A Hybrid Approach for Constrained Deep Reinforcement Learning

Scientific thesis for the procurance of the degree B.Sc.
from the Department of Electrical and Computer Engineering at the Technical University of Munich.

Supervised by
Univ.-Prof. Dr.-Ing./Univ. Tokio habil. Martin Buss
M.Sc. Ozgur Oguz
Chair of Automatic Control Engineering

Submitted by
cand. ing. Ahmed Magdy Hendawy
Karlsbergstrasse, 25
81475 Munich
015208379970

Submitted on
Munich, 27.07.2018

Acknowledgments

Thanks God

For Mom, Dad and my brothers who always believe in me. For my aunt who is supporting me. For my friends Nasr and Hossam who are pushing me forward. Finally, For my supervisor who helped me to reach that level and without him I may not have completed my thesis in this professional form.

Abstract

Recently, deep reinforcement learning techniques have achieved tangible results for learning high dimensional control tasks. Due to the trial and error interaction between the autonomous agent and the environment, the learning phase is unconstrained and limited to the simulator. Such exploration has an additional drawback of consuming unnecessary samples at the beginning of the learning process. Model-based algorithms, on the other hand, handle this issue by learning the dynamics of the environment. However, model-free algorithms have a higher asymptotic performance than the model-based one. Our contribution is to construct a hybrid structured algorithm, that makes use of the benefits of both methods, to satisfy constraint conditions throughout the learning process. We demonstrate the validity of our approach by learning a reachability task. The results show complete satisfaction for the constraint condition, represented by a static obstacle, with less number of samples and higher performance compared to state-of-the-art model-free algorithms.

Contents

Chapter 1

Introduction

Over the last few years, deep reinforcement learning approaches demonstrated remarkable results for learning high dimensional control tasks, starting from learning how to play Atari games from raw-pixel inputs [MO15, MK16], competing with the human for playing games like Go [SP16, SB17], participating in the autonomous driving [SY17], learning locomotion tasks and manipulation skills from raw sensory inputs [LA6a, SLM+15, SA16, LW15]. Most of these results have been achieved by using the trial and error interaction between the agent and environment assuming there are no constraints for the learning process. Safety is one important constraint for learning that should be taken into consideration if we want to transfer the learning from a simulated environment into the real world. Assume that we have manipulator which tries to learn a task in the workspace. The actions of the manipulator should be constrained for safety requirements for itself and the workers and the machines around it. Then we need to satisfy the safety condition within each step of learning.

Introducing constraints to the reinforcement learning framework is a well-known area of research which is formulated as constrained Markov Decisions Process (CMDP) problem [Alt99]. CMDP formulation is working well with finite policies for known dynamical model which can be solved then by linear programming [LK17]. However, this is limited to low-dimensional environments and can not scale to high-dimensional control tasks we target.

From control theory, constrained model-predictive control (CMPC) [MS00] is one way for performing a task with regards to some constraints. Optimization is done for sequence of actions that satisfy constraint functions with applying the first action from this sequence. Dynamical model is used to estimate the next states within the optimization process. We are inspired by CMPC for achieving the target of constraining our learning process, but we need a dynamical model for predicting the next states.

Formulating the dynamical model or equations of motions of a complex system is not trivial for non-linear high dimensional environments. Model-based algorithms tackle this issue by learning the dynamical model of the environment and optimize for the actions by using optimal control methods. Such methods show faster learning process than any other model-free algorithms which have problems with sample complexity. However, the asymptotic performance of the model-based approaches can not be compared with the model-free one. This raises an auxiliary problem for balancing this well-known trade-off.

In this work, we propose a hybrid reinforcement learning algorithm that satisfies constraint conditions within the learning process. The hybrid structure of our approach helps to balance the trade-off between model-based and model-free methods. Strengths of model-based and model-free algorithms are utilized, starting from learning the dynamical model in a simulator in unconstrained form by using a model-based method, then using it in a constrained environment to improve the policy performance by maximizing the expected reward using a model-free algorithm. We show the validity of our approach by applying it on learning a manipulation task with collision avoidance constraint for safety.

1.1 Related Work

Constrained reinforcement learning has long been a topic of interest specially with the consideration of the safety as constraint. A comprehensive overview of safety in reinforcement learning was given by [GF15].

Safe policy search methods have been proposed in prior work. [UD07] introduce a policy gradient algorithm that enforces active constraints by using gradient projection, but this algorithm suffers from not making the policy always safe within the learning process. [AE15] propose a theoretically-motivated policy gradient method for lifelong learning with safety constraints, but this method includes an expensive inner loop optimization, in terms of computation complexity, of a semi-definite program which inapplicable for deep reinforcement learning framework.

Policy search within CMDP framework is well-known approach for satisfying constraints for reinforcement learning. [AHTA17] is considered as the state of the art constrained reinforcement learning approach that both guarantees constraint satisfaction throughout training and works for arbitrary policy classes. However, in the implementation on continuous control tasks, this method suffers from constraint violation within the learning process.

Introducing corrective actions, to avoid constraints violation, is a preferred method in the optimization and control research fields. From control theory, [KH17] achieved a safe human-robot interaction by using invariance control beside the nominal one

to introduce corrective actions to keep the constraints satisfaction. However, using invariance control in the learning scheme introduces a challenge of linearizing the input-output equation which means using a linear dynamical model rather than Gaussian process or neural network. On the other hand, [PT17] propose a practical method that uses optimization layer to provide corrective actions in the learning process. This work is based on [AK17] that develop an optimization layer with neural network. The method achieves zero constraints violation for manipulation tasks, but it is not clear if the method can perform the same with high dimensional control tasks as locomotion.

Combining control theory with learning is considered as powerful tool that is used to satisfy safety guarantees. [KK18] propose a learning-based approach for safe exploration within reinforcement learning framework by using model-predictive control as a control scheme. This work provides provable high-probability safety guarantees by constructing a confidence interval for the predicted trajectories, but the method is applied on a simple inverted pendulum without testing it on high dimensional control tasks. [FT17] propose a general safety framework based on Hamilton-Jacobi reachability methods that can work in conjunction with an arbitrary learning algorithm.

Model-free algorithms based on Q-learning, actor-critic methods and policy gradients have been showed the ability to learn complex skills in high-dimensional state spaces. However, model-free algorithm suffers from the high sample complexity problem even [SLM$^+$15], that achieve a monotonic improvement for the learned policy, can not be compared to the sample efficiency of the model-based algorithms. On the other hand, model-based algorithms achieve a low asymptotic performance comparing to the model-free algorithms. This dilemma pushes the idea of combining the benefits of both algorithms in one method [OS15, HT00]. We make use of [NKFL17] hybrid structure for balancing the trade-off within our constrained approach.

We propose a deep reinforcement learning algorithm for constrained learning on a hybrid form that balances the trade-off of achieving sample efficiency and high asymptotic performance. We test our algorithm on a manipulation task, for which zero constraints violation is achieved with fast learning process in terms of number of samples and high asymptotic performance.

In the next chapter, we focus on the methodology for our algorithm. Then we show the results for our algorithm on 2DOF planar robotic arm for learning reaching task. After that, we have a discussion for the results obtained then a conclusion for the whole work.

Chapter 2

Methodology

Our approach is a deep reinforcement learning algorithm that can be used to learn a specific task with constraints introduced within the system. Our algorithm is used to balance the trade-off of the reinforcement learning between the sample efficiency of the model-based algorithms and high asymptotic performance of the model-free algorithms. We are inspired by [NKFL17] to balance this trade-off. For better understanding of our approach, we need first to introduce some basic knowledge in Sec. 2.1. Then we will explain the details of our algorithm in Sec. 2.2, with our contribution to have a learning algorithm that can highly perform within a constrained system.

2.1 Technical Background

2.1.1 Deep Reinforcement Learning

Reinforcement learning is one of three branches of machine learning beside supervised and unsupervised learning. It focuses on goal-directed learning from interaction with the environment. The idea of interacting with our environment for learning is the first concept we think about for the nature of learning. Learning decision making is the main problem reinforcement learning tries to solve. And this is by learning how to map situations into actions to maximize a numerical reward signal. The agent interact with the environment by applying actions that change the environment state. Then the environment sends the next state with the appropriate reward signal back to the agent as shown in Fig. 2.1. Discovering the right sequence of actions with the most reward by the agent is based on trial and error. The most delayed rewards are affected by the actions taken at the current time. We formalize the problem of reinforcement learning using ideas from dynamical systems theory, specifically, as the optimal control of incompletely-known Markov decision

processes. Markov decision process is intended to include the three main aspects of reinforcement learning - observation, actions and reward. Any method solves this problem, considered as reinforcement learning method.

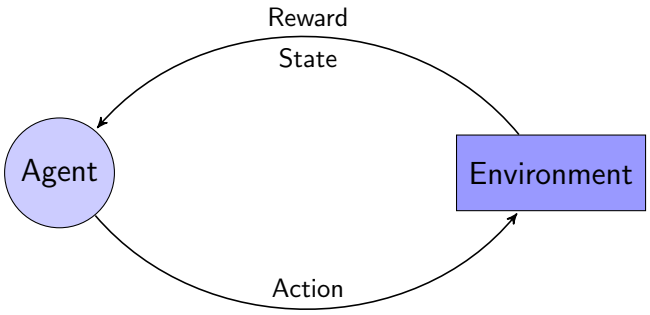

Figure 2.1: Flow chart for reinforcement learning interaction

Reinforcement learning is different from supervised learning. In supervised learning, a model is trained on a training set of labeled examples provided by knowledgeable external supervisor. The objective of this type of learning is to have a generalized model that can act correctly a situation that is unpresented in the training set. Imitation learning is a supervised learning method tries to solve the decision making problem which the reinforcement learning tries to solve too. The supervisor or the expert in some cases is the human. Human labels the appropriate actions for each situation to construct the training set. Imitation learning is part of our algorithm that is covered in Sec. 2.2.2 with non-human supervisor.

Reinforcement learning is also different from what is called unsupervised learning. Unsupervised learning tries to learn hidden structure of unlabeled data. One might think that reinforcement learning is trying to find the hidden structure in the agent's experience which will help in learning. However, the main objective of reinforcement learning is to maximize the reward signal. We can consider reinforcement learning as semi-supervised learning.

One of the challenges that rises in reinforcement learning, is the trade-off between exploration and exploitation. The agent takes actions that maximize the reward where these actions had been taken before by the agent. But to discover these actions from the beginning the agent needs to take new actions never taken before. The agent should balance between exploiting actions, that are taken before to achieve maximum reward, and exploring the action space for new actions that may lead to better performance and higher reward. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task the problem even worse as each action must be tried many times to gain a reliable estimate of its expected reward.

The agent experience can be broken up into a series of episodes with a finite number

of states, actions and reward signals. We call that an episodic reinforcement learning problem. The episode starts by an initial state s_0 that is sampled from initial state distribution ρ_0. For each time steps, the agent acts by choosing action a_t from a distribution $\pi(a_t \mid s_t)$. π is called policy, it is the probability distribution from which the agent uses to sample actions (in case of stochastic policy, but if the policy is deterministic, the actions are the output directly from a function approximator like neural network). Then the environment generates the next state and the reward signal, according to some distribution $P(s_{t+1}, r_t \mid s_t, a_t)$ which is the transition probability. The episode ends by terminal state s_T where T is the maximum time steps.

