Safe Learning for Control

CAROLINE HEIDENREICH

Master's Degree Project at the department of Automatic Control
Supervisor: Mohammad Sadegh Talebi
Examiner: Mikael Johansson

Contents

Chapter 1

Introduction

1.1 Motivation

Applying methods from the field of machine learning to control problems is a promising approach. Many traditional control strategies rely on a model of the system that shall be controlled. As models usually reflect only parts of the reality, those approaches often suffer from poor model accuracy. Learning the control in an online manner by employing standard reinforcement learning methods such as Q-Learning overcomes the shortcomings of traditional model-based control techniques. The problem with applying reinforcement learning methods to control tasks is that the challenge of satisfying constraints during the learning process is not well addressed yet. Safety-critical control applications are therefore not feasible in this framework. A workaround is to assume some knowledge about the system and worst-case disturbances to design a safety-preserving controller that acts when the learning algorithm would cause a constraint violation. The disturbance bounds can then be updated iteratively by applying statistical modelling tools to obtained data.

Applications of this framework could come from many different fields. One simple application is a lawn-mowing robot that learns how to move within a garden without destroying the family's wading pool. Another one could be a unmanned aerial vehicle that should avoid to hit trees, houses or humans. A third possible application of this framework could be an autonomous vehicle. Based on this example, I will further illustrate our approach in the next section.

1.2 General Idea

This thesis implements a safe learning approach on an inverted pendulum. This system has the advantage to only have two states so that it can be analysed neatly and results can be illustrated fairly easy. As pendulums however not really are the "safety-critical applications" you would come to think of first, I will explain the approach based on an autonomous vehicle. Controlling such a vehicle with a standard model-based control technique, requires an accurate model of the state space that might be difficult to obtain. To overcome this challenge, we could apply model-free reinforcement learning algorithms. This implies that the car itself would try a control action (e.g. steering left) and subsequently receive a reward that depends on if the car ended up in a desirable state (the middle of the road), or not (the roadside ditch). Based on this feedback, the vehicle could gradually learn a control policy that assigns control actions which make the car stay in the middle of the road to each state-action pair. An obvious issue with this approach is that one might need to waste quite a lot of cars to roadside ditches before the controller finally learned an accurate policy. This becomes even more critical, if we regard a crowd of people alongside the road instead of a ditch. Therefore, we need to be able to ensure that the car never leaves the road. For example, one could apply a safety-preserving control each time the car gets too close to the edge of the road. An obvious question now is how to determine how close "too close" is. This question is not trivial, because we do have uncertain dynamics. Therefore, we cannot predict exactly in which direction we turn after applying a certain control action. Luckily, there is way out of this dilemma that does not require a certain model of the system (which was something we wanted to avoid initially). Instead, we model the known parts of the vehicle into a state-space model and and think of the unknown parts of the model as an additive state-dependent disturbance. The only thing we then need is a conservative bound of this disturbance. Dependent on this bound and our known control limits, we can predict to which state the vehicle can get in the worst case, i.e. with a disturbance that drives the vehicle perpendicularly away from the middle of the street. Calculating all states that one should avoid because they potentially lead to an unsafe state is what Hamilton-Jacobi-Isaacs (HJI) reachability analysis does. Assuming a malicious disturbance and a control action that is trying to counteract that disturbance, HJI reachability analysis provides us with a safe set around the middle of the street and a safety-preserving control. If the vehicle applies this control at the edges of the safe set, no disturbance will ever manage to push it off the road. Inside the set, the learning controller can freely choose actions to learn a policy. This approach enables us to safely learn a control for the vehicle while staying on the road for all time. The only problem remaining is that the initial estimate for the disturbance bounds might be very conservative and lead to a small safe set. We therefore collect data samples while learning and estimate a less conservative disturbance model based on that. The method employed for disturbance estimation is Gaussian Process (GP) regression. GP regression is a valuable tool because it yields not only an estimate for the disturbance but also a measure for how certain this estimate is. We then can apply the new estimate with a safety margin that depends on the certainty of the estimate to calculate a larger safe set.

