each action state pair can be selected from the set $\mathbb{P}_{s,a} \subseteq \mathbb{R}_+^{|s|}$ unrelated to other action state pairs, are defined as follows:

$$\mathbb{P} = \times_{s \in S} \mathbb{P}_{s,a}, \text{ where } \mathbb{P}_{s,a} \subseteq \mathbb{R}_+^{|S|} \tag{3.1}$$

These uncertainty sets are so called rectangular uncertainty sets. The rectangularity therefore is a geometric concept, defined by possible conditional projections of the uncertainty set ([Mannor et al., 2016]). Here, states as well as actions are considered, which leads to the name of a *(s, a)-rectangularity*. Through *robust value iteration* the optimal robust policy for this kind of uncertainty sets can be computed.

The next very important generalization according the rectangularity is mentioned by Epstein and Schneider [2003] and further analyzed by Wiesemann et al. [2013]. In this case a *row-wise* or *s-rectangular* uncertainty set is mentioned. In this generalization just one dimension of the rectangularity is taken into account. Therefore, the transition probabilities are unrelated to the uncertainty sets and therefore just corresponding to different states leading to:

$$\mathbb{P} = \times_{s \in S} \mathbb{P}_s \text{ where } \mathbb{P}_s \subseteq \mathbb{R}_+^{|S| \times |A|} \tag{3.2}$$

Wiesemann et al. [2013] shows in their work that with this assumption an optimal robust policy can be computed. In this case the policy has not to be deterministic in contrast to the previous defined rectangular uncertainty sets.

Concerning the performance using RMDPs, these approaches lead to very conservative behaviours of the agent. Xu and Mannor [2007] had a deeper look into the trade-off between performance and robustness using different MDPs. As a result of this work, the research on RMDPs is ongoing and different approaches took place. Described by Mannor et al. [2016] the extension to $k-$rectangularity, a two-dimensional rectangularity, which gives a way to solve RMDPs more efficiently. Also, Delage and Mannor [2010] created a less conservative approach through the extension of the worst-case expected objective. Performing a percentile optimization according to this objective, with regard to the uncertainties in transition kernel and rewards is the method described in the research. Since the transition probabilities are not independent of different states, the cross-correlation enables this approach. Performing state of the art reinforcement learning algorithms has also been investigated in several other researches with RMDPs like [Goyal and Grand-Clément, 2018], [Tamar et al., 2013], [Lim and Autef, 2019], [Petrik and Russel, 2019],[Behzadian et al., 2019] and [Lim et al., 2013].

During the first researches usually just smaller action and state spaces can be handled, when modeling a problem with RMDPs. Therefore, Tamar et al. [2013] considered taking linear function approximation in the reinforcement learning algorithms into account. In this research it is assumed that $\tilde{Q}^{\pi}(s,a) = \phi(s,a)^T w$, where w is a parameter vector and $\phi(s,a)$ stands for the state action feature. With this expression a greedy policy $\pi(s)$ at state s and a vector w can be computed with respect to the

approximation by

$$\pi^*_w(s) = arg\max_r \phi(s,a)^T w \tag{3.3}$$

In this equation let Φ^*_w denote a matrix with $\phi^*_w(s)$ in its rows with $\phi^*_w(s) = \phi(s, \pi^*_w(s))$ with respect to a d-weighted Euclidean norm. Note, that the additional assumption $d \in \mathbb{R}$ is positive has to hold.

Through the usage of the policy optimization step, this approach is the first approach to handle large scale RMDPs with an RL algorithm. The following examples of trading issues show the usefulness of the approach and demonstrates results that handles risk-awareness as well as mitigation of model miss specification under uncertainties. Also, the first approach to scale up RMDPs with linear function approximation is given through this research.

Lim et al. [2013] made an approach applying reinforcement learning in finite horizon problems and using total rewards as performance measures. In the paper Lim and Autef [2019] considered the so-called *course of dimensionality* and also scaled up the RL on RMDPs. Due to the fact that in usual function approximation models mismatch and parameter uncertainties can be amplified, the researches focused on kernel-based function approximation for RMDPs. Kernel averages are used for two different purposes: First to approximate the reward function shown as $\hat{r}(s,a)$ or the transition function is shown as$\hat{p}(s,a)$ and secondly, to approximate a value function. The aim is to use RL in continuous action spaces and to handle with this objective also all uncertainties of the RMDP.

The starting point for the algorithms based on the kernel averager theory is therefore a modified robust Bellman operator for all abstract states $j \in \{1...m\}$:

$$(\tilde{T}w)(j) := \max_{\pi \in A^{|S|}} \min_{i \in \hat{M}(j)} \hat{r}(i, \pi(i)) + \gamma(\hat{p}(i, \pi(i)), \Phi_w) \tag{3.4}$$

In this equation the value v is approximated by Φ_w and this corresponds to a feature vector.

To proof, the performance in the experiments a Gaussian kernel is used in this paper. Described are three different approaches, all with modelled uncertainties in form of varying systems and for example, additional noise to model the environments in RMDPs: The first one is called *Puddle World*. In this approach, the agent has to find a trajectory to get to the target. The second one is called *Acrobot* environment, which is a two-link robot. The agent has to learn to swing its tip above a given high in order to earn rewards. And the last one is called *double pole balancing* where a two-link robot is initialized and has to stay upwards like an inverted pendulum. The result of the research is that the robust approach on kernel-based reinforcement learning outperforms the non-robust approach in the described cases.

Since not only learning algorithms are important to handle RMDPs, also defining the ambiguity sets of the systems is an important part of the research, where Behzadian

et al. [2019] described a new method. In this paper, high-confidence sampling bounds for different ambiguity sets are given. It is also described how to compute weights from rough value functions. The whole research focuses on finite horizon RMDPs. In contrast to the $s, a - rectangularity$ described earlier, this research wants to consider an ambiguity set to maximize the guaranteed return for a confidence level of $1 - \delta$. Therefore, the uncertainty is bounded to a smaller amount of states. With further optimization of the norm weights, the performance improvement is achieved.

In different experiments like *River Swim* and CartPole the concept is tested with different RMDPs. The result in all of the environments is that, if the ambiguity set is within the confidence interval, with optimized weights outperforms the common approach of uniform ambiguity sets significantly.

In literature solving RMDPs with deep reinforcement learning has only been investigated so far in the course of a master thesis of Bjarre [2019] which is explained in Section 3.3. In further researches with respect to robustness in simulation environments, to the best of my knowledge, no one used deep learning algorithms in combination with RMDPs to improve robustness of reinforcement learning problems.

3.2 Robustness in Deep Reinforcement Learning

Besides RMDPs, robustness in deep reinforcement learning is a very frequent research topic. As one of the first researches Morimoto and Doya [2005] addressed this topic in a more general way. Afterwards a lot of different approaches have been investigated in the research, starting from changing system parameters during training ([Mankowitz et al., 2018] or [Hiraoka et al., 2019]) over adding noise to the given system ([Wang et al., 2019]) up to hierarchical reinforcement learning approaches to gain more robustness ([Kulkarni et al., 2016] and [McGovern and Barto, 2001]). Also new distributional deep reinforcement learning algorithms to handle more different and more complex cases ([Bellemare et al., 2017], [Smirnova et al., 2019] and [Bellemare et al., 2019]) took place in the research. Furthermore other papers like [Derman et al., 2018], [Castro et al., 2012], [Hausknecht and Stone, 2015], [Venkataraman and Seiler, 2019], [Sharma and Kitani, 2018], [Tessler et al., 2019] and [Kober et al., 2012] handle this topic. A few of these concepts are described in the following chapter.

Varying Systems

A very common approach to improve the robustness of reinforcement learning approaches concerning varying system dynamics is to change these dynamics during the learning process. They try to scale up approximate value iterations by options is already done by Mann and Mannor [2014]. By adding some changing components during the learning process, better performances over different system settings are achieved. So far several different approaches have been investigated in this direc-

tion. There are several possible methods when changing system dynamics during training. One possibility is to define particular skills, which means behaviours that can be learned by the agent, as in [Mankowitz et al., 2016], [Konidaris and Barto, 2009], [Takacs et al., 2002], [Stulp et al., 2013] and [Da Silva et al., 2012]. Another possibility is to introduce options, which means to introduce different predefined system parameters in the learning process and afterwards test the performance according to these options against the standard environment [Mankowitz et al., 2018] or [Hiraoka et al., 2019].

In the research of Mankowitz et al. [2018] an algorithm which takes the nominal transition behaviour in this case $\hat{P}_\mu$ as well as a behaviour with uncertainties P_μ into account, was provided. By creating trajectories with $\hat{P}_\mu$ and the option policy π_θ the *critic* of the *action-critic method*, implemented as described by Tamar et al. [2013] is used in this case. The actor works like a usual deep Q-network (DQN) as described in the paper of Mnih et al. [2013].

To proof this concept it has been tested in two different environments: *Acrobot* and *CartPole*. In the *CartPole* environment the options are linear interpolated values in the range from $0.5\,m$ to $5\,m$. In the *Acrobot* environment the weights of the arm-links, that have to swing up, is modelled in different options from $1\,kg$ to $5.5\,kg$. To prove the performance an agent is trained with a common DQN and another agent is trained with this robust DQN approach.
After having trained both of the agents, the evaluation takes place with the agent handling all prior specified different system dynamics. The agent trained with the so-called *robust option DQN* outperforms the usual DQN already after the smallest tested change in system dynamics in both environments. The results clearly show that this approach is more robust than a usual approach, however, the research does not address how the performance is developing when taking the agent into an environment outside the options it has learned. Evaluating the agent and therefore the policy over different system dynamics after training is a common approach to discuss robustness.

Noisy Systems

In the real world, the goal for an RL agent is to interact with its environment based on the information it gets from the environment. In this part, another very common use case in RL in order to improve robustness is described. The data recording of the surroundings in real-world scenarios is usually made with sensor data. In contrast to the simulation data, his data is usually somehow noisy. To learn from this data, or to act on this data is a lot more challenging than just working in simulation with perfect data. Therefore, the noise has an important role in the domain of machine learning, where many studies take care of it during the whole training process as described by [Wang et al., 2020] and [Zheng et al., 2014]. Noise can also be used as a metric for robustness in order to benchmark the robustness by adding noise while testing the agent as described in Wang et al. [2019].

While studies like Scott [2015] made experiences in handling noise data in the learning process another option made by Wang et al. [2020] is to process the noisy data in a different way. Often in RL problems, the reward and the observations that the agent gets from its environment in real-time applications is noisy. Therefore, in this research, the main goal was to learn with perturbed rewards and gain more robustness in this way. The idea is to get a noisy reward $\tilde{r}(s, a)$ for every state-action pair from the environment. By calculating a confusion matrix C based on this noise reward and the mean reward $\bar{r}$ a corrected reward $\hat{R}$ is calculated as follows:

$$\hat{R} = (1 - \eta) \cdot R + \eta \cdot C^{-1} R, \tag{3.5}$$

where R is the matrix of noisy rewards from the environment. The linear combination in the term with $\eta \in [0, 1]$ has a role for the trade-off between bias and variance tuning.