Another famous trade-off in reinforcement learning, is the model-free and model-based trade-off. Model-free and model-based algorithms are the methods that try to solve the reinforcement learning problem but with different approaches. There are pros and cons for each one of these methods. Model-free algorithms treat the environment as a black box with trial and error until reaching to a high performance. Due to this trial and error learning based, the interaction time increases which means the samples or the trail of learns increase. This is known as sample complexity problem. On the other hand, model-based algorithms are able to solve this problem by learning the dynamics of the environment then act by optimal control methods to get a high performance. The model-free algorithms achieve high asymptotic performance better than that of the model-based. Then the trade-off is to choose the appropriate algorithm for the appropriate problem. We cover both methods in the next couple of sections as our approach is based on this trade-off.

2.1.2 Model-Free Algorithms

Many reinforcement learning algorithms consider the environment as unknown, which makes the interaction between the agent and the environment based on trial and error. These algorithms are called model-free which assumes no information about the future after taking some actions. Model-free algorithms prove that this an effective way to interact with the environment for learning a specific task. These algorithms depend only the reward value at the end of each episode for the policy improvement by maximizing the expected total reward,

$$\theta^* = \arg\max_{\theta} \quad E_\tau \left[\sum_t r(s_t, a_t) \right] \tag{2.1}$$

where θ defines the parameters of the policy we need to optimize and τ is a trajectory which is sequence of states s and actions a. One of the strongest model-free algorithms is trust region policy optimization TRPO [SLM+15] which achieves monotonic improvement for complex continuous benchmark tasks.

According to the way that model-free algorithms interact with the environment, there is a drawback for using these algorithms which is the sample complexity problem. Sample complexity means that the algorithm need a lot of samples to learn a task which may cause some problems if the simulator is expensive or we train an agent on a real system which causes power consumption. On the other hand, the asymptotic performance, which is the performance at the end of the learning phase, is high.

2.1.3 Model-Based and Optimal Control

There is another way to learn a task which is learning the dynamics of the system. This means that we know how the environment will behave and what states we will end up in after applying some actions. Then we can act optimally by using optimal control method for planning and choose the appropriate action. These type of algorithms we call it model-based algorithms. Model-based algorithm is considered as a very powerful tool for learning the task with prior knowledge. We need to approximate our dynamical system, which may be linear or non-linear, into a trainable model. In model-based algorithms, dynamical model can be approximated either by linear models [LA6a], Gaussian processes (GPs) [DR11] or neural networks [NKFL17, KCD+18].

Model-based algorithms have sample efficiency that make it attractable to be used to tackle the problem of huge number of samples consumed by model-free algorithms. On the other hand, these algorithms could not achieve very high asymptotic performance comparing to the model-free algorithms. These is considered one of the difficult reinforcement learning trade-off that needed to be balanced to achieve the desired aim.

2.2 Hybrid Constrained MPC Reinforcement Learning Algorithm

Our algorithm consists of three main stages which form a sequential learning pipeline as shown in Fig. 2.2. The first stage is the model-based algorithm at which we achieve the sample efficiency by learning the dynamics of the system and acting optimally by using an MPC-like controller. We explain this stage in detail in Sec 2.2.1. In the second stage, we mimic the behavior of the MPC controller with a policy represented by a neural network. This is called imitation process which is explained in Sec. 2.2.2. The last stage is our main contribution at which we use constrained MPC controller to interact with constrained environment where the actions are sampled from the

model-free policy that has been initialized with the parameters of the imitation policy in the second stage. Our contribution is discussed in detail in Sec. 2.2.3.

We consider an infinite-horizon discounted Markov Decision Process (MDP) characterized by a tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, with $\mathcal{S}$ the set of states, $\mathcal{A}$ the set of actions. $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the transition probability distribution to go from one state to another by taking a specific action, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the function of associated rewards, $\rho_0 : \mathcal{S} \to [0, 1]$ the initial state probability distribution, $\gamma \in [0, 1)$ a discount factor, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ a stochastic policy.

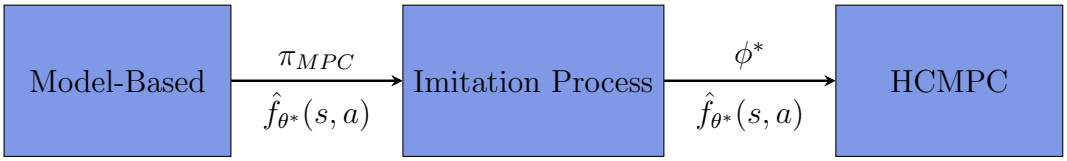

Figure 2.2: Flow chart for hybrid constrained MPC pipeline.

2.2.1 Model-Based Deep Reinforcement Learning

The model-based deep reinforcement learning method is the first stage in our algorithm. The learning process within this stage will be in unconstrained form (using a physics engine simulator). The method gives us an advantage of having a general learned dynamical model that we can use later for any other task. On the other hand, a policy, which performs a specific task, can be extracted from this method. We explain the dynamical model approximation in detail in Sec. 2.2.1.1, how to construct initial dataset and preprocessing the data with the help of normalization vector in Sec. 2.2.1.2, using the dataset to train the dynamical model in Sec. 2.2.1.3, extracting a policy that performs specific task in Sec. 2.2.1.4, using reinforcement learning to improve our dynamical model performance in Sec. 2.2.1.5.

2.2.1.1 Dynamical Model Approximation

We approximate our dynamical model by a deep neural network $f_\theta(s_t, a_t)$ which is parameterized by θ which represents the weights and biases for hidden and output layers for the neural network. The straightforward way is to make $f_\theta(s_t, a_t)$ accept as an input the current state s_t and action a_t and output the predicted next state s_{t+1}. But the neural network is difficult to learn the difference between the s_{t+1} and s_t when they are similar in case the action has a very small effect on the next state. The problem becomes difficult when the time between the states Δt is very small. The solution for this problem is to make $f_\theta(s_t, a_t)$ output the difference between the next state and current state $\hat{\Delta}$ over time step duration Δt . Then the estimated

next state $\hat{s}_{t+1}$ is as follows,

$$\hat{s}_{t+1} = s_t + f_\theta(s_t, a_t) \tag{2.2}$$

This approach will relieve the neural network from memorizing the input state [DR11, FA16, NKFL17, KCD+18]. Increasing Δt can help to increase the information for each time step. On the other hand, it will increase the discretization and the complexity of the underlying continuous-time dynamics, which can make the learning process more difficult.

2.2.1.2 Collecting and Preprocessing the Training Data

Learning the dynamics of a system needs only the actual transition (s_t, a_t, s_{t+1}) which means that to train the dynamical model it does not need transitions coming from optimal or even sub-optimal policy. Random policy π_{RAND} is used by the agent to collect some actual transitions after interacting with the environment by taking actions each time step. We record the resulting trajectories $\tau = (s_0, a_0, ..., s_{T-1}, a_{T-1}, s_T)$ for T time steps where the initial configurations are sampled $s_0 \sim \rho_0$.

We slice the resulting trajectories into current state s_t and action a_t as data inputs and the difference between the consecutive states Δ (where $\Delta = s_{t+1} - s_t$) as the output labels. We will use the dataset D to extract the normalization vector which consists of mean μ and standard deviation σ of each of the dataset components (s_t, a_t, Δ). Normalization vector is used to nomalize the inputs for the neural network and denormalize the output,

$$\hat{\Delta} = \mu_\Delta + \sigma_\Delta \odot f_\theta(\frac{s_t - \mu_s}{\sigma_s + \epsilon}, \frac{a_t - \mu_a}{\sigma_a + \epsilon}) \tag{2.3}$$

where $\mu_s, \sigma_s, \mu_a, \sigma_a, \mu_\Delta, \sigma_\Delta$ are normalization vector components, $\odot$ is an elementwise vector multiply and ϵ is a small positive value (to prevent divide-by-zero). Normalization for the data helps in relieving the effect of exploding gradients.

2.2.1.3 Train the dynamical model

We train the dynamical model on dataset D by minimizing the error function,

$$\varepsilon(\theta) = \frac{1}{|D|} \sum_{(s_t, a_t, \Delta_t) \in D} \frac{1}{2} \left\| \Delta_t - f_\theta(s_t, a_t) \right\|^2 \tag{2.4}$$

where $\varepsilon(\theta)$ is a mean square error function and $f_\theta(s_t, a_t)$ is the output state difference $\hat{\Delta}$ after denormalization, using stochastic gradient descent. In our case, we use Adam optimizer [KB14] to solve this supervised learning problem.

2.2.1.4 Model-Based Control

In order to learn a specific task optimally with the help of the learned dynamical model $\hat{f}_\theta(s_t, a_t)$ and cost function $c(s_t, a_t)$ which describes the task, we use model-based controller, that is both computationally tractable and robust to mismatch in the learned dynamical model, to achieve the desired performance for learning the task. We optimize the sequence of actions $A_t^{(H)} = (a_t, ..., a_{t+H-1})$ over a finite horizon H, using the learned dynamical model to predict the next states,

$$A_t^{(H)} = \arg\min_{A_t^{(H)}} \sum_{t'=t}^{t+H-1} c(\hat{s}_{t'}, a_{t'}) \tag{2.5}$$

where $\hat{s}_t = s_t, \hat{s}_{t'+1} = \hat{s}_{t'} + f_\theta(s_{t'}, a_{t'})$. This method is difficult to be used in our case due to the non-linearity in the learned dynamical model (which is a neural network) and the cost function. There are some techniques that can solve the optimization problem to get an approximate solution for finite horizon H. One of these methods is simple shooting method [Rao09] that we will use to solve our optimization problem. At the beginning, we will generate N imaginary trajectories,

$$\tau_t^j = (s_t^j, a_t^j, ..., s_{t+H-1}^j, a_{t+H-1}^j, s_{t+H}^j) \tag{2.6}$$

where for j ($0 \leq j \leq N$), initial state $s_t^j = s_t$, actions are randomly sampled from $a^j \sim \pi_{RAND}$ and next states are predicted by using the learned dynamical model $s_{t+1}^j = \hat{f}_\theta(s_t^j, a_t^j)$. Using our cost function $C(\tau)$, we will evaluate all the imaginary action sequences by calculating the cost for each, and the action sequence with the least cost value will be chosen,

$$j^* = \arg\min_j \quad C(\tau^j) \tag{2.7}$$

Because our learned dynamical model is not accurate enough, we address this problem of mismatching by using model predictive control (MPC): the MPC controller π_{MPC} executes only the first action $a_t^{j^*}$ from the resulting trajectory directly on the environment, receives the actual next state s_{t+1} from the environment and re-optimizes for the action sequence at the next time step. This interaction with the environment will construct actual trajectory τ that have been executed for T time steps. Simple random shooting method is very sufficient in case we have finite horizon, but for longer horizon and higher dimensional action space, we need other powerful method, *e.g.* ILQR [LT04].

2.2.1.5 Improving Model-Based Control with Reinforcement Learning

To improve the performance of our model-based algorithm which is especially affected by the accuracy of our learned dynamical model, we need to train the dynamical model on more data. We make use of the data collected after interacting with

the environment using the MPC controller π_{MPC}. We increase the on-policy data by alternating between gathering data, D_{RL}, by interacting with the environment and retraining the dynamical model with the aggregated data.