In a summary, this thesis introduces a framework for safely learning a control strategy for a given system with an additive disturbance with known bounds. On the basis of the known disturbance bound,s a safe set in which the system can learn safely is estimated with HJI reachability analysis. Within that set, the Reinforcement Learning algorithm can choose learning actions freely as long as the safety-preserving control is applied when the system hits the edges of the safe set. After some learning episodes the disturbance bounds can be updated based on real-world data. To this end, GP regression is conducted on the collected disturbance samples.

1.3 Outline

The remaining parts of the thesis are structured as follows:

- **Chapter 2** provides the reader with the theoretical background that is necessary to understand the details of the implementation. A short introduction is given to reinforcement learning, GP regression and HJI reachability analysis.
- **Chapter 3** describes some related areas of research and presents the approach in [1] that this thesis is largely based on.
- **Chapter 4** introduces the general framework in which the learning takes place. This chapter aims at giving a rough overview of the employed method without going into much detail.
- **Chapter 5** describes the implementation details of the approach. This chapter builds largely on the theoretical background and the solution framework provided in chapters 2 and 4.
- **Chapter 6** presents some of the findings from the implemented approach and briefly compares different methods.
- Finally, **Chapter 7** concludes the thesis and discusses possible suggestions for future work.

Chapter 2

Theoretical Background

2.1 The Reinforcement Learning Problem

The following section aims at giving an overview to the standard reinforcement learning problem and methods and is entirely based on the introduction provided in [2]. Reinforcement learning is a field of machine learning that tackles the problem of how a learning agent should choose actions in order to maximise some notion of cumulative reward. To learn an optimal policy the agent needs to interact with its environment by taking actions and collecting the reward following from these actions. Given a certain state s_t, the agent selects an action a_t according to some policy π_t and finds itself subsequently in a new state s_{t+1} while receiving some immediate reward r_{t+1}. Reinforcement learning algorithms provide different ways of how the agent should modify π to maximise its expected return, where the return at time step t is typically defined as

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}. \tag{2.1}$$

In this equation, T describes the horizon of the problem (which could also be ∞) and $\gamma \in [0, 1]$ denotes the discount factor that can be used to weigh sooner rewards more than later rewards.

An important framework to model reinforcement learning problems are Markov Decision Processes (MDP). To model a problem as an MDP, it is necessary that the system fulfills the Markov property. That means that the system response at time $t + 1$ only depends on the state and action at time t. An MDP is then defined by a quintuple (S, A, T, r, γ), where S is the state space and A is the action space. T describes the transition probabilities of arriving at a state s' given a state s and an action a i.e. $T(s, a, s') = p(s'|s, a)$. $r(s, a)$ is the reward function that assigns rewards to state s and action a and γ is the discount factor that will be further explained below.

Given a certain MDP, reinforcement learning algorithms typically try to estimate value functions that describe “how favourable” a state is, given that starting from that state s a policy π is applied. The state-value function under policy π (or value function for short) V^π is defined in terms of expected return i.e.

$$V^\pi(s) = E_\pi\{R_t | s_t = s\}. \tag{2.2}$$

The value of a state s indicates how favourable that state is. Thus, a higher expected reward implies that a state is more favourable. Alternatively, the value of being in a state s, taking an action a and then following a policy π is defined by the action-value function

$$Q^\pi(s,a) = E_\pi\{R_t | s_t = s, a_t = a\}. \tag{2.3}$$

This action-value function describes how favourable an state-action pair is. Value functions are used to determine how good a policy is. A policy is better than another, if it achieves a higher expected return for all states and thus has a higher value $V^\pi(s)$ for all s. An optimal policy π^* is an optimizer to the function

$$V^*(s) = \max_\pi V^\pi(s), \tag{2.4}$$

where V^* is called the optimal state-value function. More formally, π^* can be written as:

$$\pi(s)^* \in \arg\max_\pi V^\pi(s). \tag{2.5}$$

The relation between V^* and the action-value function Q can then be expressed as

$$V^*(s) = \max_a Q(s,a). \tag{2.6}$$

Given a MDP as defined above, there are algorithms to compute the optimal policy explicitly. The algorithm depends on the formulation of the return R. If the horizon over which the return is calculated is finite, the MDP is called a Finite-Horizon MDP and the optimal policy along with its average reward can be computed using Dynamic Programming. If the horizon is infinite and the reward is discounted, i.e. $\gamma < 1$, the MDP is called Infinite-Horizon MDP with Discounted Reward. If the reward is undiscounted, the MDP is a Infinite-Horizon MDP with Average Reward and the objective is instead to maximise the average reward. Both classes can be solved with two algorithms, Value Iteration and Policy Iteration. As the MDPs in this thesis are Infinite-Horizon MDPs with Discounted Reward, the solution methods for this class will be explained below. The first algorithm, Value Iteration, initializes the values of the states randomly and then updates the values each iteration according to the following expression:

$$V_{k+1}(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} T(s,a,s') V_k(s') \right]. \tag{2.7}$$

This algorithm is based on the Bellman optimality equations that state

$$V^*(s) = \max_a E\left\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\right\} \tag{2.8}$$

$$= \max_a \left[r(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s') \right]. \tag{2.9}$$

For more details the reader is referred to [2]. Value Iteration can be shown to converge to the optimal policy in polynomial time. As Value Iteration requires an infinite number of steps to converge exactly, the execution of the algorithm is usually stopped when the changes in the values become smaller than some value ϵ. Policy Iteration works in a similar way but evaluates and improves a policy π directly. As the policy usually converges much faster than the values, Policy Iteration gives good results when the actual underlying policy is more important than accurate values.

Reinforcement learning algorithms

Reinforcement algorithms in general try to estimate the value function when the MDP is not entirely known, i.e. transition probabilities and reward function are unknown. There are model-based and model-free algorithms. The former ones try to estimate the underlying model to make it more accrately match the real environment. The latter ones try to obtain the optimal policy directly without approximating the underlying dynamics. All algorithms explained below are model-free. Another important distinction is the one between off-policy and on-policy algorithms. On-policy algorithms update the Q-values according to the policy followed while off-policy learns the optimal Q-values independent of the policy followed. All algorithms explained below except for Delayed Q-Learning are off-policy algorithms.

In the following section, some reinforcement learning algorithms are introduced. A classic algorithm in the field of reinforcement learning, known as Q-learning, was presented by Watkins in 1989 [3]. In its simplest form the learning update of this algorithm is formulated as

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \left[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right]. \tag{2.10}$$

In this expression, Q_{t+1} is the learned action-value function, α is the step size or learning rate, and r_{t+1} is the reward obtained by taking action a_t from state s_t and arriving in the succeeding state s_{t+1}. Per iteration only one state-value pair is updated such that

$$Q_{t+1}(s,a) = Q_t(s,a) \qquad \forall (s,a) \neq (s_t, a_t). \tag{2.11}$$

Q approximates the optimal action-value function Q^*. Convergence is ensured as long as the following two conditions on the sequence of step sizes α are fulfilled [4]

$$\sum_{t=1}^{\infty} \alpha_t = \infty \tag{2.12}$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty, \tag{2.13}$$

and all state-action pairs are visited infinitely often as the number of transitions approaches infinity. The former condition is for instance fulfilled with a step size of $\alpha = \frac{1}{n+1}$ where n is the iteration count. The latter condition deals with the trade-off between exploration and exploitation. This describes the need to find a balance between applying the policy that is currently estimated to be best and exploring unknown regions of the state space. One way to ensure this balance is to employ an ϵ-greedy policy that makes the agent choose a random action with probability ϵ and the action maximising $Q(s, a)$ elsewise. Alternatively, the Q-values could be optimistically initialized to the maximum possible value V_{max} (for an infinite horizon) that is achieved when the agent would always receive the maximum possible reward.

$$Q_0 = V_{\text{max}} = \sum_{k=0}^{\infty} \gamma^k \max_{s,a} r(s, a) = \frac{\max_{s,a} r(s, a)}{1 - \gamma}. \tag{2.14}$$

In this case exploration is encouraged because the algorithm will in the beginning of the learning process lower the values from all experienced states and therefore the choice probability of unexplored states increases.