By handling this reward noise with this approach the robustness improved independently from the used deep learning algorithm. Different environments like *Pendulum, CartPole* or *Atari* were tested, in each of them, discrete as well as continuous action spaces, could be proven.

As already mentioned noise can also be taken to benchmark the robustness of an agent in an environment as described by Wang et al. [2019]. In this case a lot of *classic control* environments such as *Acrobot, CartPole* and *Hopper* are used and modified as benchmark environments for algorithms. Noise is added on observations ($\sigma_o \in [0.01, 0.1]$) as well as on actions ($\sigma_a \in [0.1, 0.3]$). The noise was modeled as white noise, i.e. $\eta \sim \mathcal{N}(0, \sigma)$.

By adding this noise with a small standard deviation almost all of the tested standard algorithms lost significant performance in the benchmark environments. Therefore, adding noise to benchmark the robustness of algorithms is a good way to also make a link to real-world applications where noisy data always takes place.

Hierarchical Reinforcement Learning

While the system dynamics of a learning system are changing in the real world, the agent often loses a lot of its performance. To produce more robustness often more than just one MDP is used during the learning process. A very promising approach by using an ensemble of source domains for training is described by Rajeswaran et al. [2017]. The training in different source domains is a kind of adversarial learning to gain more robustness. Optimization, in this case, is performed by a *batch policy optimization*. This optimization according to the conditional value at risk is described by Tamar et al. [2015]. This approach is tested with the *Hopper* as described in Chapter 4 and *Half Cheetah* according to Wawrzyński [2009]. These results compared to the standard algorithm *TRPO* had significant higher robustness against changing system dynamics.

Another way to handle changing systems is with a hierarchical behaviour is described in [Kulkarni et al., 2016] and [McGovern and Barto, 2001]. In these cases

often more different learning algorithms are trained. The aim is to perform with the best performance in every system independently. It is building a universal policy to handle all different environments. Therefore, the agents have to consider the system and which agent can bring the best performance in the underlying environment. The *online system identification* ϕ is also investigated by Yu et al. [2017]. In this research, the training process of the model can be formulated as a supervised learning problem with the history roll out H and the model parameter μ as output, which leads to:

$$\theta^* = \min_{\theta} \sum_{(H_i,\mu_i)\subseteq B} \| \phi_\theta(H_i) - \mu_i, \|^2 \tag{3.6}$$

where θ describes the parameter of the neural net in this case. This approach can be used to predict a system and therefore optimizes the reward generated by an agent. These researches have led to more robust and especially more flexible solutions, concerning changing tasks or environments.

Distributional Reinforcement Learning

Another way to improve robustness, often used in games, to cover many games within one algorithm is the so-called distributional reinforcement learning considered by [Lyle et al., 2019], [Bellemare et al., 2017] and [Bellemare et al., 2019]. The idea behind distributional RL is to model not just a single value function, but a distribution of values at every state. The optimal Bellman equation in this case is:

$$\mathcal{T}Q(s,a) := \mathbb{E}\big[R(s,a)\big] + \gamma \mathbb{E}_P\big[\max_{a'\in A} Q(s',a')\big]. \tag{3.7}$$

Based on this equation a distributional policy optimization can be performed. The original idea with a *Wasserstein metric* did not work for the optimization, therefore this part is performed with a usual *KL-Divergence*. The main part of the algorithm works based on tabular methods and gets the best performance by working with a distribution of over 51 bins (called: *C51*).

The algorithm on distributional approximate learning outperforms other algorithms like *DQN* tested on several games. The performance of this approach is very good in many different environments, which differs from other approaches where big differences occur. Also having one algorithm in many environments can be a scale of robustness where these algorithms perform very good.

In the original implementation, the *C51* algorithm is mainly based on tabular methods. Therefore, further research like Bellemare et al. [2019] extend this approach with linear function approximations (called: *S51*). The results of this approach are not always better than the original approach but has similar performances over several different environments.

3.3 Transfer Learning

One of the biggest challenges in machine learning is to perform a transfer learning where Taylor and Stone [2009] made important researches. Transfer learning means to learn solving a problem and applying it to a different but related problem. Or to start training in a domain and transfer it for further training into another domain, such that the agent already has prior knowledge. This can result in a better overall performance. Defining this problem means an agent has to learn in its source domain $\mathcal{D}_S$, where $\mathcal{D} = \{S, P(S)\}$ and the source learning task $\mathcal{T}_s$ with a state-space Y and a predictive function $f(\cdot)$ formulated as $\mathcal{T} = \{Y, f(\cdot)\}$. Defining the transfer learning problem means either the domain or the learning tasks have to differ from source to target as Lin and Jung [2017] described:

$$\mathcal{D}_S \neq \mathcal{D}_T \text{ or } \mathcal{T}_S \neq \mathcal{T}_T. \tag{3.8}$$

The approach of transfer learning is also shown in Figure 3.1. The main goal of transfer

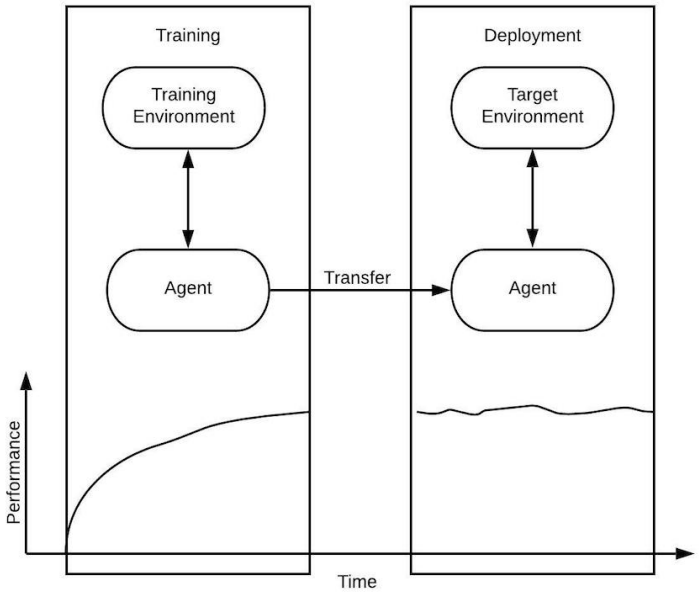

Figure 3.1: Schematic illustration of Transfer learning is shown in this figure according to Bjarre [2019]. The left side describes the training behaviour and the rising performance of the agent, while the right side indicates the deployment of the learned behaviour with almost constant performance.

learning is to perform the learning process in simulation and then fulfil the task in a real-world environment described in the next section.

3.3.1 Sim-to-Real

To achieve a good performance in the real world, while training just on simulated data, is a big challenge because usually in the simulation the whole system is perfectly defined without any uncertainties or noise. Transferring the policy of an agent from simulation to real world is called a *sim-to-real* transfer. These disturbances on the system can have a big impact on the performance and many research studies try to solve these problems as seen in the previous sections. In real-world approaches the system can slightly be different based on manufacturing tolerances or some forces that are not considered in the simulation. The problem of differences between simulation and the real world is called the *reality gap* in literature as described by Jakobi et al. [1995]. If joints have slightly more resistance in the real world than in the simulation, this can already can have a big impact on the performance of the agent. Another problem of the real-world application is noisy data. The sensors provide a state of the agent in the real world, but according to the training, the agent is not aware of noisy data after the training in simulation in perfect simulated conditions. Besides very various approaches to improve robustness of reinforcement learning some researchers as [Peng et al., 2018], [Sadeghi and Levine, 2017], [Andrychowicz et al., 2020], [Tan et al., 2018] and [Bjarre, 2019] deal with the problem. If the *reality gap* can be closed and an agent can be trained accurate just through simulation training, the development of robotics can be accelerated. Another motivation to perform an accurate *sim-to-real* transfer is, that the research gets a lot cheaper while just training in simulation. A lot of hardware dependencies and extra effort to build up the hardware for every training setup is not necessary anymore.

Domain Randomization

A very common approach performing transfer learning into the real world is called domain randomization, which is introduced by Tobin et al. [2017]. The purpose of domain randomization is to bring enough variability into the simulation data to cover the uncertainties that appear in the real world. Therefore very different aspects are randomized in the simulated data. As example domain randomization can be used in the object detection. Here various characteristics like varying number and shape of objects, position, texture, on the objects as well as random noise in the pictures and more sensor specific uncertainties can be randomized and then used for the training process. Randomization in general is very specific to the problem that is covered with the approach. It also can be performed by randomizing the system dynamics of a robot in the simulation. By changing the weights, lengths and other important system parameters for the robot during training, the domain randomization often is fulfilled as seen in [Peng et al., 2018]. Another variation of different randomization methods is the times when randomization takes place. In case of an object detector in every image, that is created for training, the object differs. In robot applications, the randomization

can take place in every step the robot has to perform as described by Andrychowicz et al. [2020] as well as after every episode that the agent has performed.

Looking at this approach, training a robot in a simulation and transfer the learned behaviour into a real world problem, a big issue is to simulate an accurate environment. Expecting the real world transition function as $p^*(s_{t+1}|s_t, a_t)$ this should be close to the simulation transition function.

$$\|\hat{p}(s_{t+1}|s_t, a_t) - p^*(s_{t+1}|s_t, a_t)\| \leq \mathcal{E} \tag{3.9}$$

For the randomization additionally a set of dynamics parameters μ is introduced that parametrize the dynamics of the simulation. The training process is performed with changing these parameters and in the same time maximize the expected return across several dynamic models ρ_μ leads to:

$$\mathbb{E}_{\mu\sim\rho_\mu}\left[\mathbb{E}_{\tau\sim p(\tau|\pi,\mu)}\left[\sum_{t=0}^{T-1} r(s_t, a_t)\right]\right] \tag{3.10}$$

The policy adapts to the variability in the dynamics and therefore the resulting policy should perform better, because it is generalizing the real world dynamics. To achieve this in practice often physic engines and simulators like *PyBullet* or *MuJoCo* are used.

Andrychowicz et al. [2020] as well as Peng et al. [2018] also used the *MuJoCo* set up to perform the simulation. They did not only randomize the system parameters of the robot but also added noise to the observations to get the simulation closer to the real-world setup. Both research papers observe the situation with cameras and therefore the robot can fulfil the task through this observation. In both cases, the result of the research was a working robot with good performance in the real world that just has been trained in simulations.

Closed loop control

Usually, reinforcement learning agents realize an open-loop control on the robot, which means that there is no feedback and it is typically realized with torque control. Since this approach is very sensitive against noise, in the research of Peng et al. [2018] a closed-loop control is used for the *sim-to-real* transfer. By using a *PID*-controller that controls the robot, instead of direct control, the agent more robustness against noise can be achieved. In the closed-loop control, the agent does not output torques to control the robot but instead gives the target position to the controller. Therefore the inverse kinematics of the system has to be calculated. The whole controlling process, therefore, gets more complicated and computationally expensive. Nevertheless, this approach as used in Peng et al. [2018] lead to the benchmark performance in *sim-to-real* approaches.