To summarize the full algorithm, we collect random trajectories, and added to dataset D_{RAND}, by using random policy π_{RAND} by which the dynamical model will be trained with initially. After that MPC controller π_{MPC} will be used to interact with the environment to collect some on-policy trajectories which form dataset D_{RL}.Then we combine it with dataset D_{RAND} to retrain our dynamical model with $D = D_{RAND} \cup D_{RL}$. The algorithm continues alternating between retraining the dynamical model and gathering new data until we reach to the predefined maximum number of iterations. The pseudo code in Alg. 1 and the flow chart in Fig.2.3 describe the model-based deep reinforcement learning algorithm.

Algorithm 1 Model-Based Deep Reinforcement Learning

gather dataset D_{RAND} by using π_{RAND}
initialize empty dataset D_{RL}, and randomly initialize $\hat{f}_\theta$
for $iter = 1$ to $maxiter$ **do**
    train $\hat{f}_\theta(s, a)$ by performing gradient descent on Eqn. 2.4
    using D_{RAND} and D_{RL} ▷ using D_{RAND} only at $iter = 1$
    for $ep = 1$ to N_{ep} **do** ▷ N_{ep} episodes per iteration
        for $t = 1$ to T **do** ▷ T time steps per episode
            get agent's current state s_t
            use $\hat{f}_\theta$ to generate N imaginary trajectories of H horizon (Eqn. 2.6)
            use $C(\tau)$ to get τ with least cost (Eqn. 2.7)
            execute the first action a_t from the selected trajectory τ
        end for
        add (s_t, a_t, Δ_t) to D_{RL} for T time steps
    end for
end for

At the end of this stage, we have a general learned dynamical model $\hat{f}_{\theta^*}(s_t, a_t)$ that can be used for performing many tasks. We extract an MPC controller π_{MPC} that can perform a specific task with the help of the learned dynamical model and cost function $C(\tau)$ that can be designed according to the task description. As we mentioned before, the model-based methods can achieve sample efficiency but still the asymptotic performance is low. We need to make use of the model-free algorithms for achieving the high asymptotic performance. Imitation process, that is explained in the next section in detail, links between the two algorithms by copying the behaviour of the model-based controller and using it for the initialization of model-free policy.

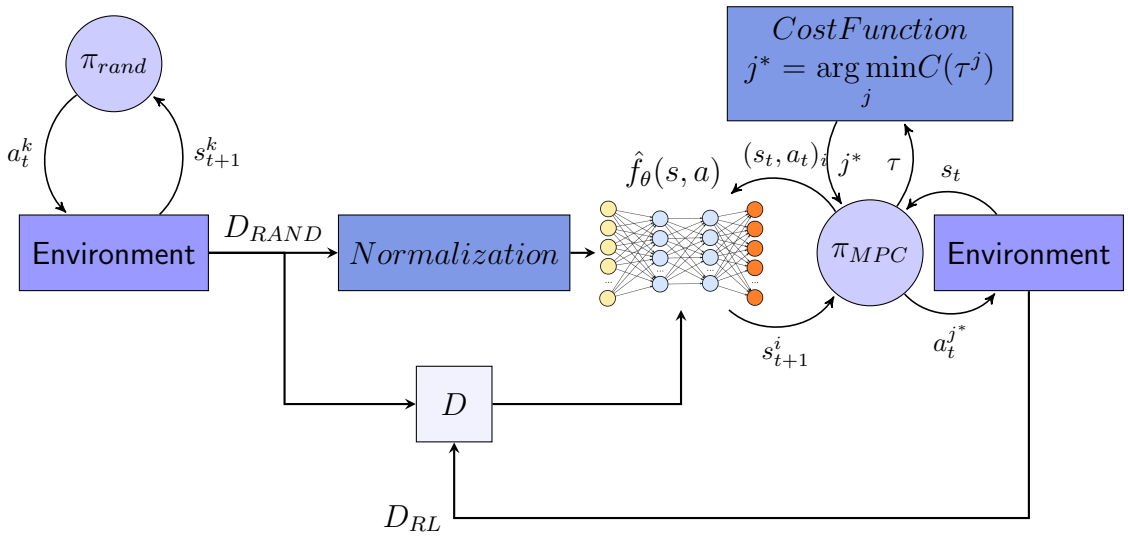

Figure 2.3: Model-based deep reinforcement learning flow chart.

2.2.2 Imitation process

In this stage, we train a policy $\pi_\phi(a|s)$, represented by a neural network and parameterized by ϕ, to mimic the behaviour of expert policy which is the model-based controller in our case. We have as inputs, from the last model-based stage, the final learned dynamical model $\hat{f}_{\theta^*}(s_t, a_t)$ and the MPC controller π_{MPC}. To train the imitation policy $\pi_\phi(a|s)$, we need dataset that is collected and labeled by the expert controller. Our agent will interact with the environment again by using the model-based controller, following the model-based algorithm in Alg. 1, to collect N^* trajectories with which we construct a dataset, by extracting the states s_t and the true actions by the expert a_t from the trajectories to form the dataset D^*. The policy parameters ϕ will be trained by optimizing behaviour cloning objective

$$\min_\phi \frac{1}{2} \sum_{(s_t,a_t)\in D^*} \|a_t - \pi_\phi(a_t \mid s_t)\|_2^2, \tag{2.8}$$

using stochastic gradient descent which is Adam optimizer in our case. Representation for the imitation process is shown in Fig. 2.4.

At the end of this stage, we have as a result the final parameters ϕ^* of the imitation policy which mimic the performance of the model-based controller. We will use the final parameters ϕ^* and the final learned dynamical model as input for our next stage. We will initialize the model-free policy with the final parameters ϕ^*.

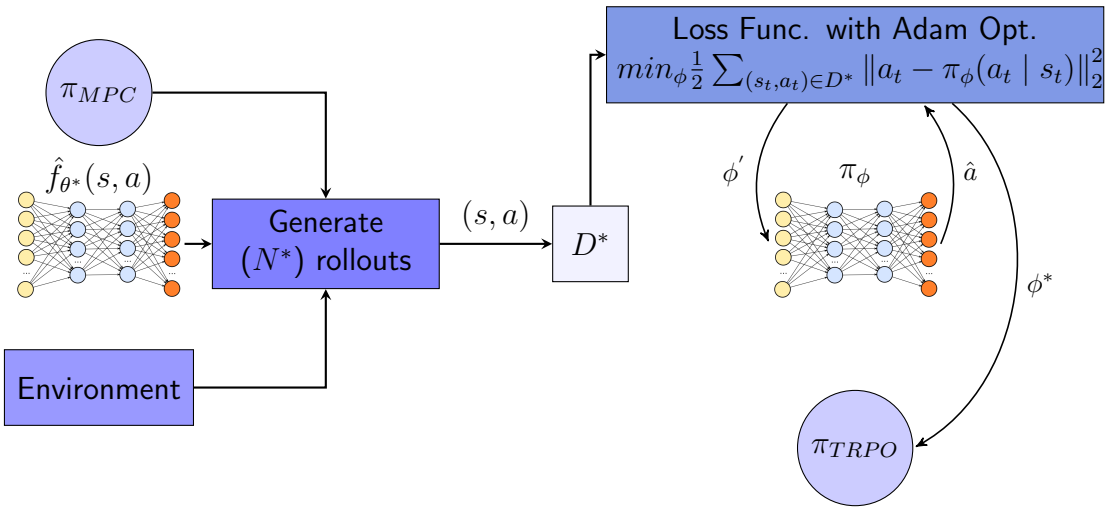

Figure 2.4: Imitation process flow chart.

2.2.3 Hybrid Constrained MPC

Our main contribution is discussed in this section. In this stage, we will have the second phase of learning the task but in a constrained form. We will make use of having a learned dynamical model that predict the next state and the parameters ϕ^* that mimic the behaviour of the model-based controller after the imitation process for safe complete learning process. We use the parameters ϕ^* as initial values for the model-free policy parameters and that is explained in Sec. 2.2.3.1, interacting with the constrained environment with safe trajectories introduced by MPC controller that samples actions from the model-free policy in Sec. 2.2.3.2, extensions introduced to solve some problems in Sec. 2.2.3.3.

2.2.3.1 Initialization of The Model-Free Learner

In this stage, we use trust region policy optimization TRPO as a back-end model-free algorithm to train our policy. TRPO is the state of the art reinforcement learning algorithm that achieves a monotonic improvement for the policy and does not need to initialize any critic or value function. To achieve this monotonic improvement, they limit the policy improvement within a trust region by constraining the update step by a predefined value δ_{TRPO}. Our algorithm can also be combined with any other model-free algorithm. The model-free policy $\pi_{TRPO,\phi}$ is parameterized as a conditionally Gaussian policy,

$$\pi_{TRPO,\phi} \sim \mathcal{N}(\mu_\phi(s), \Sigma) \tag{2.9}$$

where the mean $\mu_\phi(s)$ is a deep neural network that is parameterized by the final

Algorithm 2 Hybrid Constrained MPC Algorithm

initialize π_{TRPO} and $\pi_{primary}$ with ϕ^*
set values for the hyperparameters Σ, δ, ρ and failure cost
for $iter = 1$ to $maxiter$ **do**
    for $ep = 1$ to N_{ep} **do**
        for $t = 1$ to T **do**
            get agent's current state s_t
            generate imaginary trajectory set T for H horizon using $\hat{f}_{\theta^*}(s, a)$
            use $G(\tau)$ to filter T to get T_{safe}
            if <T_{safe} `is empty`> **then**
                for c to N_r **do** ▷ Resampling loop
                    generate new T and filter to get T_{safe}
                    if <T_{safe} `is not empty`> **then**
                        $Failure = False$
                        Break
                    else
                        $Failure = True$
                    end if
                end for
                if $Failure$ **then**
                    Reset the environment and add failure cost to the episode's reward function $r(s, a)$
                    Break
                end if
            end if
        end for
        add episode data to D
    end for
    train policy π_{TRPO} with D (Eqn. 2.14)
    decrease the value of ρ ▷ ρ decrease by constant predefined value
end for

parameter ϕ^* and co-variance Σ will be selected manually by choosing the standard deviation.

One important step for our algorithm is to choose an appropriate standard deviation. The parameters ϕ^* describe a conservative policy (Conservative policy means that the policy is directed to one of the local minmus, in our case the model-based performance) that performs the task according to the model-based controller behaviour. This conservative police can be completely unsafe which means we need to decrease the conservativeness of the policy and increase the possibility of having safe different actions sampled from the model-free policy that form complete safe sequence of actions to achieve the task. We can solve that by choosing appropriate value for the standard deviation. Another reason for choosing appropriate value for the standard deviation, is to increase the exploration rate to find higher performance (in terms of the average return) than the model-based one. Standard deviation is considered as a hyberparameter that you need to tune. Low standard deviation leads to conservative policy. On the other hand, high standard deviation diverts the policy from imitation policy performance or from even performing the task.

2.2.3.2 Safe Interaction with Hybrid Learning

The agent interacts with the constrained environment. This means that the state and action-spaces are tight. The agent uses the MPC-like controller that has two main properties of the normal MPC form the control theory, which are the idea of receding horizon using the dynamical model, we use this for violation checking, and the concept of applying the first action component in optimized action sequence. As we mentioned before in the model-based stage, it is difficult to optimize for a sequence of actions with a non-liner dynamical model and constraint function. This is the reason we generate number of trajectories with actions sampled from the model-free policy $a \sim \pi_{TRPO,\phi}$, instead of random actions to make use of the model-free algorithms to achieve high asymptotic performance, rather than optimize for the sequence of actions.