Classical Q-learning converges under the above described conditions, but it does so slowly. Therefore, several improvements to the original algorithm have been proposed. Speedy Q-learning is a variant of traditional Q-learning that was introduced in 2011 aimed at increasing the convergence speed [5]. To achieve faster convergence, there are slight changes in the update of the Q-values. Instead of only taking the Q-function of the current iteration into account, the update incorporates also the previous Q-function. The learning update becomes

$$\begin{aligned} Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) &+ \alpha_k \left[r_{t+1} + \gamma \max_a Q_{t-1}(s_{t+1}, a) - Q_t(s_t, a_t) \right] \\ &+ (1 - \alpha_k) \left[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - r_{t+1} + \gamma \max_a Q_{t-1}(s_{t+1}, a) \right]. \end{aligned} \tag{2.15}$$

The second term goes to zero as the Q-values converge to the optimal values. Therefore, the usual step size conditions do not need to be fulfilled for this term and the algorithm achieves a larger step size while still ensuring convergence. More detailed convergence results can be found in [5].

Another variation of Q-learning is the Delayed Q-learning algorithm for the online problem. The most major change compared to traditional Q-learning is that the

Q-values are only updated after m samples are collected so that the update rule becomes

$$Q_{t+1}(s_t, a_t) = \frac{1}{m} \sum_{i=1}^{m} \Big(r_{k_i} + \gamma \max_a Q_{k_i}(s_{k_i}, a) \Big) + \varepsilon \tag{2.16}$$

where m and ε are inputs to the algorithm. The algorithm is designed such that for an update to be performed, the change in the Q-value has to be larger than 2ε, where $\varepsilon \in (0, 1)$. Furthermore, the number of attempted updates is limited such that after the update of a state-action pair's Q-value has been attempted m times a successful update of any Q-value has to occur until the update is attempted again. It is then possible to obtain PAC-MDP guarantee [6], where PAC stands for Probably Approximately Correct. This guarantee implies that with high probability an upper bound holds on the sample complexity, i.e. the number of time steps that the agent executes a policy whose value at the current state is not ϵ-close to that of the optimal policy. The details of the convergence proof can be found in [6].

A different approach to Q-learning for large state and action spaces is Q-learning with function approximation. In this method, information from a limited subset of the state space is generalised to cover the whole state space. In case of linear function approximation the Q-function becomes

$$Q_\theta(s, a) = \sum_{i=1}^{n} \theta_{i,a} \varphi_i(s) = \theta^T \varphi \tag{2.17}$$

where θ is the parameter matrix and φ is the vector of basis functions. φ_i can for instance be chosen to be a Gaussian radial basis function

$$\varphi_i(s) = \exp\left(\frac{-||s - m_i||^2}{2\sigma_i^2} \right) \tag{2.18}$$

where m_i is the center and σ_i is the standard deviation of the resulting Gaussian function. These parameters need to be tuned in order for the function approximation to accurately approximates the true function. For instance, the standard deviation of the basis functions determines which distance generalization takes place over. In the t-th iteration of the learning process, a gradient-descent update of one column of the parameter matrix can then be performed in each iteration:

$$\delta_t = r_{t+1} + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t) \tag{2.19}$$

$$\theta_{\bullet, a_{t+1}} = \theta_{\bullet, a_t} + \alpha \cdot \delta_t \cdot \varphi_t(s_t). \tag{2.20}$$

A detailed convergence analysis of this method can be found in [7].