Robust Markov Decision Processes

In the research *Robust Reinforcement Learning for Quadcopter Control* Bjarre [2019] fulfills a transfer learning based on modelling an RMDP like described in Chapter 2.3. This approach of using a *PPO* algorithm and apply it on a RMDP is very similar to the approach used in this master thesis described in Chapter 4. To the best of my knowledge, this is the first attempt applying deep reinforcement learning on RMDPs in case of manipulation and locomotion tasks in literature.

The aim of this research is to train a quadcopter based on computer simulation and then transfer the policy to the real world. By making some assumptions to simplify the system it makes it possible to model it within an RMDP. The aim of this research, to model the problem as RMDP, is to make the system a deterministic system in continuous action space. The learning process is performed with a variation of the *PPO* algorithm. The uncertainties are modelled with varying system parameters that are implemented with a linear interpolation within a value range of the parameters. By using this approach a transfer to the real world is successful and also concerning robustness testing with different agents in their environments this approach achieved acceptable results as described in Bjarre [2019].

Other Approaches

Another approach to model an environment close to the real world is described by Sadeghi and Levine [2017], where a CAD-model of the real world is taken for the training. Building a whole hallway as a model made it possible to fly through this hallway without training it in the real world. In this case for collision avoidance, a deep reinforcement learning algorithm is used. Although this approach is working without a collision this approach is very complex and computational costly to do with different or larger environments. In addition to that the agent, in this case, a quadcopter, is just able to move very slowly through the hallway because the camera data it gets in the real world had noise, which was not considered in the training process.

Also, the described method of learning an environment as described in Chapter 3.2 is used to perform a sim-to-real approach by Golemo et al. [2018]. In this case, a recurrent neural net is trained on the different trajectories in a simulated and in a real-world application. Afterwards, this neural net is used to augment the simulator to perform a training close to the real-world scenario. Also, this approach led to good performances, but in contrast to the other approaches knowledge about the real world is necessary.

3.3.2 Other Transfer Learning

Not just bringing an agent's performance from simulation to real-world as described in the previous section, but also performing more tasks or transfer between different tasks

as described in the overview of Taylor and Stone [2009] is part of transfer learning. Therefore, already a different reward function of the same task in the same environment can be a form of transfer learning. This approaches of transfer learning are done already a lot earlier than the sim-to-real tasks. Selfridge et al. [1985] for example already changed the systems to perform a form of transfer learning.

Not just transferring the agent's policy to another task but also performing the same task with the same reward function transferred to a slightly different environment is called transfer learning. Considering the transfer from just simple theoretical dynamics implementation to a more complex physics simulation like *PyBullet* can be a method of transfer learning. The physic simulation is the next step closer to reality because parameters like friction can be considered. Also, the dynamics and control are often fulfilled in another way in a different implementation of the environment. This step represents the first step to close the reality gap from simple simulation to the real world.

4 Methods

This chapter is dedicated to the methods, that are applied in this thesis. At first, the state of the art algorithm proximal policy optimization (PPO) is introduced and transformed to the RMDP approach called PPO-RMDP in this thesis. At the end of this chapter, the metrics used for evaluation are described and discussed.

4.1 PPO on RMDPs

In this thesis, the PPO algorithm is implemented and applied to RMDPs. The PPO is working in discrete and continuous action spaces. Modern policy gradient methods usually make use of local approximations of the policy performance around μ. This leads to the generalized objective with Φ_T being a generalized estimator as described by Schulman et al. [2015]

$$J(\pi_\theta) = \mathbb{E}_{s_t \sim \rho_\pi, a_t \sim \mu}[\frac{\pi_\theta}{\mu}\Phi_T] \tag{4.1}$$

assuming that the state distribution ρ_μ is close to the state distribution ρ_{π_θ} of the policy π_θ. To hold the assumption it has to be ensured that the policy update is not excessively large. Therefore in their work, Schulman et al. [2017] clipped the objective function into a penalized policy beyond the probability ratio $\lambda = \frac{\pi_\theta}{\mu}$, to work in a similar behaviour than Kurutach et al. [2018]. As a result, the new PPO objective function can be written as

$$J_{clip}(\pi_\theta) = \mathbb{E}_{s_t \sim \rho_\pi, a_t \sim \mu}\Big[min\Big(\lambda \cdot \hat{\Phi}_t, clip(\lambda, 1-\epsilon, 1+\epsilon) \times \hat{\Phi}_t\Big)\Big] \tag{4.2}$$

with ϵ controlling the interval of the policy updates and the estimated advantage function $\hat{A} = \Phi_T$. Clipping in this equation means that the policy updates which differ from $\lambda = 1$ are adjusted by limiting the change of the objective function to the interval $[1-\epsilon, 1+\epsilon]$. The first term inside the $\min$ function is $\lambda \times \hat{A}_t$, which describes the unconstrained objective function of 4.1. The other part inside the minimization is the clipping term $(\lambda, 1-\epsilon, 1+\epsilon) \times \hat{A}_t$ which forces the objective function to be in the defined interval of ϵ. Through the minimum operator, the update chooses the lower bound of the two described terms. Therefore, the learning ratio λ is limited to at most $\lambda = 1+\epsilon$ for positive advantages and render the ratio to be at least $\lambda = 1-\epsilon$ for negative advantages. An accurate estimation of the expended gradient just requires many clipped terms. In Figure 4.1 the clipping is visualized for positive advantages with $A > 0$ as well as for negative advantages with $A < 0$.

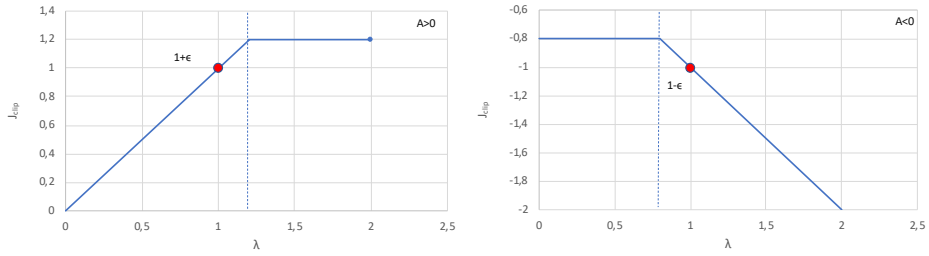

Figure 4.1: In this figure a clipped loss function is plotted over λ according to Schulman et al. [2017]. The red dots indicate the starting point for the optimization for positive advantages (left) as well as for negative advantages (right). The dashed line indicates the upper limit of the policy changes to prevent large changes.

To ensure the performance in order to compute variance-reduced advantage function estimators, a generalized advantage estimator (GAE) as described by Schulman et al. [2016] is used. If the estimation of V is accurate to the true value function, at every time step t the advantage function equals the TD residual term $\gamma_t^V = \hat{A}_t$ with the TD residual: $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$. Considering the TD residual over n-steps the advantage can be determined as follows:

$$\begin{aligned}\hat{A}_t &= -V(s_t)r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} + \gamma^{T-t}V(s_T) \\ &= \sum_{k=0}^{n-1} \gamma^k \delta_{t+k}^V\end{aligned} \tag{4.3}$$

Here the advantage function is taken over a whole trajectory similar to the TD-λ method as described by Sutton and Barto [2018]. In this case the single advantage functions can be exponentially weighted by a factor λ. Further, taking the advantage functions over all trajectories into account $\hat{A}_t^{GAE}$, it can be computed as follows in an infinite horizon according to the Equation 4.3:

$$\hat{A}_t^{GAE} = \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k}^V \tag{4.4}$$

Similar to TD-λ methods the parameters γ and λ contribute a trade-off between variance and bias of the estimation. On the one hand factor γ reduces the variance but on the other hand at the same time it causes a growing bias, in case the value function is inaccurate. In practice it is common to normalize the advantage function by:

$$\bar{A} = \frac{(A - mean(A))}{std(A)}. \tag{4.5}$$

This equation holds over the entire training batch with $A = \hat{A}_t^{GAE}$ in this case.

The implementation of the PPO algorithm is shown in pseudo-code in Algorithm 1. As a first step, the policy parameter π_θ and an initial state-value estimation $\hat{V}_w(s)$ of the algorithm are initialized. Then, after K cycles of the policy optimization, the optimized policy π_θ is assigned to the behaviour policy μ. In every step i, $\mathcal{D}$ trajectories are rolled out using policy μ for at most T time steps. Here the implementation differs from the standard PPO. Taking the RMDPs and therefore the uncertainties into account, the calculation of the values at each step is performed according to the robust Bellman equation described in Chapter 2.29. To represent the uncertainties of the RMDP, each step is sampled to cover a distribution over the system dynamics. Because the agent always acts in deterministic environments, sampling the next step of the environment and therefore predicting the next value is possible. As a result, a distribution of values is given for each step in the system. Acting according to the robust Bellman operator the lowest value is taken for the further approach. For each transition pair (s_t, a_t) a discounted reward-to-go of the current state is calculated. The next step of generating new observations is fulfilled with the standard parameters. These parameters are used because only the values have to be minimized according to the PPO-RMDP. After having estimated the advantage function according to the method described in this Chapter, the policy is updated according to the PPO objective with an ADAM optimizer. Afterwards, the value function is fitted by using a regression on the mean squared error via a gradient descent algorithm as described in Section 2.22. Based on different environment and learning behaviours, the number of K differs as well as the steps performed in each cycle.

PPO-DR

According to several *sim-to-real* approaches in further researches, a robust version of the PPO algorithm is also used in this work. The robust variant is based on domain randomization. The difference to the normal proximal policy optimization algorithm is, that there the environment is implemented with uncertainties. In contrast to this approach in the PPO-RMDP in this work, the steps of the environment are performed in the standard environment. In the case of domain randomization, there is no fixed environment and thus the steps through the episodes to roll out the trajectories are performed with varying environments in every step. Therefore the randomization with up to $+/-20\,\%$ of all system parameters is done. Through this technique a lot of *sim-to-real* approaches like [Peng et al., 2018] and [Andrychowicz et al., 2020] are realized. This approach is ought to give more robustness against problems appearing in the real world and is called the PPO-DR within this work.

Position Based PPO

Another option to improve the robustness besides the PPO-DR is to perform the PPO algorithm in a position based control. In the standard PPO approach, the neural net-

Algorithmus 1 : The implemented Proximal Policy Optimization algorithm on RMDPs

initialization of μ_k with policy parameters θ_0 and an initial value function parameters $\hat{v}_0$

for $k = 0, 1, 2, \ldots$ **do**

sample trajectories $|\mathcal{D}| = \{\tau_i\}$ with policy $\mu_k = \theta_k$ at each step with:

$$v_{t,\mu} := \min_{(r_t, P_t) \in \mathcal{U}_\pi} (r_{P_t})$$

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates $\hat{A}_t$ using $A = \hat{A}_t^{GAE}$ based on the current value function.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\Big(\frac{\pi_\theta(a_t|s_t)}{\mu_{\theta_k}(a_t|s_t)} A^{\mu_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\mu_{\theta_k}}(s_t, a_t)) \Big)$$

with a stochastic gradient ascent with Adam optimizer.