The agent receives state s_t from the environment. We generate K imaginary trajectories with H prediction horizon,

$$\tau_t^{(H)} = (s_t, a_t, ..., \hat{s}_{t+H-1}, a_{t+H-1}, \hat{s}_{t+H}) \tag{2.10}$$

where $a_t \sim \pi_{TRPO,\phi}(. \mid s_t)$ and $\hat{s}_{t+1} = \hat{f}_{\theta^*}(\hat{s}_t, a_t)$. We can include all the trajectories in one set T,

$$T = \left\{ \tau_i^{(H)} \mid i = 0, ..., K,\ H > 0 \right\} \tag{2.11}$$

We filter the trajectory set T from the unsafe trajectories by using constraint func-

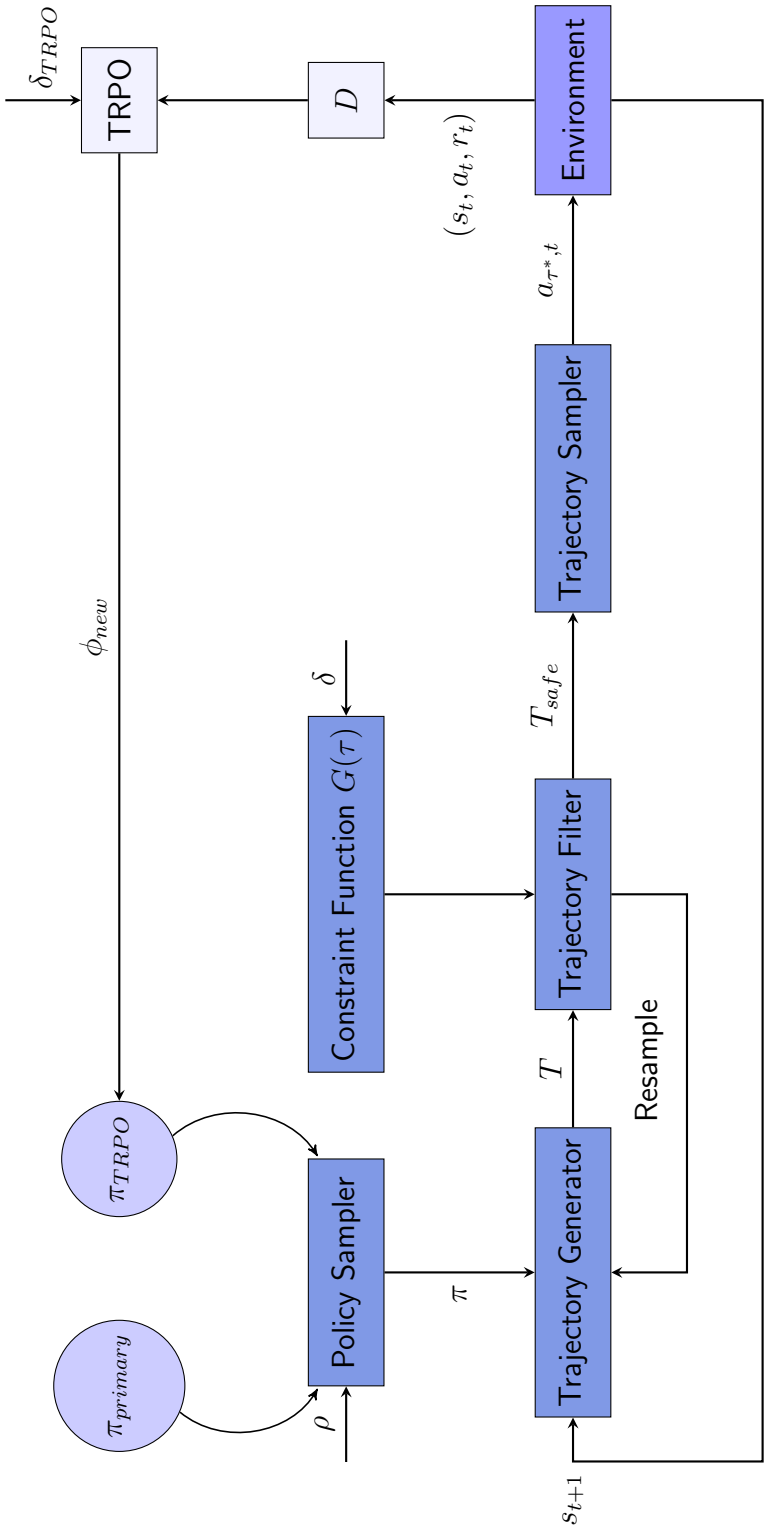

Figure 2.5: Hybrid constrained MPC algorithm flow chart.

tion,

$$G(\tau_i) = \sum_{t'=t+1}^{t+H-1} g(s_{t'}) : \tag{2.12}$$

$$g(s_{t'}) = \begin{cases} 0, & No\ violation \\ 1, & Violation \end{cases}$$

After filtration, we have a safe set of trajectories which is a sub-set of the initial set $T_{safe} \subset T$,

$$T_{safe} = \left\{ \tau_j^{(H)} \mid j = 0, ..., m,\ m \leq K,\ G(\tau_j) = 0 \right\} \tag{2.13}$$

We have a safe set of trajectories that we can sample a trajectory from it to interact with the environment. In the model-based stage, we evaluate the trajectories by cost function $C(\tau)$ as in Eqn. 2.7. However, we randomly sample a trajectory τ^* from T_{safe} to allow exploration from the current policy to reach a performance better than the model-based one. We will not apply the full trajectory on the environment, but according to the MPC controller we are using, we apply only the first action $a_{\tau^*,t}$ to address the problem of model mismatch which it is critical in the constrained environment. After T time steps, we have a complete closed-loop trajectory τ_c that is evaluated by reward function $r(s, a)$ for performing the task.

After interacting with the environment for N_{ep} episodes, we train the policy using the model-free algorithm (TRPO in our case) by the data collected from the interaction. We update the parameters ϕ for the current policy π_k by,

$$\begin{aligned} \pi_{k+1} = \underset{\pi \in \Pi_\phi}{\arg\max} \quad & E_{\tau_c} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \\ & s.t. \quad G(\tau_c) = 0, \quad \bar{D}_{KL}(\pi \mid\mid \pi_k) \leq \delta_{TRPO} \end{aligned} \tag{2.14}$$

where k is the iteration count, $\bar{D}_{KL}(\pi \mid\mid \pi_k) = E_{s \sim \pi_k} \left[D_{KL}(\pi \mid\mid \pi_k)[s] \right]$, and $\delta_{TRPO} > 0$ is the step size for the policy update. The set $\left\{ \pi_\phi \in \Pi_\phi : \bar{D}_{KL}(\pi \mid\mid \pi_k) \leq \delta_{TRPO} \right\}$ is the trust region. Then we use ϕ_{new} of the π_{k+1} for the interaction with the environment of the next iteration. There are some practical issues that we can handle to reach to the desired performance. In the next section, we propose solutions for this problems which help for achieving our target.

2.2.3.3 Extended HCMPC

Learning from Failure: The ideal case of this algorithm to find safe trajectory each time step, but this is not always the case. There will be some time steps where

the agent is stuck in a state with empty safe trajectory set T_{safe}. We call this episode a failed one. The important question in this case, how can we punish the agent for these failed episodes to learn that it is not a good idea to take the same path. The we use the concept of learning from failure. Example for learning from failure that we can find around us, the child who is trying to hold a hot cup of tea, when he hold it in a wrong way, it hurts, then he will not do this again as he got the appropriate punishment that will make him stay away.

we use the same concept to solve our problem. Let's assume that our agent is stuck at time step $t_{failure}$ in a state $s_{failure}$, then we resample again to find a safe trajectory. We do this resampling process for N_r iterations. If we got a safe trajectory, then we will use it to move to the next step. If we did not find one, we will terminate this episode by resetting the agent to an initial position and add a failure cost to the reward function for this episode at this time step $t_{failure}$. This ensures that there will be no violation for any time step and the policy's actions will divert the agent from this failed states. We will see the effect of the learning from failure concept by showing results in Ch. 3.

Model Mismatch: Although the learned dynamical model $\hat{f}_{\theta^*}(s, a)$ is sufficient for the agent to perform well in the model-based stage where the agent interacts with unconstrained environment, the model mismatch is critical when we deal with constrained environment especially a high dimensional one. We need to increase the the safety probability by decreasing the model mismatch. We solved this problem by introducing boundary layer δ that can be added to the constraints and agent dimensions as shown in the representation in Fig. 2.6. This increases the gap between the agent and the constraints which compensates the error from predicting the next state by the dynamical model. The boundary layer δ is added to the calculation in the constraint function $G(\tau)$ to determine the violation in the next states. Choosing the value of the boundary layer δ is critical which is considered as a new hyperparameter we need to tune. We need to find a value that balances the trade-off of having low value that may not solve the problem and cause violation as the model mismatch is relatively high , or a high value that make the state space tighter, so eliminate some possible local minima. We show the effect of the boundary layer in the results in Ch. 3.

Inconsistency in the final performance: In reinforcement learning, the performance is not always the same, even if we train the agent with the same environment and the same parameters. The final performance differs approximately within small range. The reasons for that are the random initialization for the parameters and the stochasticity of the policy. Each training trial, we are sampling from a stochastic policy to explore the possible combination to find the optimal actions and that may lead to different final converged performance.

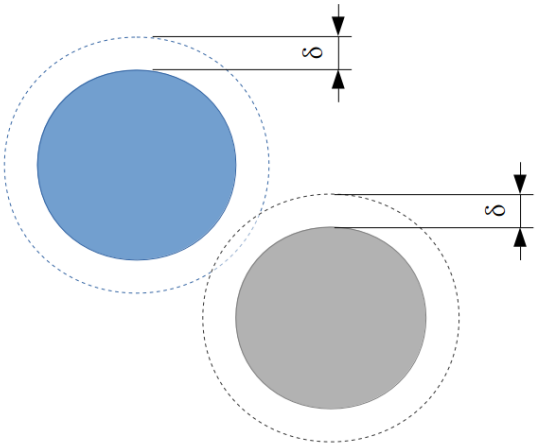

Figure 2.6: Boundary layer representation

In our algorithm, we noticed that such a difference in the performance each trial. The reason for that the random sampling from the safe trajectory set T_{safe} which will lead to different possible direction for convergence of the policy. The conservative policy increases the problem as the convergence of this policy will be faster than policy with random parameters. This means that the exploration of the possible action combinations, that can be sampled from the policy, will be small.

We solve this problem by adjusting the step size δ_{TRPO} within the policy update process. Then the leaning rate becomes slower and this helps for the policy to have sufficient number of iterations to find the same performance each trial.

We also increase the exploration rate. We explore from the initial model-free policy we started the final stage with and exploit from the current policy. We call this policy, the primary policy $\pi_{primary}$. We sample each time from $\pi_{primary}$ by some probability of exploration ρ. Decrementing the exploration probability ρ by some constant value each training iteration. ρ is considered as a hyperparameter that we need to tune to balance the trade-off of exploration and exploitation.