2.2 Gaussian Processes for Regression

The following introduction to Gaussian processes (GP) is based on [8] and [9]. Generally, GPs are powerful non-parametric methods that can be applied for regression and classification. Our specific interest lies in applying GPs to a number of collected data points and obtain an upper and lower bound for a disturbance. Model fitting is a typical application of GP regression. The striking advantage of GP regression in our application is that we not only obtain an estimate for the disturbance, but also a measure for the certainty of this estimate. This enables us to choose upper and lower disturbance bounds that hold with a very high probability. A more formal introduction to GP regression is given in the following. In [9], GPs are defined to be a "collection of random variables, any finite number of which have (consistent) joint Gaussian distributions". GP can be understood as the generalization of Gaussian distributions to a continuous input space and thus an infinite number of random variables. While Gaussian distributions are over finite-dimensional vectors and defined by a covariance matrix and a mean vector, GPs are over functions and defined by a mean function and a covariance function.

GPs can be applied for regression in the following manner. A GP

$$f(x) \sim GP(m(x), \kappa(x, x')) \tag{2.21}$$

with mean function $m(x)$ and positive definite covariance function $\kappa(x)$ can be used as a prior for Bayesian inference. Then, a set of known function values $\mathbf{f}$ associated with training inputs $\mathbf{X}$, and a set of unknown function values $\mathbf{f}^*$, associated with test inputs $\mathbf{X}^*$, are according to the definition above jointly Gaussian distributed. Formally, this can be written as

$$\begin{pmatrix} \mathbf{f}(\mathbf{X}) \\ \mathbf{f}(\mathbf{X}^*) \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right) \tag{2.22}$$

with $\mathbf{K} = \kappa(X, X)$, $\mathbf{K}_* = \kappa(X, X_*)$ and $\mathbf{K}_{**} = \kappa(X_*, X_*)$. For predicting values for the test inputs $\mathbf{f}_*$, the conditional distribution of $\mathbf{f}_*$ given $\mathbf{f}$ can be calculated to be

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f}) = \mathcal{N}(\mathbf{f}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \tag{2.23}$$

$$\boldsymbol{\mu}_* = \boldsymbol{\mu}(\mathbf{X}_*) + \mathbf{K}_*^T\mathbf{K}^{-1}(\mathbf{f} - \boldsymbol{\mu}(\mathbf{X})) \tag{2.24}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T\mathbf{K}^{-1}\mathbf{K}_*. \tag{2.25}$$

The result does not only give a prediction of the unknown function values (the test means) but also a measure of how reliable this prediction is (the test variance). An important result is that the function interpolates the data points exactly in case of noiseless function outputs. An example for a Gaussian prior and a Gaussian posterior conditioned on noisefree data can be seen in figure 2.1. In the lower figure it is evident that the confidence in the prediction decreases with increasing distance to a datapoint.

If the function outputs are corrupted with Gaussian noise, the covariance function of the training inputs is no longer $\mathbf{K}$ but