Fit the new value function pramaters with a gradien descent algorithm:

$$\hat{v}_{k+1} = arg\min_{\hat{v}} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \Big(V_{\hat{v}}(s_t) - \hat{R}_t \Big)^2$$

end

work decides which action to use at the given state in order to get the best results. Therefore, the actions can be seen as forces or torques directly acting at the robot. In contrast to this approach, Peng et al. [2018] made the neural net give the algorithm a target position instead of an action. After computing the position for the next step, a controller, like for example a PID controller, controls the step of the robot. Therefore, additionally, the inverse kinematics of the environment is needed. Performing this step and therefore control the robot with a controller instead of the torque calculation and controlling of the robot directly the torque based control is modified to a position based control. The aim of this approach is to gain more robustness against noise because the controller is not that sensitive against the fast changes.

This approach can be performed with the standard PPO as well as with the PPO-RMDP. Therefore, both possibilities and the impact is tested in this research.

4.2 Metrics

Robustness is a very big topic in the whole field of machine learning. Big questions are in this case what is robustness and how can it be measured in practice. Due to the fact that there is no common best practice to handle robustness in general, different aspects of robustness are used for the evaluation of the algorithm in this work. By taking different metrics, described in this section, more different aspects of robustness are covered and therefore a more complete view on robustness is provided. In general different influences, such as changing system parameters, adding noise in the action- and observation space and changing the whole environment, as shown in Figure 4.2 are possible parameters in an RL environment to test the robustness.

4.2.1 Dynamic System Parameters

One main aspect of robustness is that the agent should also provide good results even when the system dynamics, like for example the system parameters such as lengths and weights, are different while performing the same task. This metric is motivated to compensate for manufacturing tolerances or slightly different real systems based on inaccuracy or misspecification. Taking the approach of changing system parameters is also done in [Mankowitz et al., 2018]. As suggested in Mankowitz et al. [2018] research, in this thesis also the system parameters are changed during the tests of the learned policy. Different variations of lengths and weights, like the so-called *options* in the former research, are used for the performance evaluation. In two dimensional problems, the performance can be visualized with heat maps to show the chosen parameters and the resulting reward as shown in Figure 4.3

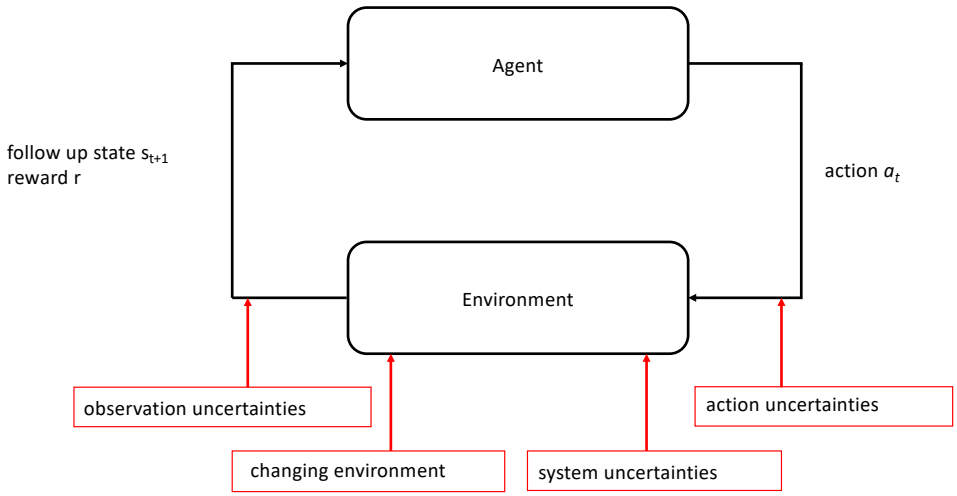

Figure 4.2: This figure shows the normal model of an RL agent acting with its environment. Highlighted in red are the influences, such as changing system parameter, noise in action- and observation space and changing the whole environment, are shown.

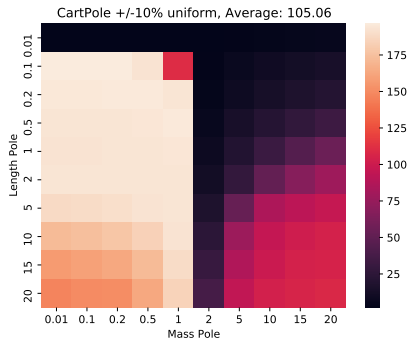

Figure 4.3: In this figure a heat map for the performance evaluation of CartPole trained with a PPO-RMDP. The system uncertainty, in this case, is a uniform distribuiton with $+/-10\%$. On the x-axis the different weights, and on the y-axis, the different lengths are shown. The bar on the right describes the colours that are used to indicate the reward-score.

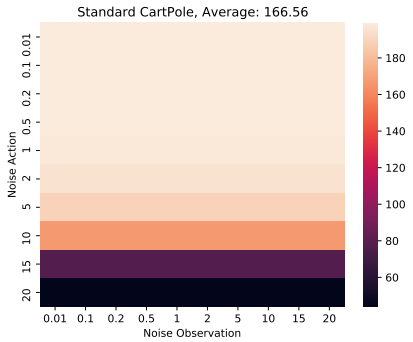

Figure 4.4: In this figure a heat map for the performance evaluation of standard PPO in *CartPole* is shown. On the x-axis the different noises on the observations, and on the y-axis the different noises on the actions are shown. The bar on the right describes the colors that are used to indicate the reward-score.

4.2.2 Noise

The evaluation of how the performance changes when noise is observed in the system, is another parameter that helps to evaluate the robustness of the algorithm. This method is motivated by the use of reinforcement learning agents in real-world environments. Adding noise to the agent performing its learned policy is as a performance metric is first done by Wang et al. [2019]. With this approach robustness of an algorithm against noise can be evaluated.

The performance can be evaluated by analyzing the averaged reward over the different amplitudes of noise. According to the metric based on changing system parameters, the results also can be visualized in form of a heat map where one axis describes the noise on the observations while the other axis describes the noise on the action as seen in Figure 4.4. Therefore the noise can be applied to the action as well as on the state space with this method.

4.2.3 Metrics of Transfer Learning

Another briefly discussed question concerning transfer learning is how to measure the success of the transfer. Transferring a policy of an agent will usually not lead to the same reward as in the original environment. All in all, if the agent fulfils the task the transfer learning counts as successful. To evaluate the quality of the transfer there are several metrics described by Taylor and Stone [2009] and Da Silva and Costa [2019], some of them are illustrated in Figure 4.5.

The most common metric is the total reward or the total average reward over varying system dynamics or with added noise if the transfer should be considered under a robustness perspective. This implies the total reward the agent gains by acting according

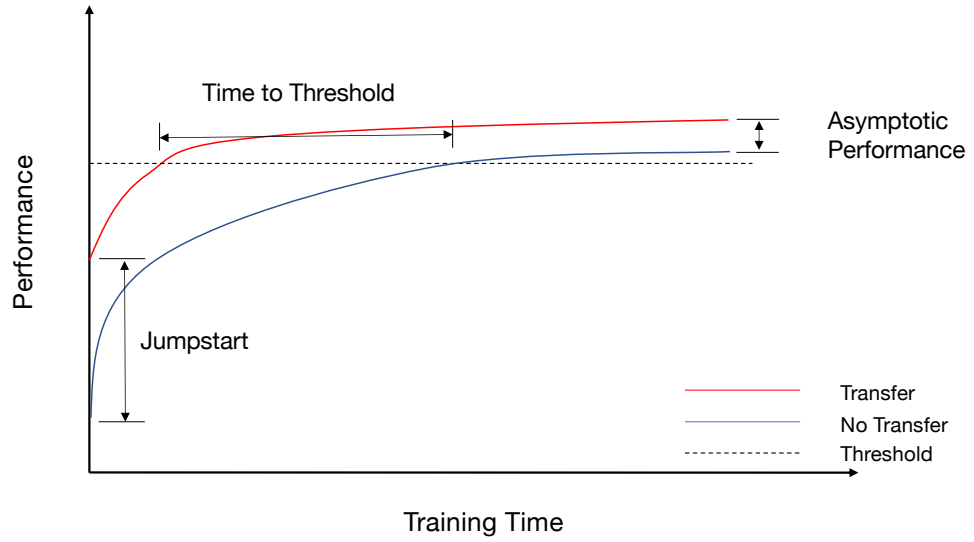

Figure 4.5: Different metrics are possible to evaluate transfer learning according to [Taylor and Stone, 2009]. In this figure the *Jumpstart*, *Time to Threshold* and *Asymptotic Performance* are shown in the example. Also the total reward is shown by the area under the learning curve.

to the learned policy in the environment. The difference between the total rewards of the agent in different domains is then called the transfer ratio. Since in this work we do not have a focus on the time until convergence, the transfer ratio is calculated based on the comparison of performance with, shown as PwT, and without transfer, shown as $PwoT$, in the environment.

$$T = \frac{PwoT}{PwT} \tag{4.6}$$

During training also the maximum asymptotic performance can be used as a metric to show how good the agent is able to act in the environment.

According to the timing and speed of learning of an agent, there are several more metrics to use. In the case of transfer learning, it is also possible, that the learning process of the agent goes on after transferring it to the new environment. The initial performance in the new environment should be better than the performance of an agent without prior training or knowledge about the RL problem. This advantage in the learning behaviour is called *Jumpstart* and is also shown in Figure 4.5. Another metric mentioned is the time to cross a predefined performance level. The performance level often represents a minimal needed performance to solve the task of the agent in the environment. This is also an important indicator of how good the transfer worked because in some applications also the time and not just the overall performance is important to evaluate the performance of the learning process.

Due to the fact that time is not always relevant to evaluate the learning progress, but the overall reward is often the most significant indicator, the total average reward is taken as a metric to evaluate the learning behaviour without performing any further learning iterations in the target domain in this research.

Transferability

Another metric concerning the robustness of a reinforcement learning algorithm is the problem described in Section 3.3. Therefore, the training is done in a simpler implementation of the environment based on dynamics and the derivations in *PyTorch*, as described in 5.1. The transfer is then performed to an implementation that is closer to the real world in *PyBullet*. Transferring the behaviour of the agent into the physic simulator environment the task is made more complex due to the more complex environment. Because transfer learning is one of the biggest challenges in reinforcement learning and often leads to problems, this approach is a useful way to evaluate the robustness of an algorithm. Part of the evaluation of transfer learning is the normal performance of the agent in the after the transfer as well as the combination with noise and system parameter changes. Not just the average reward but also the transfer ratio can be taken into account to evaluate and visualize the result. In case of transfer learning, the PPO-DR is the benchmark according to previously performed researches as mentioned in Chapter 3.3. The evaluation of the transfer learning is made with the PPO-DR, the new PPO-RMDP and also the standard PPO.