To summarize, our algorithm trains the agent for learning the task in constrained environment with a main target of satisfying the safety conditions. We balance the trade-off of achieving sample efficiency and high asymptotic performance by combining model-based and model-free algorithm in our method. We have a general learned dynamical model that we can use to learn any other task just by designing the cost function. Our contribution is presented in the last stage by introducing the constraints on the learning process. For a better explanation for the last stage, we show the main steps in Alg. 2 and Fig. 2.5. We will show the validity of our algorithm in Ch. 3 on a two degree of freedom robotic arm using mujoco physics engine [TT12].

Chapter 3

Results

In this chapter, we will show the strength of our algorithm for solving the problem of handling constraints within the reinforcement learning framework. In addition, the ability to balance between achieving sample efficiency and high asymptotic performance. At the beginning, in Sec. 3.1, we describe the environment on which we apply our algorithm. Then, in Sec. 3.2, we evaluate the model-based stage. In Sec. 3.3, we evaluate the design decisions for our main contribution (HCMPC stage). Each result is a 10 trials experiment which is represented as a mean solid curve with standard deviation region for showing the variance between the different trails.

3.1 Environment

We evaluate our algorithm on agent in MuJoCo [TT12]. The agent we use is the reacher environment ($\mathcal{S} \in \mathbb{R}^{18}, \mathcal{A} \in \mathbb{R}^2$) which is a continuous control benchmark task. At this environment, 2 DOF planar robotic arm tries to reach a sphere goal in the state space while avoiding a constraints. The constraint is a sphere obstacle as shown in Fig. 3.1.

3.2 Evaluating the Model-Based Stage

The number of iteration, for the model-based algorithm, is depend on the type of the task and the complexity of the environment. For our task, our dynamical model is just trained by the random dataset D_{RAND}, even the performance of the MPC controller is saturated after one iteration. However, We applied the model-based stage on more than one environment and we show the performance in the Appendix.

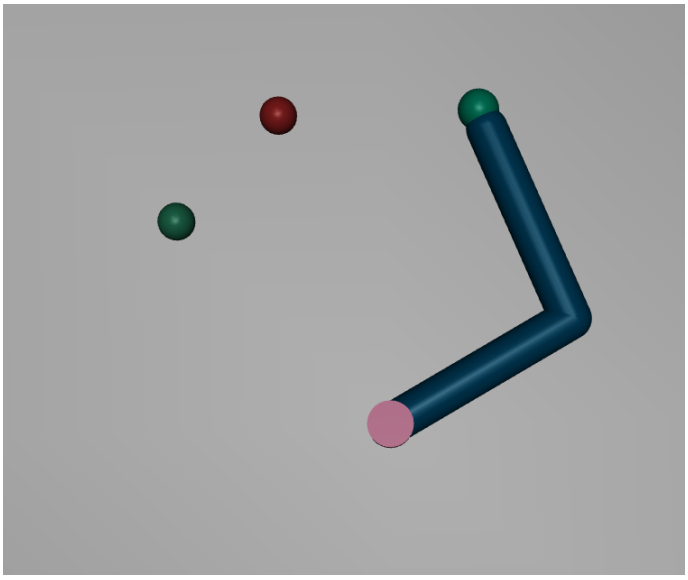

Figure 3.1: This figure represents a 2 DOF planar robotic arm with a green sphere goal and red sphere obstacle.

3.3 Evaluating Design Decisions for HCMPC

We show in this section the effect of each of the main hyperparameters in our algorithm. We study the effect of standard deviation initialization in Sec. 3.3.1 for the model-free policy (π_{TRPO}), the advantage of the boundary layer δ to solve the model mismatch problem in Sec. 3.3.2, the trade-off between the standard deviation value and the boundary layer in Sec. 3.3.3, learning from failure effect in Sec. 3.3.4, TRPO step size for solving the inconsistency in the performance in Sec. 3.3.5. Finally, we compare our approach, in Sec. 3.3.6, with state of the art TRPO algorithm with violation cost within the reward function.

3.3.1 Standard Deviation Effect

Standard deviation is a critical hyperparameter in our algorithm. Standard deviation affect the performance in terms of convergence and response. The parameters ϕ^* of the imitation policy that are used for the initialization of model-free policy π_{TRPO} represent the performance reached by the model-based controller. This means that either this performance is low comparing to the desired performance or unsafe which may not coverage to any safe performance. This is the reason why we need to choose manually the standard deviation to allow exploration to find high and safe asymptotic performance.

We show in Fig. 3.2 the performance of different value of the standard deviation

on the reacher benchmark. From the results, we can see that for low standard deviation (std=0.5), we are approaching a conservative policy which it may converge to low asymptotic performance for some trials, or for others, it may not find safe performance. This causes increase in the variance in the result. For very high standard deviation (std=3.0), we are approaching a random policy which Loses what it learned in the model-based stage. This leads to a slow response which means sample complexity problem. Intermediate values for the standard deviation (std=1.3 or std=1.7) lead to high and safe asymptotic performance.

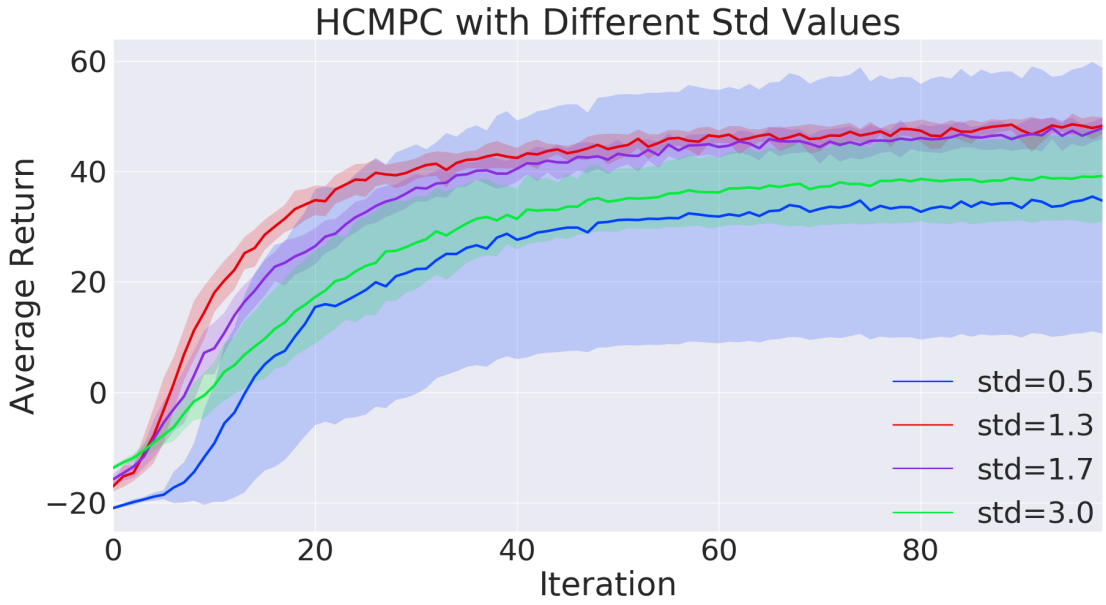

Figure 3.2: The performance of our algorithm with different standard deviation values where the boundary layer δ is 0.003 and TRPO step size δ_{TRPO} is 0.001.

3.3.2 Boundary Layer Effect

Boundary layer, δ, is another key hyperparameter to be adjusted to satisfy the safety requirements. As we discussed in Sec. 2.2.3.3, the boundary layer δ is used to tackle the model mismatch problem. We show in the result below in Fig. 3.4 the comparison between the violation cost with and without the boundary layer. As we see the boundary layer δ succeeded to achieve zero violation cost. On the other hand, without using the boundary layer, there are violations through out the learning process. For the performance, we can see in Fig. 3.3 that by using boundary layer, the performance becomes slightly low. This is an important point we need to discuss. After using the boundary layer, the state space becomes tight. This is why, we need to choose appropriate value for δ that keeps zero violation for the

constraints and still with high performance. We show also in Fig. 3.5d the boundary violation cost which means how many times we violate the boundary layer region.

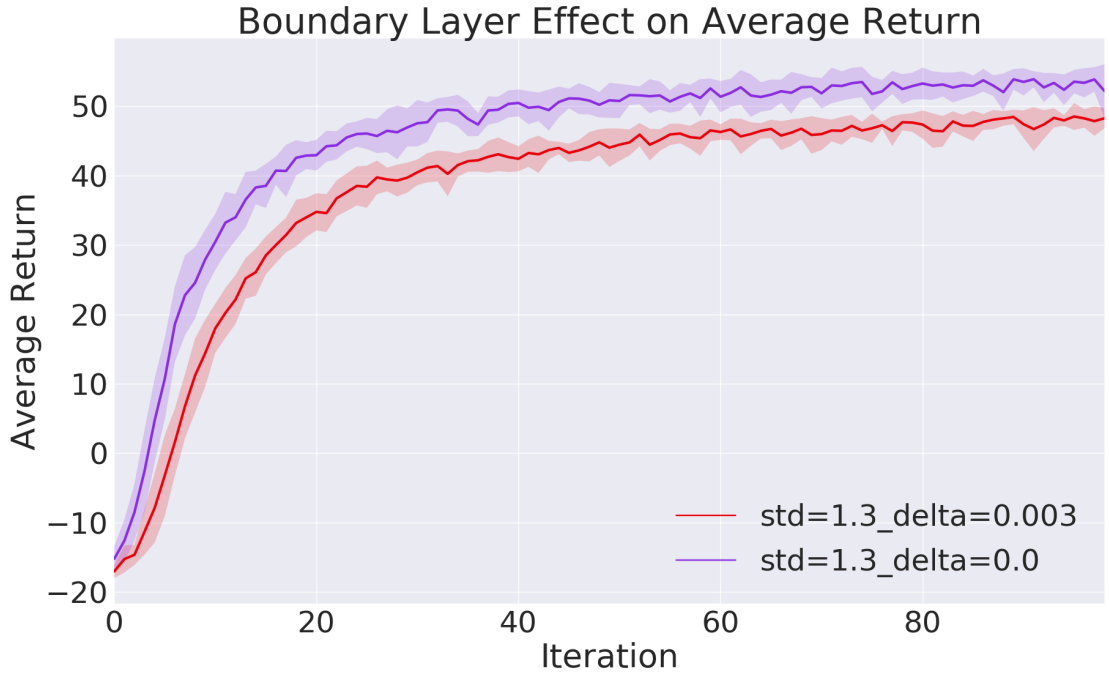

Figure 3.3: Boundary layer effect on the average return with std is 1.3.

3.3.3 Standard Deviation Vs. Boundary Layer

In this section, we show the effect of the standard deviation over the violation cost. We have two types of violations, actual violation and boundary violation (the violation happens over the safe boundary layer around the constraint). As we see in Fig. 3.5, the violation cost with constant boundary layer δ value and different values for the standard deviation. As we increase the standard deviation the actions taken by the policy are unexpected by the learned model, so the mismatch increases. This is the reason for the increased the violation cost in the results for standard deviation 1.7 and 3.0. On the other side, as the standard deviation decreases, we approach a policy that is similar to the model-based controller, the model can predict well the next state as the data that the model is trained with coming mainly from the model-based controller.