$$\mathbf{K}_y = \mathbf{K} + \sigma_y^2\mathbf{I}_N \tag{2.26}$$

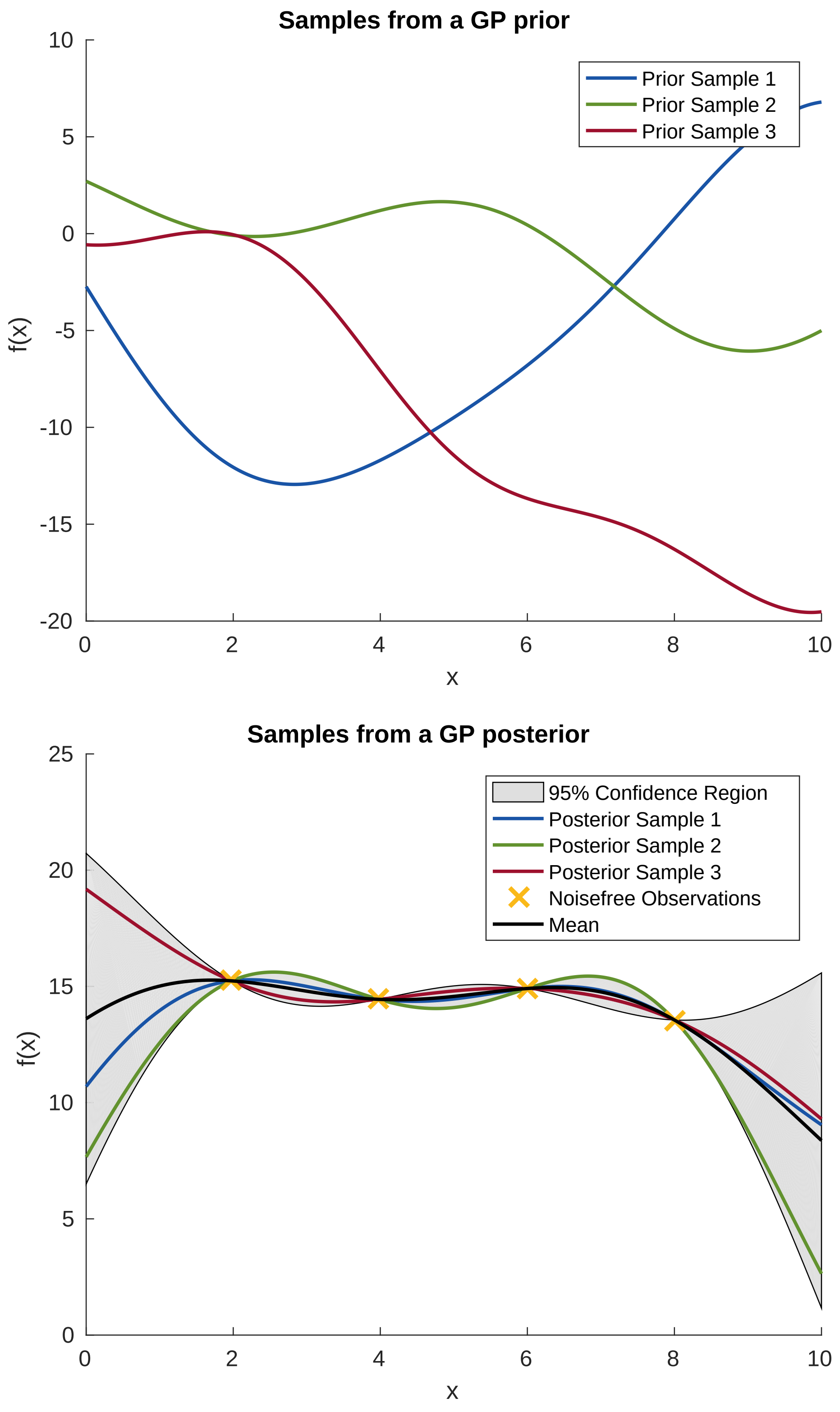


Figure 2.1: The upper figure shows samples from a Gaussian prior with a squared exponential kernel. The lower figure shows samples from the Gaussian posterior conditioned on four noisefree observations (yellow). Additionally, the mean function (black) and the $\pm 2\sigma$-confidence region (grey) is shown.

where σ_y^2 is the covariance function of the independently added noise. In this case the model does not exactly interpolate the observed data points.

Covariance and mean functions determine some function properties such as smoothness. The mean function is often chosen to be zero as the GP model is flexible enough to still model the function arbitrarily well. A widely employed covariance function that results in a smooth function is the squared exponential kernel:

$$\kappa(x, x') = \sigma_f^2 \exp(-\frac{1}{2l^2}(x - x')^2). \tag{2.27}$$

This kernel reflects the intuition that "nearby inputs" should map to "nearby outputs".

An important question is how the kernel parameters l,σ_f and σ_n should be chosen. The horizontal length scale l determines the horizontal scale of the function, i.e. how "wiggly" the signal looks. The signal standard deviation σ_f is a vertical length scale and defines how much the signal deviates from the mean. The noise standard deviation σ_n describes how much noise is expected to be present in the data. If $\sigma_n = 0$ the data points are noiseless and will be fitted exactly.