5 Experiments

In this chapter, the different experiments that are done in order to analyze the robustness of the different robust PPO variations concerning different evaluation metrics are described. First, the different environments are introduced. Afterward a more detailed analysis of the experiments is conducted: In this thesis, three different main experiments are performed based on the metrics mentioned in Section 4.2. Then, slightly different versions of RMDPs are considered and the impact on the robustness is analyzed. All experiments are performed with the PPO-RMDP, the robust PPO based on domain randomization (PPO-DR) and the standard PPO.

5.1 Environments

The previously described algorithms are used to train an agent with the aim of solving a reinforcement learning problem. Therefore, the agent is set in an environment and learns how to fulfill the task while interacting and getting rewards. Within this thesis, several different environments are taken into account to make an analysis concerning the robustness of the algorithms. Also, an important point concerning the environments is, that the system dynamics are available and therefore the behaviour of the system is deterministic. For this reason modeling, the problem with an RMDP can be realized by implementing the uncertainties as distributions over system dynamics. In this research two types are used to implement the environments.

First, the environments are realized through ordinary differential equations (ODE) implemented in *PyTorch*, an open-source machine learning framework that works with Python. In this case, the environment dynamics are implemented based on the differential equations of the systems and their derivations.

The second way to implement the environment is by using *PyBullet*. This is an open-source physics simulator. Simulation of the RL problems in the physics simulation is the first step to close the *reality gap*. The preparation of the robot as well as the consideration of all physical parameters that occur in the system such as friction and other parameters make it more complex. Furthermore, this implementation method distinguishes itself from the other one through different control mechanisms of the environments, although the task, observations, states and actions are defined the same way in both types. Two different environment implementations are used to perform transfer learning as metric described later in this Chapter (4.2.3). Additionally, all environments have a maximum number of steps where the rewards of a whole trajectory

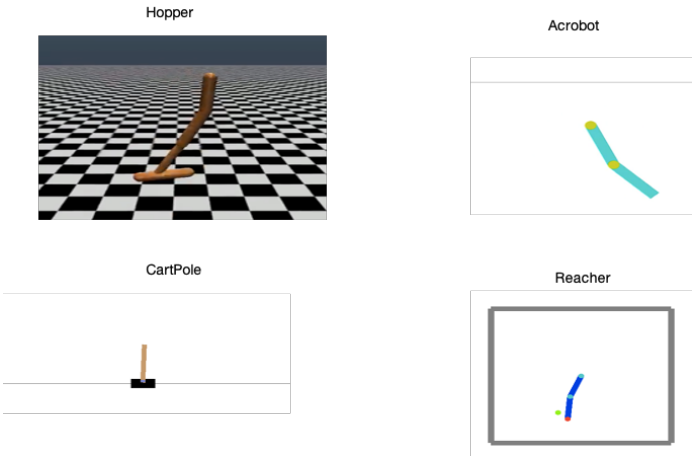

Figure 5.1: In this figure all different environments used for the robustness evaluation during this thesis are shown which are: In the upper left corner *Hopper*, below it *CartPole*, in the upper right corner *Acrobot* and in the bottom right corner *Reacher*.

are generated. In the following section, the used environments are described based on the researches in which they were published. Also, screenshots of all environments are shown in Figure 5.1

CartPole

The so-called *CartPole* problem was first described by Barto et al. [1983]. It is also known as an inverted pendulum, a pendulum with a center of gravity above its pivot point. The goal is to keep the pole in a balanced position, by applying forces to the pivot point (cart). For each time step, the pole is in a position upright and not falling down too fast, it gets a reward of $+1$. The episode ends when the pole gets out of the range of $+/-15$ degrees around the center and when the cart moves more than 2.4 units away from the center. Each episode is stopped after 200 time steps, therefore the maximum reward through one episode is 200. The system holds balance by applying a force to the cart in a range of -1 to 1. According to this force, the *CartPole* problem is distinguished into two versions: the discrete and the continuous action space. The RMDP realization, in this case, is performed by making distributions over the system dynamics: pole length, pole mass, cart mass. Since the cart mass, based on the first experiments, has no impact on the robustness behaviour, the uncertainties are modelled with the remaining two parameters.

For this environment, there is a derivation as well as a physic simulation implementation. Both, the discrete and the continuous version are available. Therefore, the evaluation can be performed on all presented metrics in Chapter 4.2.

Hopper

The next environment that is used for the experiments is called *Hopper* and is implemented according to [Erez et al., 2011]. In this environment, the robot looks like a leg and the goal is to move forward as fast as possible. It is moving in a two-dimensional world, therefore it just can go forwards and backward. The leg robot is built up of three main parts: the foot, the lower leg and the upper leg with joints between the parts. By learning to move forward the robot gains reward in this environment. The robot is moving while hopping in one direction. This behaviour is realized through different torques at the joints of the robot leg. The aim is to move the robot's center of mass with a horizontal velocity while keeping its vertical position around 1. The farther the robot gets the more reward it gains. The episode is limited to 1000 time steps. Actions are modelled in continuous action space. Actions are modeled in the way that they can act on every joint: the foot joint, the thigh joint and the leg joint. Therefore, the action is a three-dimensional value. Modelling uncertainties in this deterministic environment is performed by building uncertainties in the properties of the different parts.

Reacher

The *Reacher* environment is implemented according to [Li and Todorov, 2004]. It is a two-segmented robot arm that is fixed on one end. The goal of the robot arm, which moves in a two-dimensional plane, is to reach a randomly generated target within the range of the robot. The robot is fixed on a horizontal plane, therefore gravity does not have an impact on the robot's behaviour. By reaching the goal with the tip of the robot arm, the reward increases. Episodes are limited to 150 time steps in this implementation. The robot is controlled through the forces at the joints and therefore has an action dimension of 2 in a continuous action space between -1 and 1. The control takes place as a direct torque control in the torque based approach. To create uncertainties this environment was modelled as an RMDP, in which the lengths of the two parts as well as the mass was changing. Transfer learning is also enabled in this environment as well as the comparison of torque based and postition based control.

Acrobot

The next environment, used for the experiments in this thesis, is the *Acrobot* environment. It was first presented in [Sutton, 1996], but is implemented according to [Geramifard et al., 2015]. Similar to the *Reacher* environment the *Acrobot* also is a two-segment robot arm with two joints. In contrast to the other environment, the arm is not fixed on a horizontal plane and therefore initially hangs down at the starting point due to gravity. The goal is to swing up the end of the lower part to a given height. Crossing this height the agent earns a positive reward otherwise it gets a negative reward. The perfect behaviour, in this case, is to swing the arm up and balance it above

the target like an inverted double pendulum. Episodes in this environment are limited to a maximum of 500 steps per episode. The action space is also continuous in this deterministic environment. Similar to all the other environments again the uncertainties of the RMDP are modelled through distributions over system parameters.

5.2 Algorithm Analysis

Before deducting the robustness evaluation of the experiments in the various environments, different approaches have been tried to achieve the best performance learning with deep RL in RMDPs. The aim is to follow the most promising approach according to RMDPs in deep RL. All of these methods are initially tested with the *CartPole* environment, due to the fact that this is the fastest way in terms of computational costs. If these approaches give promising results, further experiments will be performed on the algorithms. Some approaches are combining different approaches. The algorithms in detail and how the performance is evaluated are described in the following section.

5.2.1 Worst Case Expectations

The classical idea of RL with RMDPs is to consider the transition uncertainties in the case to minimize the expected value at each state in each step of the algorithm. Therefore, the RMDP is acting with respect to the worst-case expected scenario. Afterwards, the next step within the standard system dynamics of the environment is performed and the action, as well as the observation, are saved in the replay buffer. In the following step, the variation of the previously described PPO-RMDP is implemented. Instead of taking the standard observation and action for the replay buffer, the corresponding worst-case observation and action are saved in the buffer. Therefore, the algorithm not just acts with respect to the worst-case but acting on the worst-case scenario. Using this approach and always acting on the worst possible transition can lead to increased robustness than the PPO-RMDP achieves. Due to the fact that this approach has a high probability of acting in a very conservative way. This means the robustness of the algorithm is good but the overall performance and the peak performances is much worse compared to the other approaches. The evaluation of this approach is performed via the system parameter change and is evaluated with respect to the total average score.

5.2.2 Domain Randomization

The actual state of the art approach for transfer learning in RL is domain randomization as for example Peng et al. [2018] used for their sim-to-real transfer. The idea is to combine the approach of learning in RMDPs and also perform domain randomization in the same algorithm. After minimizing the value over the uncertainty model of the

system, in the PPO-RMDP the nominal environment is taken to perform the next step in the environment. In contrast to this behaviour, a randomly chosen system within previously defined bounds of the uncertainties from the RMDP performs the next step and therefore performs domain randomization. This combines the idea of the training on RMPDs and the PPO-DR. Evaluation is done with the changing system parameters metric in the *CartPole* environment.

5.2.3 Impact of Distribution Type

The uncertainties of the RMDP can be modelled in different ways. Because of that point, one leading question, in the beginning, was: how does the type and spread of distribution influence the result towards robustness. Therefore, different types of distributions (e.g. linear, Gaussian and logarithmic distributions) have been tested. All of those with different spreads starting from $+/-5\,\%$ up to $+/-40\,\%$ tolerances. All of the distributions are modelled with a distribution dimension of 100. In this case, the uncertainty of the RMDP is represented in 100 bins for each system parameter. Evaluating this against the parameter change metric of the environment lead to different average scores. Taking these scores and also the heat maps for evaluation the influence of the type and the spread of the distribution concerning robustness and performance is done.

5.3 Robustness Analysis

In the following section, different methods to evaluate the robustness of the RL agent are described.

Robustness Against Changing Parameters

As a first experiment, the robustness against changing system dynamics is tested. Therefore, all masses and link lengths of the robot have to be changed. An array of different lengths with following values $[0.01, 0.1, 0.2, 0.5, 1, 2, 5, 10, 15, 20]$ is defined. After training the algorithm according to one of the PPO algorithms the environment gets no uncertainties but is defined through different parameters to evaluate the performance of the agent. Therefore in each environment a different amount of parameters has to be modified, in *CartPole* for example there is just one single link, but in *Hopper* there are three links to handle. Also for each reinforcement learning problem in this case the learning curve has to be converged before the evaluation step. This results in a different amount of training steps for each environment.

In addition to the evaluation parameter, there are multiple plays of the RL problem in order to make the results more reproducible. For each setup e.g. every constellation of system parameters, 50 episodes are played and the average is taken. This is done

because in the initialization the robot is often created randomly within a parameter area. Through this random part in the initialization and also the random part of the exploitation of the agent during the learning process, the agent's learned policy can differ and as such lead to slightly different results. Therefore the whole training and evaluation loop is done with 10 (for very computational expensive environments) up to 50 runs per learning problem and setup.

Robustness Against Noise

This experiment is performed very similarly to the experiments described in the previous section. In this experiment, noise is added on the observations as well as on the actions for the evaluation process. The noise is modeled as so-called *white noise* which means it is a Gaussian distribution with mean $\mu = 0$. To differ the amplitudes of the noise different standard deviations σ of the Gaussian distribution are taken for the evaluation. The used standard deviations are filled with following values: $\sigma = [0, 0.001, 0.01, 0.03, 0.05, 0.07, 0.1, 0.15, 0.3, 0.5]$. The rest of the reproducible evaluation process is done according to the previously described practice. To evaluate it properly, for each action - observation pair the corresponding noise is performed to get to the heat map as shown in Figure 4.4.