As we adjust the value of the boundary layer δ to show that we can satisfy the safety requirements for different possible values of the standard deviation. We show those results in Fig. 3.6. For our experiment, we can add a very small value for boundary layer for standard deviation 1.3 to get the preferred performance. Then, for high

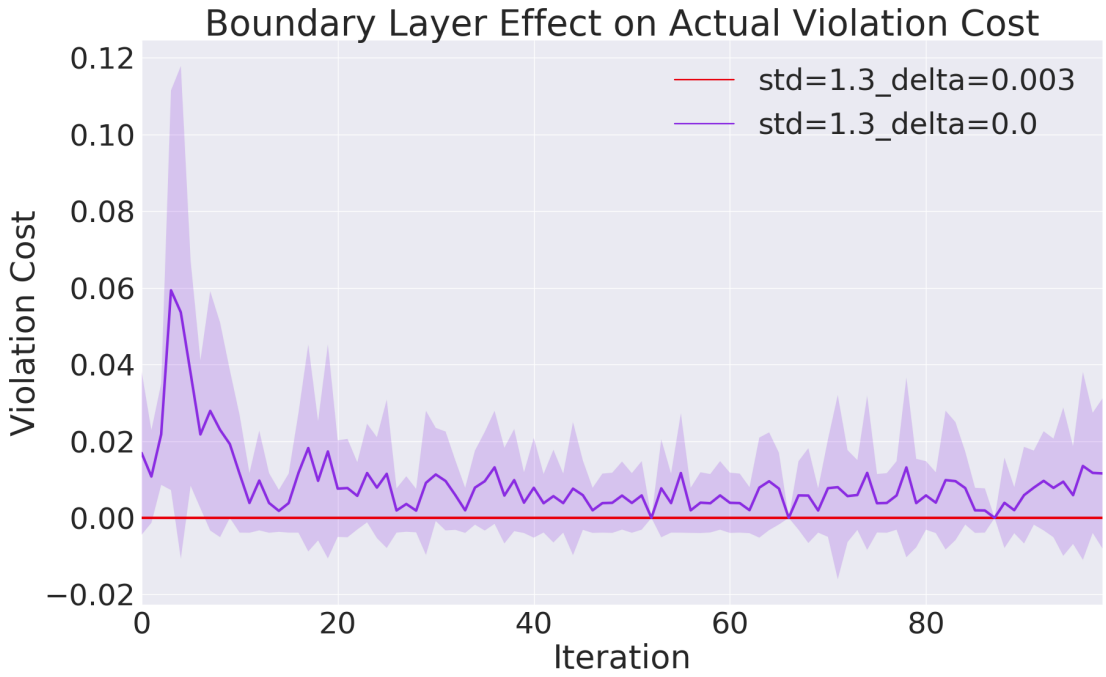

Figure 3.4: Boundary layer effect on the actual violation cost with std is 1.3.

standard deviation values, these are extreme cases that lead in most of the case to slow response performance and that is not what we want to achieve.

3.3.4 Learning from Failure with Violation Cost

Introduce the learning from failure concept within the learning process is considered as very important step in our proposed approach. We add failure cost to the reward function at each failed episode. This failure cost is dynamic according to the time step at which the failure happens. The reward function for our environment is described in details in the Appendix. We measure the validity of these step by calculation a success rate which is the ratio between the successful episodes to the total number of episodes. In Fig. 3.7, we list all the results for different standard deviation that represent this success rate. As we see all the success ratio approaching 1.0 except standard deviation 0.5 that show a high variance in the ratio as in the average return performance (Fig. 3.2). This is due to the conservativeness of the policy which has a probability of reaching unsafe final performance for some trials.

3.3.5 TRPO Step Size Effect

One of the problems we faced as mentioned in Sec. 2.2.3.3 is the inconsistency in the results. We tackle this issue by tuning the value of the step size for the policy update

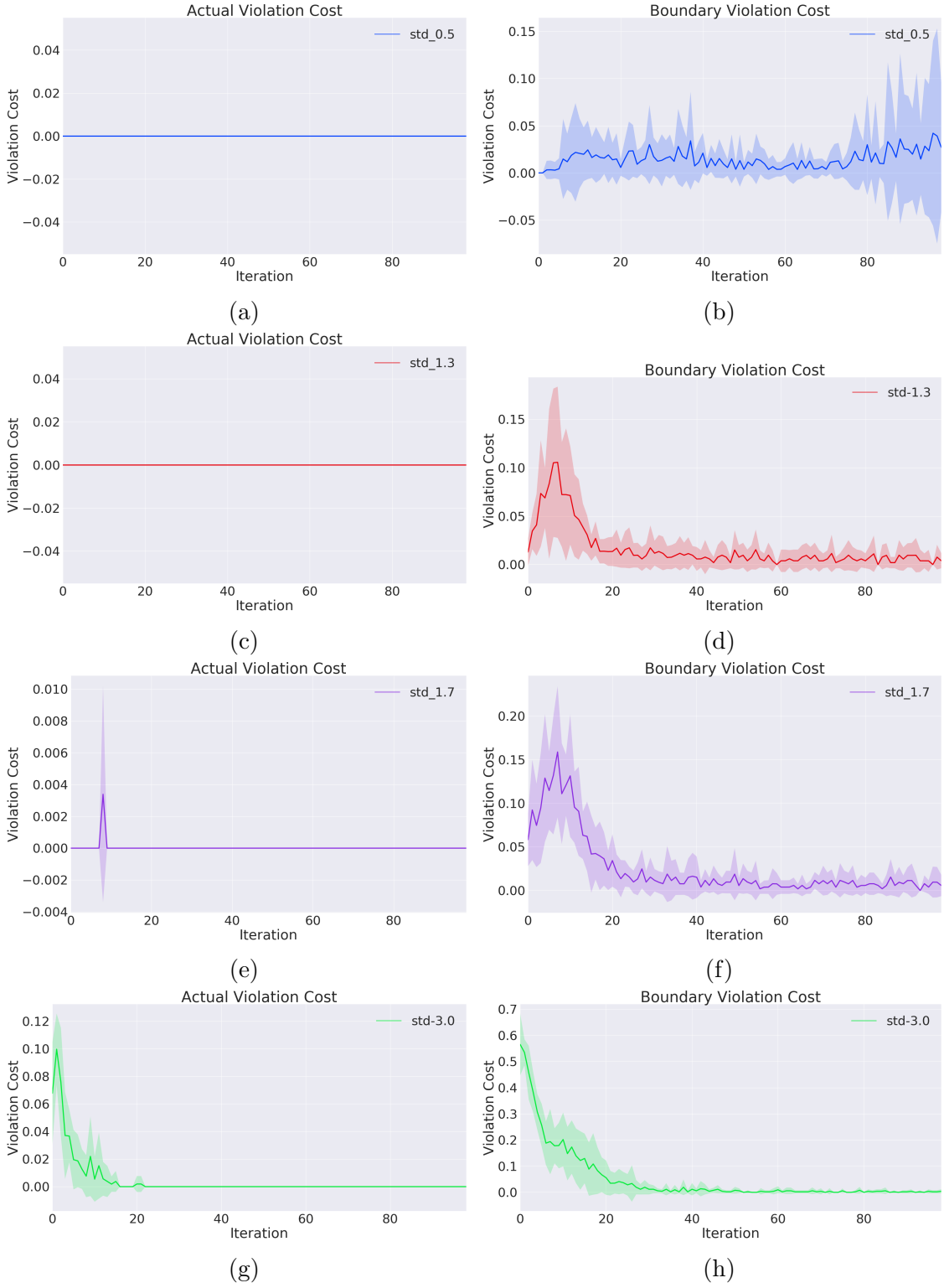

Figure 3.5: (a, c, e, g) show the boundary layer effect of value 0.003 over the actual violation for different standard deviation values, while (b, d, f, h) show the boundary layer violation for different standard deviation values too.

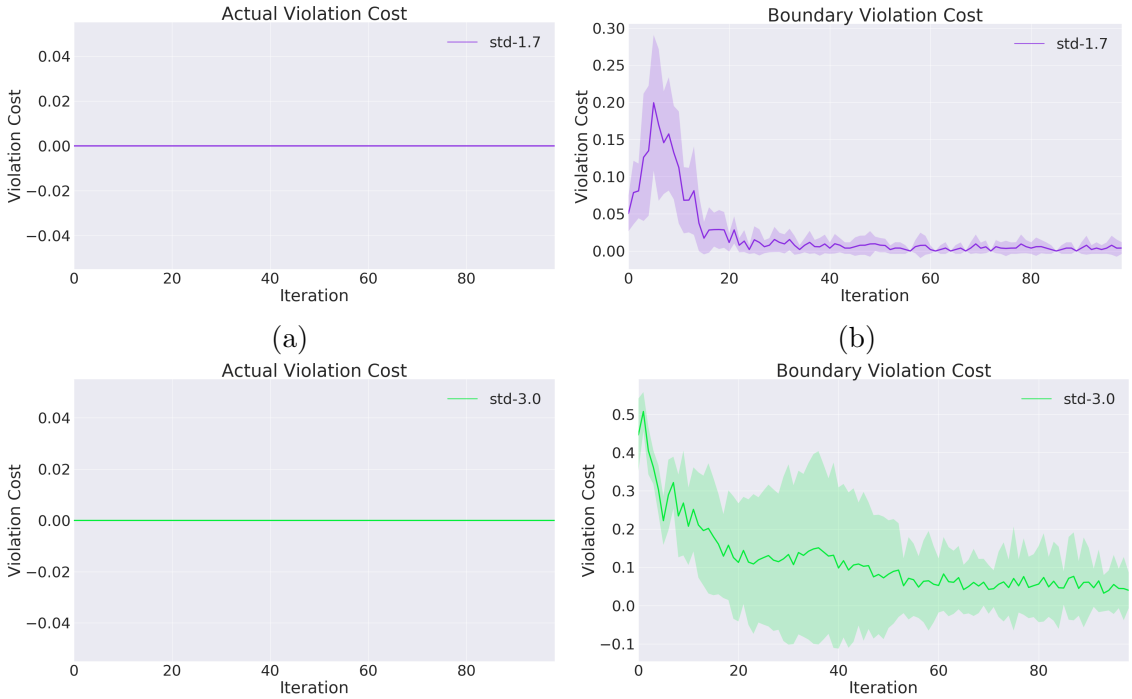

Figure 3.6: (a,b) show the actual and the boundary violation costs after increasing the boundary layer value from 0.003 to 0.006, while (c, d) show the actual and the boundary violation costs after increasing the boundary layer from 0.003 to 0.015.