In this section the estimation of the hyperparameters $\theta = \{l, \sigma_f, \sigma_n\}$ by maximizing the marginal likelihood will be explained. Generally, the goal is to find the set of hyperparameters that explains the obtained data points in the best possible way or more formally, maximises the probability $p(y|x, \theta)$. The corresponding log-likelihood function is given by

$$L(y|X, \theta) = \log p(y|x, \theta) = \log \mathcal{N}(0, \mathbf{K}_y) \tag{2.28}$$

$$= -\frac{1}{2} \log |\mathbf{K}_y| - \frac{1}{2} y^T \mathbf{K}_y^{-1} y - \frac{n}{2} \log(2\pi), \tag{2.29}$$

where N is the dimension of the data set. The first part of the formula corresponds to a complexity penalty term, whereas the second term is a data-fit measure. This property is of great importance as it implies that a trade-off between complexity and data-fit is automatically introduced and does not need to be tuned manually [9]. Maximising the likelihood function can be done based on the derivative

$$\frac{\partial L}{\partial \theta_i} = \frac{1}{2} \operatorname{trace}\left(\mathbf{K}_y^{-1} \frac{\partial \mathbf{K}_y}{\partial \theta_i}\right) + \frac{1}{2} y^T \mathbf{K}_y^{-1} \frac{\partial \mathbf{K}_y}{\partial \theta_i} \mathbf{K}_y^{-1} y. \tag{2.30}$$

Given this expression, the kernel parameters can be estimated with standard gradient-based optimizers. Since the objective is however not guaranteed to be convex, local minima can pose problems.

2.3 Safe Set Computations based on Reachability Analysis

This section aims at giving a brief explanation of a method known as Hamilton-Jacobi-Isaacs (HJI) reachability analysis that is employed in this thesis to guarantee

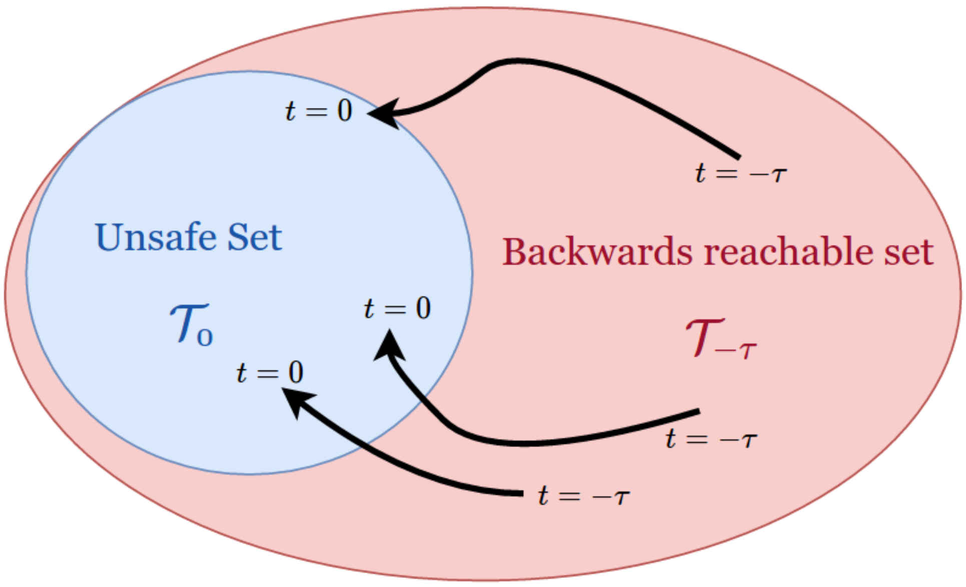


Figure 2.2: The backwards reachable set $\mathcal{T}_\tau$ is the set from which trajectories can reach the unsafe set within time τ.

safety within the learning framework. Both explanation and implementation are based on Ian M. Mitchell's Level Set Toolbox [10] and doctoral thesis [11].