The noise as well as the changing system dynamics experiments are performed with all of the described environments. This is done to get an impression of the influence of algorithms on robustness according to different influences. This wide evaluation is done to make the results more meaningful according to the robustness of an algorithm.

Transfer Learning

In this experiment, just the *CartPole* and *Reacher* environment are tested, due to the fact that both of them where available in two environment implementations. The reinforcement learning problems have been trained in the ODE implementation of the environment and have been transferred to the physics simulator implementation because this one is the more complex one for the agent. Also, the learning time in the first environment is significantly smaller, therefore transfer learning leads also to an improvement in training duration. The evaluation of transfer learning is made according to both previously described experiments. Therefore, for each transfer learning problem, two different results are generated. The whole evaluation process also includes 50 episodes in every environment setup and also up to 50 runs over the whole learning process to generate reproducible results.

Position Based RMDP

In addition to the robustness evaluation of the different PPO approaches another method is evaluated in this master thesis. The idea of the new approach is to com-

bine the PPO-RMDP described in 1 with the approach of a position based controlling described in 4.1. Therefore, the values are calculated according to usual RMDP behavior but actions are calculated differently than before. In contrast to calculating a torque based action, the actor-network in the algorithm now has to calculate a target position. The action then is fulfilled by the position based control via a PID-controller.

Due to the complex implementation of this approach, it is just performed in the *Reacher* environment due to the availability of position-based control. In this case, changing system dynamics as well as adding noise is evaluated for the position-based RMDP behaviour. To have a benchmark of how good position-based controlling for RL approaches is in general also experiments with the PPO-DR and a standard PPO are done and evaluated.

6 Results

In the following chapter, the results of the experiments are described and analyzed. First, the PPO-RMDP is analyzed and afterwards, a robustness benchmark of the PPO-RMPD, PPO-DR and the PPO is provided.

6.1 Algorithm Analysis

In the following section, the results of the experiments to find the best PPO-RMDP setup are described and evaluated. As a reference for the analysis of the PPO-RMDP algorithm, the result of the standard PPO in the *CartPole* environment is shown in Figure 6.2.

The average score of 102.22 over all system changes and runs is used as benchmark for performance and robustness to evaluate the different approaches. Significant in this case is the bad performance with pole mass 2. This is an effect caused by the *CartPole* implementation of *OpenAI*.

6.1.1 Distributions with the PPO-RMDP

At first the results over different distributions are shown in Figure 6.1.

The x axis in this case distinguishes the different distributions compared and the average scores over all runs are shown on the y axis. Considered distributions are a uniform distribution, a Gaussian distribution and a log-normal distribution always around the standard parameters of the system with different spread or standard deviations. Significant is, that there is no big difference between different types of distributions. The main difference is the width of the distributions. The best performance of 107.87 is based on the smallest uniform spread of $+/-5\,\%$ and the worst performance is with the biggest spread from 0.1 to 10. The performance heat map of the best example is shown in Figure 6.3.

The effect of using a greater spread can be seen very good with the Gaussian distributions with the mean μ of the nominal value. The performance drops significantly from 106.23 with a standard deviation $\sigma = 0.1$ compared to a $\sigma = 0.3$ down to a performance of 101.42. It can be seen that also the log uniform distributions do not gain more performance because of the even bigger spread and only leads to an average performance of 98.24.

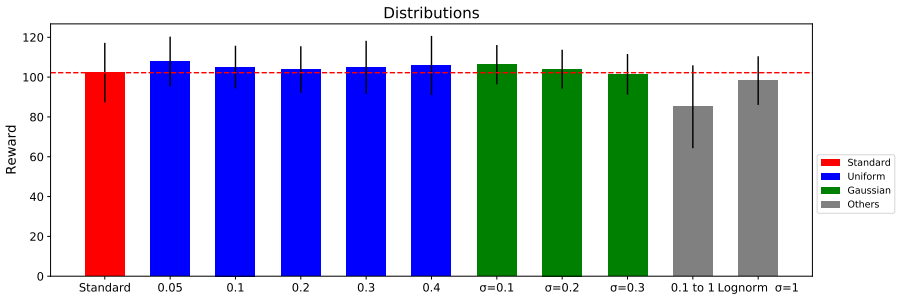

Figure 6.1: Average rewards of the trained agent in the *CartPole* environment due to changing system parameters is shown. The standard deviations over all runs are shown with the black errorbars in the figure. On the x axis, the different distributions of the RMDPs are shown. The y axis represents the average achieved score over all disturbances and runs.

Because of the results based on the *CartPole* environment for the PPO-RMDP, just random uniform distributions are taken into account. Due to the fact that the spread of the distributions has an impact on the results, as seen in the $+/-40\,\%$ example, always more different distributions are tested in the following experiments. Based on the results of further experiments as described in the Chapter 6.2 one can say that each environment can be optimized with its own distribution bounds. In general smaller bounds lead to better performances than very large spreads of a distribution does.

6.1.2 Worst Case Expectations

By using the worst expectation according to the minimization problem of the value used in the PPO-RMDP, a very conservative performance is the result. The average of 91.04, shown in Figure 6.4, is significantly lower than the performance of the same distribution with the PPO-RMDP approach achieving an average of 105.05. Both algorithms are tested with a uniform RMDP distribution of $+/-10\,\%$ to give a comparable result. Good to see in this heat map is the robust behaviour, in the region over a pole mass of 2, which is comparable to the best case distribution shown in Figure 6.3. Therefore the robustness of this approach is good but the overall performance is too bad to take this approach into account.

This is called a conservative learning behaviour as described in previous researches, which has been mentioned in Chapter 3.1. The problem of taking always the worst case as next steps lead to a much worse overall performance than taking just the worst probability into account of the system as it is done in the other approach. Even if the robustness increase with this approach is good, the overall performance is too poor. As a result, this algorithm does not fit for further experiments due to the very conservative performance.

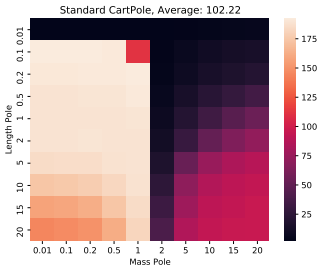

Figure 6.2: A heat map for the performance evaluation of *CartPole* with a standard PPO is shown. On the x-axis the different masses, and on the y-axis the different lengths are shown. The bar on the right shows the colors that indicates the reward-score. In the heading of the figure, the overall average is shown. All values are averaged over 50 runs.

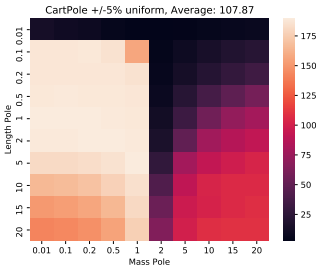

Figure 6.3: A heat map for the performance evaluation of CartPole with the PPO-RMDP and a uniform uncertainty of $+/-5\,\%$ of the standard parameter. On the x-axis the different masses, and on the y-axis the different lengths are shown. The bar on the right indicates the reward-score. In the heading of the figure, the overall average is shown.

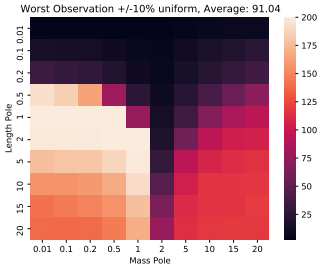

Figure 6.4: Heat map for the performance evaluation of CartPole with the worst case expectations and a uniform uncertainty of $+/-10\,\%$ of the standard parameter. On the x-axis the different masses, and on the y-axis the different lengths are shown. The bar on the right indicates the color that indicates the reward-score. In the heading of the figure, the overall average is shown.

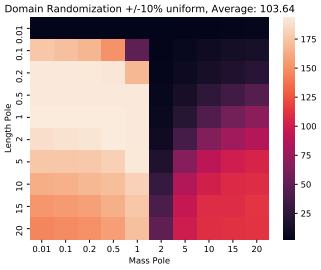

Figure 6.5: Heat map for the performance evaluation of CartPole with domain randomization in a PPO-RMDP modeled with a uniform uncertainty of $+/-10\,\%$ is shown. On the x-axis the different masses, and on the y-axis the different lengths are shown. The bar on the right shows the colors that indicates the reward-score. In the heading of the figure, the overall average is shown.

6.1.3 Domain Randomization

Combining the PPO-RMDP with domain randomization to improve the robustness as often seen in transfer learning problems, leads to the result shown in Figure 6.5. To make the result comparable to the results shown in Figure 6.4 and the standard approach, the same distribution is used for this experiment. In comparison to the previous result, this approach outperforms the worst-case algorithm, with an average reward of 103.64 significantly. In comparison to the PPO-RMDP approach, the additional domain randomization leads to worse overall performance and does not show a benefit in robustness compared to the heat map shown in Figure 6.3.

Taking the additional robustness improvement method of domain randomization, therefore, does not further improve the robustness of the PPO-RMDP algorithm, but leads to a more conservative overall performance. Because of this behaviour, the combined approach is not used for further experiments only the comparison of domain randomization PPO-DR, the benchmark on the robustness of the PPO, and the PPO-RMDP are compared in further experiments.

6.2 System Parameters

The robustness against different, not previous defined, system parameters are the first part of the robustness evaluation of all tested algorithms. In Table 6.1 the average score of the best performing distributions of every algorithm is shown. Each two rows in the Table represents an algorithm: first is the standard PPO, second is the PPO-RMDP approach and third is the PPO-DR. The firt row represents the average score and the second shows the standard deviation over all tested runs. Each of the five different environments is shown this table.

Table 6.1: This table shows the best average scores with system uncertainties of all tested algorithms in the experiment environments. After the average score the standard deviation is shown in braces.

Algorithm	CartPole	Hopper	Acrobot	Reacher Torque	Reacher Position
PPO	102.22	49.18	**-352.47**	-8.63	**-8.85**
STD	(14.98)	(13.78)	(7.89)	(2.21)	(2.78)
RMDP-PPO	**107.87**	51.43	-353.01	-8.63	-8.95
STD	(12.46)	(11.26)	(10.73)	(6.68)	(7.34)
PPO-DR	105.26	**57.8**	-359.68	**-8.61**	-8.93
STD	(11.9)	(8.35)	(6.7)	(2.03)	(3.57)

Significant in the results is, that in each environment, except of *Acrobot*, at least one robust approach of the PPO algorithm outperforms the standard algorithm in average scores with lower standard deviations. In the remaining four environments, where

a robustness increase based on the average reward is reached, both, the state of the art algorithm based on domain randomization and the PPO-RMDP lead to the best performance in two of the environments. This leads to the assumption, that both algorithms can improve the robustness against changing parameters depending on the environments.