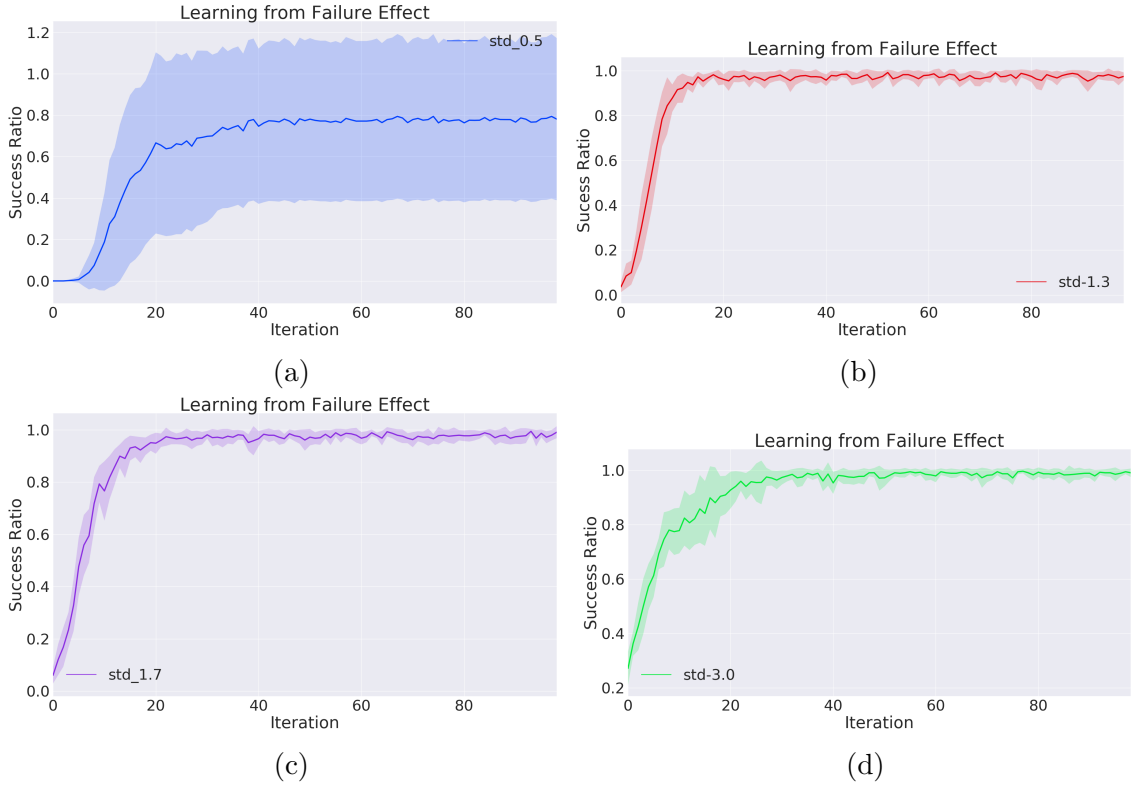

Figure 3.7: These plots show the effect of the learning from failure concept on different standard deviation values.

in the TRPO algorithm. This helps to slow down the convergence of the policy for the purpose of exploration and find consistent performance each trial of learning. The performance is affected by tuning the step size by decreasing the value from 0.05 to 0.01 as shown in Fig. 3.8. The results for step size 0.05 have high variance which means different possible final performance for each trial. On the other hand, the variance decreases with the decrease of the step size value to 0.01.

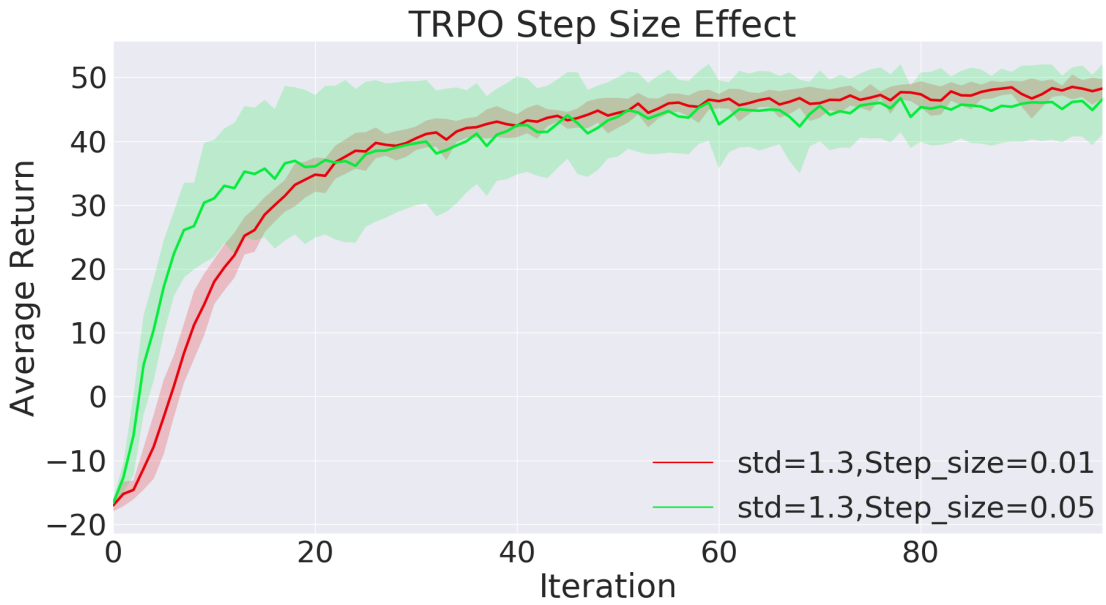

Figure 3.8: Boundary layer effect on the average return with std is 1.3.

3.3.6 Comparison to State-of-the-Art Algorithms

In this section, we compare our results in terms of average return performance and constraint violation cost with the state of the art model-free algorithm TRPO with different values for the violation cost added to the reward function. Due to the hierarchy of our algorithm in terms of using model-based and model-free algorithms in one pipeline, we achieve sample efficiency. As we see in Fig. 3.9, the response of our algorithm is faster than the response of TRPO for different violation cost values. We can see that TRPO with violation cost 20 is also faster than that with violation cost 100. In addition, our asymptotic performance is high comparing to the performance of TRPO (even the reward function is not the same but at the end all the performances become mostly safe, then we can compare them). From the violation point of view, our algorithm satisfy the constraint condition within the learning phase of the last stage. On the other hand, TRPO violates many times until it learns how to avoid the constraint. We show this results in Fig. 3.10.

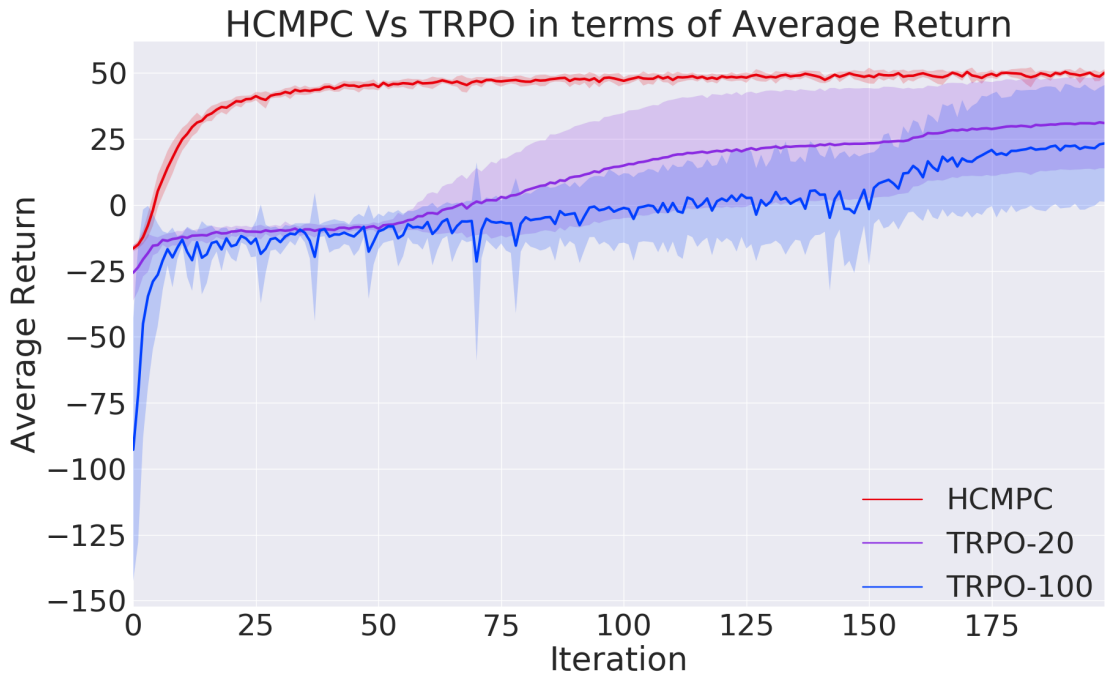

Figure 3.9: Comparison in terms of the average return performance for our HCMPC stage and TRPO with different violation costs.

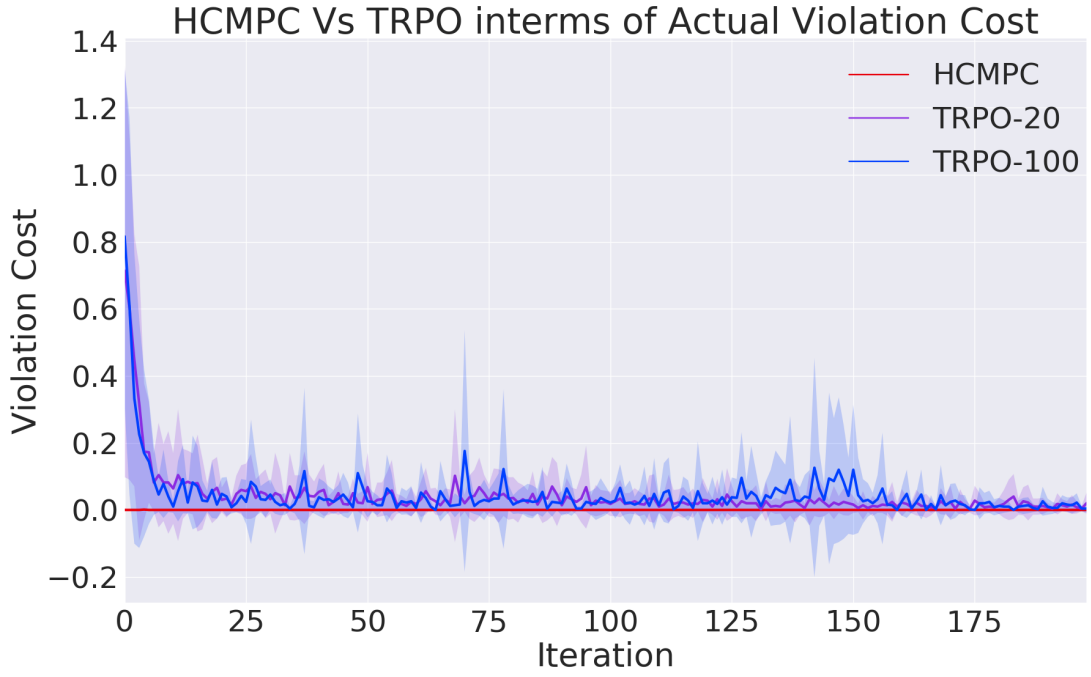

Figure 3.10: Comparison in terms of the actual violation cost for our HCMPC stage and TRPO with different violation costs.

Chapter 4

Conclusion

In this work, we construct a hybrid reinforcement learning algorithm that is capable of satisfy constraint conditions within each step of learning. This is achieved by using the concept of receding horizon, from MPC theory, for violation checking over a finite horizon. In addition, the hybrid structure balances the well-known trade off in reinforcement learning between achieving sample efficiency and high asymptotic performance. In the result of using a model-based stage for learning the dynamical model, we decrease the necessary samples for the learning process comparing to the model-free algorithms after using the dynamical model with MPC-Like controller as a initial good performance for the model-free algorithm. Thus, we make use of the model-free algorithm for achieving the high asymptotic performance.

We evaluate our algorithm performance on a constrained reacher environment. A planar robot learn reachability task with constraint condition satisfaction with high final performance for the task. Our approach outperforms against state of the art model-free algorithm (TRPO) with violation cost added to the reward function. Our algorithm highly perform in terms of constraint handling, number of samples and the asymptotic performance.

Our work handle the mismatch issue in the dynamical model by introducing a safe boundary layer, but to increase the probability of constraints satisfaction for high dimensional environments, we need more effective way. Still, our work is subject to some limitations in its current implementation. While we showed our constraints satisfaction, but it needs more formal proof for the constraints handling.

4.1 Future Work

To extend our approach for more complex environments and with high guarantee for constraints satisfaction, we need to include optimization layer for generating such

safe trajectories. For the model mismatch problem, we can make use of the idea of model ensembles in [KCD+18]. In this work, the dynamics is approximated by using multiple neural network with different initialization, so it can handle the mismatch problem effectively. In addition, our aim is to apply our algorithm for locomotion tasks and even on real systems to validate our approach. Finally, we can increase the complexity of the constraints by handling dynamic constraints or more challenging constraint like human.

Appendix A

A.1 Experimental Details for HCMPC

We validate our approach on the reacher environment with different values of the hyperparameters. The reward function for our environment, consists of three main components as shown,

$$r(s, a) = -d_{dist} + d_{prox} - d_{failure}$$

where d_{dist} is the distance between the end effector of the robot and the goal, d_{prox} is a bonus added when the end effector of the robot within some region around the goal and $d_{failure}$ is the failure cost for the failed episodes which can be described by,

$$d_{failure} = n_{remaining} * d_{dist}$$

where the $n_{remaining}$ is the remaining time steps for the episode.

A.2 Model-Based Results on different tasks

In this section we can see the performance of the model-based stage on two different environments (Half cheetah and cart-pole). We show the results below,

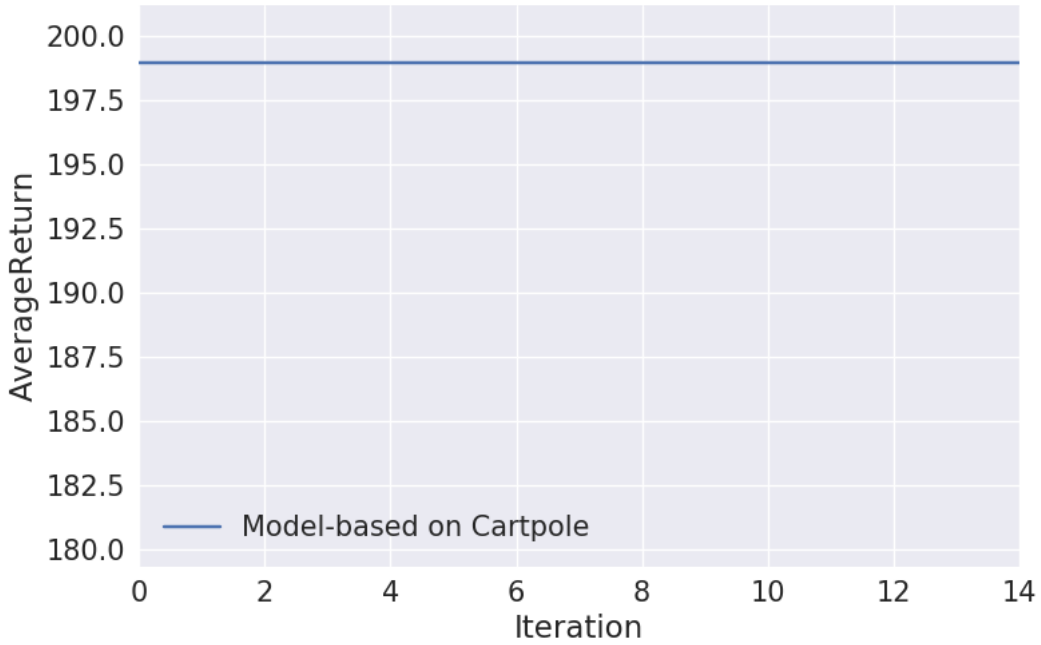

Figure A.1: Model-based stage performance on cart-pole environment

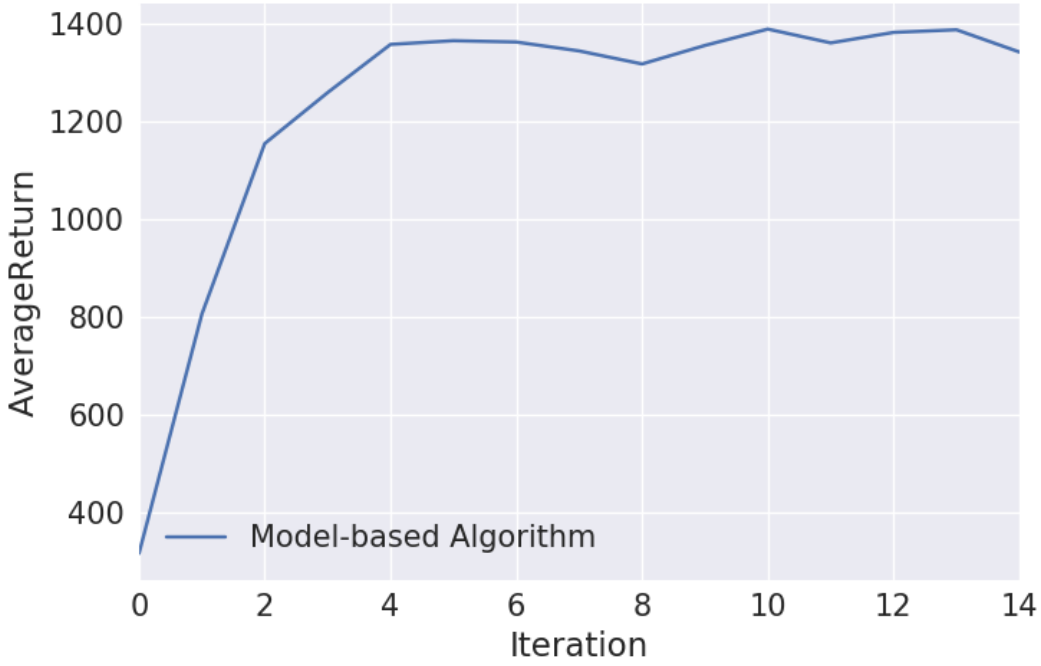

Figure A.2: Model-based stage performance on half-cheetah environment

List of Figures

Bibliography

[AE15] Haitham Tutunov Rasul Ammar, B. and E. Eaton. Safe policy search for lifelong reinforcement learning with sublinear regret. *International Conference on Machine Learning*, 2015.

[AHTA17] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. *the 34st International Conference on Machine Learning,Sydney,Australia*, 37:1–10, 2017. URL: `https://arxiv.org/pdf/1705.10528.pdf`.

[AK17] B. Amos and J.Z. Kolter. optnet: Differentiable optimization as a layer in neural networks. *In Proceeding International Conference of Machine Learning*, 2017.

[Alt99] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall, 1999.

[DR11] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. *In Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011. URL: `http://mlg.eng.cam.ac.uk/pub/pdf/DeiRas11.pdf`.

[FA16] Levine S. Fu, J. and P. Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. *In Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference*, pages 4019–4026, 2016. URL: `https://ieeexplore.ieee.org/document/7759592/?part=1`.

[FT17] Akametalu A.K. Zeilinger M.N. Kaynama S. Gillula J. Fisac, J.F. and C.J. Tomlin. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*, 2017.

[GF15] J. Garcia and F. Fernande. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, pages 1437–1480, 2015.

[HT00] Wayne G. Silver D. Lillicrap T. Erez T. Heess, N. and Y. Tassa. Learning continuous control policies by stochastic value gradients. *In Advances in Neural Information Processing Systems*, pages 2944–2952, 2000.

[KB14] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014. URL: `https://arxiv.org/pdf/1412.6980.pdf`.

[KCD+18] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. Model-ensemble trust-region policy optimization. *International conference on learning representation*, pages 1–15, 2018. URL: `https://openreview.net/pdf?id=SJJinbWRZ`.

[KH17] M. Kimmel and S. Hirche. Invariance control for safe human-robot interaction in dynamic environments. *IEEE Transactions on Robotics*, 33:1327 – 1342, 2017. URL: `https://mediatum.ub.tum.de/doc/1346232/523152.pdf`.

[KK18] Berkenkamp F. Turchetta M. Koller, T. and A. Krause. Learning-based model predictive control for safe exploration and reinforcement learning. 2018.

[LA6a] Finn C. Darrell T. Levine, S. and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17:1–40, 2016a.

[LK17] Jang Y. Poupart P. Lee, J. and K. Kim. Constrained bayesian reinforcement learning via approximate linear programming. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, 2017.

[LT04] W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. *In Proceedings of 1st International Conference on Informatics in Control, Automation and Robotics*, pages 222–229, 2004.

[LW15] Hunt J.J. Pritzel A. Heess N. Erez T. Tassa Y. Silver D. Lillicrap, T.P. and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, 2015.

[MK16] Badia A.P. Mirza M. Graves A. Harley T. Lillicrap T.P. Silver D. Mnih, V. and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. pages 1–28, 2016. URL: `http://arxiv.org/abs/1602.01783`.

[MO15] Kavukcuoglu K. Silver D. Rusu A.A. Veness J. Bellemare M.G. Graves A. Riedmiller M. Fidjeland A.K. Mnih, V. and G. Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[MS00] Rawlings J.P. Rao C.V. Mayne, D.Q. and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 2000.

[NKFL17] A. Nagabandi, G. Kahn, R. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *Conference on Neural Information Processing Systems*, 37:1–8, 2017. URL: `https://arxiv.org/pdf/1708.02596.pdf`.

[OS15] Guo X. Lee H. Lewis R.L. Oh, J. and S. Singh. Action-conditional video prediction using deep networks in atari games. *In Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.

[PT17] Magistris G.D. Pham, T.H. and R. Tachibana. Optlayer - practical constrained optimization for deep reinforcement learning in the real world. *arXiv preprint arXiv:1709.07643*, 2017.

[Rao09] A. Rao. A survey of numerical methods for optimal control. *in Advances in the Astronautical Sciences*, 2009. URL: `http://www.anilvrao.com/Publications/ConferencePublications/trajectorySurveyAAS.pdf`.

[SA16] Moritz P. Levine S. Jordan M. Schulman, J. and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *In International Conference on Learning Representations (ICLR2016)*, 2016.

[SB17] Schrittwieser J. Simonyan K. Antonoglou I. Huang A. Guez A. Hubert T. Baker L. Lai M. Silver, D. and A. Bolton. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[SLM+15] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. *the 31st International Conference on Machine Learning*, 37:1–9, 2015. URL: `https://arxiv.org/pdf/1502.05477.pdf`.

[SP16] Huang A. Maddison C.J. Guez A. Sifre L. Driessche G.V.D. Schrittwieser J. Antonoglou I. Silver, D. and V. Panneershelvam. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[SY17] Abdou M. Perot E. Sallab, A.E. and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *IST Electronic Imaging, Autonomous Vehicles and Machines 2017*, pages 70–76, 2017. URL: `https://arxiv.org/abs/1704.02532`.

[TT12] Erez T. Todorov, E. and Y. Tassa. Mujoco: A physics engine for model-based control. *Intelligent Robots and Systems (IROS), 2012*

IEEE/RSJ International Conference, pages 5026–5033, 2012. URL: https://ieeexplore.ieee.org/abstract/document/6386109/.

[UD07] E. Uchibe and K. Doya. Constrained reinforcement learning from intrinsic and extrinsic rewards. *IEEE 6th International Conference on Development and Learning, ICDL*, pages 163–168, 2007.

License