The PPO-RMDP in this case has the best average reward of 107.87 in the *CartPole* environment, which is the most simple and the approximately linear working environment in this research. It also provides better robustness in the *Reacher Position Based* environment, where the overall performance of the agents is too bad to call it a significant improvement. In the *Acrobot* environment, a nonlinear environment, the PPO-RMDP outperforms the PPO-DR significantly with a score of -353.01, which is very close to the best performance of -352.47 reached with the standard PPO. In this environment, the state of the art domain randomization achieves a lower score. The PPO-DR improves the robustness in the *Hopper* environment significantly compared to the PPO-RMDP and the standard PPO.

All in all these results show, that a robustness improvement against different system parameters of the algorithms is possible. Due to the fact, that system parameter changes are already introduced into the training process in both robust approaches, a robustness improvement was expected in this experiment. This research shows that in various environments, the standard PPO is outperformed by the other algorithms. The results also indicate that robustness improvement depends on the environment. In general, these results lead to the assumption, that the robust approaches gain a better robustness behaviour in terms of changing parameters. However, there is no significant difference in the performance comparing PPO-DR and PPO-RMDP.

6.3 Noise

The second part of the evaluation deals with the robustness of the algorithms against action- and observation noise. The results of this experiment are shown in Table 6.2. The results are visualized in the same way as in Section 6.2, therefore the shown results are averaged over all tested noise levels. Noise is modelled as white noise in this experiment. The best performing distributions of every environment and algorithm are considered in this section. Besides the learning distributions further, hyper-parameter does not change between the approaches.

In contrast to the previous results, the PPO-DR outperforms the PPO-RMDP approach in every environment. In all experiments expect of both *Reacher* environments, an improvement of the robustness against observation and action noise can be observed. In the *CartPole* environment the PPO-DR approach has a score of 170.21 and outperforms the standard PPO and the PPO-RMDP significantly. Also in the *Hopper* environment with an average score of 166.46 compared to the PPO-RMDP and the standard PPO as well as in the *Acrobot* environment with an average score of

Table 6.2: This table shows the best average scores with noise uncertainties of all tested algorithms in the experiment environments. After the average score the standard deviation is shown in braces.

Algorithm	CartPole	Hopper	Acrobot	Reacher Torque	Reacher Position
PPO	166.56	69.4	-95.22	**-8.49**	**-9.05**
STD	(5.54)	(12.29)	(20.27)	(1.89)	(2.34)
RMDP-PPO	160.92	105.14	-90.94	-9.85	-9.94
STD	(13.42)	(15.25)	(13.97)	(5.45)	(6.83)
PPO-DR	**170.21**	**166.46**	**-83.72**	-9.15	-9.53
STD	(5.16)	(13.22)	(1.07)	(4.92)	(2.63)

-83.72 compared the PPO-RMDP and standard PPO the state of the art algorithm outperforms the other approaches significantly.

Since the PPO-DR already introduces a form of noise within a permanently changing environment during the training process, the results of adding noise in the agents evaluation are significantly better. The PPO-RMDP approach in contrast just considers different system dynamics. The system always works in its standard parameters, without having noise in the environment itself. Out of these reasons the result could have been expected.

The standard algorithm outperforms the robust approaches in both of the *Reacher* environments, with average scores of -8.49 in the torque based and -9.05 in the position based experiment. Therefore also in terms of noise, the robustness improvement depends on the tested environment.

All in all the PPO-DR outperforms both other algorithms in most of the environments. This leads to the assumption, that domain randomization is still the state of the art approach by handling noise in the observation and action space of an agent.

6.4 Transfer

Transfer Learning is one of the most challenging applications of RL and therefore also indicates information about the robustness of an algorithm. In the transfer learning evaluation of this work, only the *CartPole* and the *Reacher* environment is realized. The results concerning average scores are shown in Table 6.3 and the corresponding transfer ratios according to the nominal behaviour of the agent in the environment are shown in Table 6.4. These figures are built up in the same way as described in the Section 6.2.

In the last part of the robustness evaluation, the results show that the PPO-RMDP never leads to a better average performance including the standard deviation than the standard PPO achieves. Therefore, in terms of transfer learning RMDPs do not provide any advantages. In case of the *CartPole* environment, the PPO-DR significantly

Table 6.3: This table shows the best average scores with transfer learning of all tested algorithms in the experiment environments. After the average score the standard deviation is shown in braces.

Algorithm	CartPole System	CartPole Noise	Reacher Torque	Reacher Position
PPO	50.45	66.71	**-8.51**	**-9.11**
STD	(7.36)	(12.45)	(1.83)	(1.57)
RMDP-PPO	46.04	57.58	-9.63	-10.23
STD	(6.19)	(8.16)	(3.94)	(2.21)
PPO-DR	**91.13**	**72.37**	-9.56	-10.81
STD	(13.22)	(15.46)	(1.03)	(0.99)

Table 6.4: Transfer Ratios

Algorithm	CartPole System	CartPole Noise	Reacher Torque	Reacher Position
PPO	0.49	0.4	**1.0**	**1.01**
RMDP-PPO	0.45	0.35	0.88	0.88
PPO-DR	**0.89**	**0.43**	0.89	0.84

leads to better results. Since the transfer ratio is a common indicator to compare performances in terms of transfer learning this value is taken into account for the comparison of the algorithm performances. The transfer ratio of 0.89 in system robustness of the PPO-DR, compared to the PPO and PPO-RMDP performances as well as the ratio of 0.43 in noise robustness of the PPO-DR, compared to both other values show, that the domain randomization achieves the best performance in this environment. In terms of the *Reacher* environment, no transfer ratio improvements is achieved compared to the standard PPO, which achieves very high transfer ratios. It is striking, that with transfer ratios of 1 or above in the *Reacher* environment the performance does not get worse. This can be attributed to the implementation of the environments in this case.

Even if there is no improvement in any environment in this experiments the PPO-DR holds as the state of the art algorithm for transfer learning problems. This assumption is also confirmed by several transfer learning researches like Peng et al. [2018], where the best results are achieved by introducing domain randomization into the learning algorithm.

6.5 Position Based Control

Coming to the last experiment the position based control for RL environments the results are included into the previous mentioned result Figures 6.1, 6.2, 6.3 and 6.4. The results in this case do not promise any robustness improvements in the first try of a position based control combined with deep RL algorithms. The position-based results

do not provide significantly better performance compared to the torque based results. These results are consistent through all algorithms, which means, that there is no robustness improvement by handling environments based on position-based control with robust approaches of the PPO. This underlines the assumption that robustness depends very much on the implementation of the environment and also on the environment properties itself.

In position-based control problems usually, the controller has to be optimized on exactly the environment and problem to get a good performance. In this case, no parameter optimization of the control is performed, which also could lead to a performance loss. In literature, there are several ways like described by Günther [2018] to perform an optimization of the controller, which could be part of a future work in this field.

7 Conclusion

To answer the questions about the improvement of the robustness concerning the overall performance, the results show that the robustness of an RL algorithm can be improved by using different distributions in the learning process without disturbing the best performance of the agent. This improvement depends strongly on the environment. Concerning the question of how the type of distribution influences the performance, the results show that the performance does not correlate with the type of distribution but the spread of distribution is the important factor. There is also not one spread of distribution to achieve the best performances. For every environment, the distribution differs and can be optimized on its own. All in all, the approach of deep RL with RMDPs does not provide a significant performance improvement compared to the state of the art algorithm and often achieves similar results to the standard PPO. It outperforms the standard as well as the state of the art approach only in the system variation of the very simple environment *CartPole*. This environment is simple to solve for RL algorithms, however, the performance in more complex environments stands behind the PPO-DR approach. The overall performance of the PPO-RMDP does not provide a robustness improvement in general, although the performance of the new algorithm increases the robustness in comparison to the PPO in various evaluations. In nonlinear and therefore more complex environments the state of the art algorithm PPO-DR provides, concerning system variation, noise and also transfer learning, better results than the proposed approach. The PPO-DR still holds to be state of the art and provides the best performing results concerning robustness in this case. Especially transfer and noise stability is significantly better than in the PPO-RMDP approach as well as in the standard PPO.

The robustness evaluation created in this work can be used as a new benchmark to evaluate robustness. Within this approach different facilities of robustness, such as noisy data, system parameter changes and transfer learning, are taken into account and the algorithms can be compared based on the performance in the single environments. The evaluation method also provides the possibility to combine different aspects of robustness evaluation to get more meaningful results. As the next step in this case a metric including all the separate results could be developed and taken as a key figure for robustness in RL.

Another finding is that position-based control of an RL problem does not necessarily lead to better performances regarding noise robustness, which was expected before. A problem, in this case, is the parametrization of the controller which has a big impact on the overall performance. By performing position control based on a PID-control,

the results are even worse than using a torque based control of the system. This can be attributed to the environment, but also to the not optimized controller performance in this case.

As an outlook and upcoming works, it is possible to perform a controller parametrization based on the work of Günther [2018] and therefore perform another robustness evaluation on this approach. Moreover, since the robustness performance depends on the environment the torque and position based research should be performed in more different environments. As already mentioned a very interesting and recent point to go on with the research in the field of robustness in RL is, to derive a single metric for the evaluation of the robustness out of the possible robustness indicators shown in this work. Another very interesting point, in this case, is a robustness evaluation of not just deterministic but also stochastic environments. Especially the realization of the RMDP approach as well as its performance in stochastic environments is a new possibility to continue this research.

Bibliography

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

J. Andrew (Drew) Bagnell, Andrew Y. Ng, and Jeff Schneider. Solving uncertain markov decision problems. Technical Report CMU-RI-TR-01-25, Carnegie Mellon University, Pittsburgh, PA, 8 2001.

Jack W Baker, Matthias Schubert, and Michael H Faber. On the assessment of robustness. *Structural Safety*, 30(3):253–267, 2008.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, 1(5):834–846, 1983.

Bahram Behzadian, Reazul Hasan Russel, and Marek Petrik. High-Confidence Policy Optimization: Reshaping Ambiguity Sets in Robust MDPs. *arXiv e-prints*, October 2019.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 449–458. JMLR.org, 2017.

Marc G. Bellemare, Nicolas Le Roux, Pablo Samuel Castro, and Subhodeep Moitra. Distributional reinforcement learning with linear function approximation. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 2203–2211. PMLR, 2019.

Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Lukas Bjarre. Robust reinforcement learning for quadcopter control, 2019.

Dotan Di Castro, Aviv Tamar, and Shie Mannor. Policy gradients with variance related risk criteria. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

Bruno Da Silva, George Konidaris, and Andrew Barto. Learning parameterized skills. *International Conference on Machine Learning*, 2, 06 2012.

Felipe Leno Da Silva and Anna Helena Reali Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703, 2019.

Erick Delage and Shie Mannor. Percentile optimization for markov decision processes with parameter uncertainty. *Operations research*, 58(1):203–213, 2010.

Esther Derman, Daniel J. Mankowitz, Timothy A. Mann, and Shie Mannor. Soft-robust actor-critic policy-gradient. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 208–218. AUAI Press, 2018.

Gabriel Dulac-Arnold, Daniel J. Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *CoRR*, abs/1904.12901, 2019.

Larry G Epstein and Martin Schneider. Recursive multiple-priors. *Journal of Economic Theory*, 113(1):1–31, 2003.

Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. In Hugh F. Durrant-Whyte, Nicholas Roy, and Pieter Abbeel, editors, *Robotics: Science and Systems VII, University of Southern California, Los Angeles, CA, USA, June 27-30, 2011*, 2011.

Eugene A Feinberg and Adam Shwartz. *Handbook of Markov decision processes: methods and applications*, volume 40. Springer Science & Business Media, 2012.

Jean-Claude Fernandez, Laurent Mounier, and Cyril Pachon. A model-based approach for robustness testing. In *IFIP International Conference on Testing of Communicating Systems*, pages 333–348. Springer, 2005.

Alborz Geramifard, Christoph Dann, Robert H. Klein, William Dabney, and Jonathan P. How. Rlpy: a value-function-based reinforcement learning framework for education and research. *J. Mach. Learn. Res.*, 16:1573–1578, 2015.

Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. Sim-to-real transfer with neural-augmented robot simulation. In *Conference on Robot Learning*, pages 817–828, 2018.

Vineet Goyal and Julien Grand-Clément. Robust markov decision process: Beyond rectangularity. *arXiv: Optimization and Control*, 2018.

Sven Gronauer. Multi-agent deep reinforcement learning, 2019.

Aditya Gudimella, Ross Story, Matineh Shaker, Ruofan Kong, Matthew Brown, Victor Shnayder, and Marcos Campos. Deep reinforcement learning for dexterous manipulation with concept networks. *CoRR*, abs/1709.06977, 2017.

Johannes Günther. *Machine intelligence for adaptable closed loop and open loop production engineering systems*. PhD thesis, Technische Universität München, 2018.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Takuya Hiraoka, Takahisa Imagawa, Tatsuya Mori, Takashi Onishi, and Yoshimasa Tsuruoka. Learning robust options by conditional value at risk optimization. In *Advances in Neural Information Processing Systems*, pages 2619–2629, 2019.

Alex Irpan. Deep reinforcement learning doesn't work yet. *Internet: https://www. alexirpan. com/2018/02/14/rl-hard. html*, 2018.

Garud N Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280, 2005.

Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer, 1995.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Jens Kober, Andreas Wilhelm, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33 (4):361–379, 2012.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

George Konidaris and Andrew G Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in neural information processing systems*, pages 1015–1023, 2009.

Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Erwan Lecarpentier and Emmanuel Rachelson. Non-stationary markov decision processes, a worst-case approach using model-based reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 7216–7225, 2019.

Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229, 2004.

Yuxi Li. Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274, 2017.

Shiau Hong Lim and Arnaud Autef. Kernel-based reinforcement learning in robust markov decision processes. In *International Conference on Machine Learning*, pages 3973–3981, 2019.

Shiau Hong Lim, Huan Xu, and Shie Mannor. Reinforcement learning in robust markov decision processes. In *Advances in Neural Information Processing Systems*, pages 701–709, 2013.

Yuan-Pin Lin and Tzyy-Ping Jung. Improving eeg-based emotion classification using conditional transfer learning. *Frontiers in human neuroscience*, 11:334, 2017.

Clare Lyle, Marc G Bellemare, and Pablo Samuel Castro. A comparative analysis of expected and distributional reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4504–4511, 2019.

Daniel J Mankowitz, Timothy A Mann, and Shie Mannor. Adaptive skills adaptive partitions (asap). In *Advances in Neural Information Processing Systems*, pages 1588–1596, 2016.

Daniel J. Mankowitz, Timothy A. Mann, Pierre-Luc Bacon, Doina Precup, and Shie Mannor. Learning robust options. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6409–6416. AAAI Press, 2018.

Timothy Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *International conference on machine learning*, pages 127–135, 2014.

Shie Mannor, Ofir Mebel, and Huan Xu. Robust mdps with k-rectangular uncertainty. *Mathematics of Operations Research*, 41(4):1484–1509, 2016.

Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 361–368. Morgan Kaufmann, 2001.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

Jun Morimoto and Kenji Doya. Robust reinforcement learning. *Neural computation*, 17 (2):335–359, 2005.

Arnab Nilim and Laurent El Ghaoui. *Robust markov decision processes with uncertain transition matrices*. PhD thesis, University of California, Berkeley, 2004.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1–8. IEEE, 2018.

Marek Petrik and Reazul Hasan Russel. Beyond confidence regions: Tight bayesian ambiguity sets for robust mdps. In *Advances in Neural Information Processing Systems*, pages 7049–7058, 2019.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2817–2826. PMLR, 2017.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Fereshteh Sadeghi and Sergey Levine. CAD2RL: real single-image flight without a single real image. In Nancy M. Amato, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma, editors, *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017.

Jay K Satia and Roy E Lave Jr. Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21(3):728–740, 1973.

Mariano Schain and Mariano Schain. *Machine Learning Algorithms and Robustness*. Universitat Tel-Aviv, 2015.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Clayton Scott. A rate of convergence for mixture proportion estimation, with application to learning from noisy labels. In *Artificial Intelligence and Statistics*, pages 838–846, 2015.

Oliver G Selfridge, Richard S Sutton, and Andrew G Barto. Training and tracking in robotics. In *IJCAI*, pages 670–672, 1985.

Arjun Sharma and Kris M Kitani. Phase-parametric policies for reinforcement learning in cyclic environments. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Elena Smirnova, Elvis Dohmatob, and Jérémie Mary. Distributionally robust reinforcement learning. *CoRR*, abs/1902.08708, 2019.

Freek Stulp, Gennaro Raiola, Antoine Hoarau, Serena Ivaldi, and Olivier Sigaud. Learning compact parameterized skills with a single regression. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 417–422. IEEE, 2013.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

Balint Takacs, András Lörincz, and Sridhar Mahadevan. epsilon-mdps: Learning in varying environments. *JMLR Volume 3*, 09 2002.

Aviv Tamar, Huan Xu, and Shie Mannor. Scaling up robust mdps by reinforcement learning. *CoRR*, abs/1306.6189, 2013.

Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the cvar via sampling. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In Hadas Kress-Gazit, Siddhartha S. Srinivasa, Tom Howard, and Nikolay Atanasov, editors, *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Chen Tessler, Yonathan Efroni, and Shie Mannor. Action robust reinforcement learning and applications in continuous control. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6215–6224. PMLR, 2019.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

Harish K Venkataraman and Peter J Seiler. Recovering robustness in model-free reinforcement learning. In *2019 American Control Conference (ACC)*, pages 4210–4216. IEEE, 2019.

Eugene Vinitsky, Yuqing Du, Kanaad Parvate, Kathy Jang, Pieter Abbeel, and Alexandre M. Bayen. Robust reinforcement learning using adversarial populations. *CoRR*, abs/2008.01825, 2020.

Jingkang Wang, Yang Liu, and Bo Li. Reinforcement learning with perturbed rewards. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 6202–6209. AAAI Press, 2020.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *CoRR*, abs/1907.02057, 2019.

Paweł Wawrzyński. Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks*, 22(10):1484–1497, 2009.

Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust markov decision processes. *Mathematics of Operations Research*, 38(1):153–183, 2013.

Huan Xu and Shie Mannor. The robustness-performance tradeoff in markov decision processes. In *Advances in Neural Information Processing Systems*, pages 1537–1544, 2007.

Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In Nancy M. Amato, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma, editors, *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017.

Jun Zhao, Guang Qiu, Ziyu Guan, Wei Zhao, and Xiaofei He. Deep reinforcement learning for sponsored search real-time bidding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1021–1030, 2018.

Jiangchuan Zheng, Siyuan Liu, and Lionel M Ni. Robust bayesian inverse reinforcement learning with sparse behavior noise. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Bo Zhou, Hongsheng Zeng, Fan Wang, Yunxiang Li, and Hao Tian. Efficient and robust reinforcement learning with uncertainty-based value expansion. *CoRR*, abs/1912.05328, 2019.

Appendix

Hyperparameter

In the following tabular all hyperparameters used to perform the RL experiments are given.

Table 7.1: Parameter setup for the hyperparameter search

Parameter	Symbol	Values
MLP architecture		$(64, 64)$
activation function	$h(\cdot)$	$\in$ {tanh}
learning rate	α	$\in \{0.0003, 0.001\}$
trained epochs		60
update cycle	K	$\in \{1, 4, 8\}$
clipping factor	ϵ	0.2
steps per epoche		9600
mini batch size		1200
discount factor	γ	0.95
GAE	κ	0.95
target kl		0.01

Detailed Results

In the following section, all detailed results are shown. The average scores for the PPO-RMDP and the PPO-DR with different learning distributions for every environment and experiment are shown in the following tables.

Table 7.2: System robustness average scores

CartPole				
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
104.21	103.42	105.21	**105.26**	102.22
Hopper				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	Gauss $\sigma = 0.05$
39.30	41.64	**51.43**	46.69	43.11
Gauss $\sigma = 0.1$	Lognorm $\sigma = 1$			
42.23	48.32			
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
52.21	55.13	53.71	**57.8**	36.27
Acrobot				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	Gauss $\sigma = 0.05$
-361.19	-355.15	**-353.01**	-360.12	-359.31
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	Gauss $\sigma = 0.05$
-363.23	-361.35	-365.56	-360.03	**-359.68**
Reacher Torque Control				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-8.63	-8.85	-8.92	-8.97	
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-9.21	-8.83	**-8.61**	-9.11	
Reacher Position Control				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-8.95	-9.01	-9.05	-9.12	
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-9.3	-9.12	**-8.93**	-9.21	

Table 7.3: Noise robustness average scores

CartPole				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
160.92	156.1	154.23	155.31	151.24
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
165.15	167.34	166.45	169.42	**170.21**
Hopper				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
70.85	80.79	**105.14**	85.73	91.42
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
150.31	156.15	153.82	**166.46**	147.6
Acrobot				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
-90.94	-91.46	93.12	-92.56	-95.47
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
-87.42	-85.45	-86.93	**-83.72**	-84.34
Reacher Torque Control				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-10.26	-9.86	**-9.85**	-9.92	
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-9.79	-9.75	**-9.15**	-9.58	
Reacher Position Control				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-9.94	-10.08	-10.12	-10.49	
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-12.89	-11.07	**-9.53**	-10.46	

Table 7.4: Transfer learning average scores

CartPole system robustness				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
44.58	**46.04**	41.78	40.69	45.23
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
80.31	81.6	83.64	**91.13**	86.59
CartPole noise robustness				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
56.87	**57.58**	55.43	52.34	51.89
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Gauss $\sigma = 0.05$	Gauss $\sigma = 0.1$
63.21	65.36	69.87	**72.37**	68.6
Reacher Torque Control				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-9.63	-10.04	-10.05	-11.35	
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-9.85	**-9.81**	-9.56	-9.79	
Reacher Position Control				
PPO-RMDP				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-10.23	-10.40	-10.42	-10.78	
PPO-DR				
Uniform 5%	Uniform 10%	Uniform 20%	Uniform 30%	
-13.61	-12.64	**-10.81**	-11.85	