Guaranteeing safety in our context means to ensure that a disturbed system will never leave a safe region $\mathcal{S}_0$. To this end, we assume that the disturbance d lies within a set $\mathcal{D}$ and a set $\mathcal{U}$ is given in which one can choose a control u. The question is now in which set $\mathcal{S}_\tau$ the initial conditions must lie such that there is a control strategy that makes the system stay in $\mathcal{S}_0$ within the time horizon $t \in [-\tau, 0]$.

As we require safety for all possible trajectories, it is not sufficient to simulate a few thousands or even millions trajectories, because there still is a risk to miss an unsafe trajectory. Instead, one can compute the backwards reachable set of the unsafe set $\mathcal{T}_0 = \mathbb{R}^n \backslash \mathcal{S}_0$ to capture all possible trajectories at once. To this end, we assume a control that tries to steer the system away from the unsafe set and an adversarial disturbance that tries to steer the system into the unsafe set. The backwards reachable set $\mathcal{T}_\tau$ is the set from which trajectories start that can reach the unsafe set within the time horizon τ and should therefore be avoided for all $t \in [-\tau, 0]$. Figure 2.2 illustrates this concept. Safety could therefore be guaranteed for time horizon τ, if we had a way to calculate the set $\mathcal{T}_\tau$ and a control strategy to stay outside that set.

The present section aims at showing a way as to how one can obtain both, the backwards reachable set and the safety-preserving control strategy, by solving the time-dependent HJI partial differential equation (PDE). The unsafe set $\mathcal{T}_0$ can be represented as the zero sublevel set of a bounded and Lipschitz continuous function $g : \mathbb{R}^n \to \mathbb{R}$, namely

$$\mathcal{T}_0 = \{x \in \mathbb{R}^n | g(x) \leq 0\}. \tag{2.31}$$

We can then calculate the backwards reachable set as the viscosity solution $\phi : \mathbb{R}^n \times [-T, 0] \to \mathbb{R}$ of the time dependent HJI PDE

$$\frac{\partial}{\partial t}\phi(x,t) + \min\left[0, H\left(x, \frac{\partial}{\partial x}\phi(x,t)\right)\right] = 0, \qquad \forall t \in [-T, 0], \forall x \in \mathbb{R}^n \tag{2.32}$$

$$\phi(x,0) = g(x), \qquad \forall x \in \mathbb{R}^n \tag{2.33}$$

where $H(x,p)$ is the Hamiltonian:

$$H(x,p) = \max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} p^T f(x,u,d). \tag{2.34}$$

The function $f(x,u,d)$ in this expression describes the system dynamics.

While the sublevel set of $\phi(x,0)$ corresponds to the unsafe set $\mathcal{T}_0$ as stated above, the sublevel set of $\phi(x,t)$ describes the backwards reachable set $\mathcal{T}_\tau$:

$$\mathcal{T}_\tau = \{x \in \mathbb{R}^n | \phi(x,t) \leq 0\}, \qquad \forall \tau \in [0, T]. \tag{2.35}$$

There are well-established numerical methods to accurately approximate $\phi(x,t)$ and thereby $\mathcal{T}_\tau$ even for non-linear dynamics. It can then be guaranteed that for states outside the backwards reachable set, there exists a control $u \in \mathcal{U}$ for all $d \in \mathcal{D}$ to keep the system outside the unsafe set for a time horizon τ. More specifically, it is sufficient to apply the optimizer u_{safe} of (2.34) given by

$$u_{\text{safe}} = \arg\max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} p^T f(x,u,d) \tag{2.36}$$

whenever the system hits the boundary of the safe set. Within the safe set $\mathcal{S}_\tau$, the control can be freely chosen e.g. by a learning controller. Furthermore, the backwards reachable set typically converges for a sufficiently large τ so that safety can then be guaranteed by staying within the control-invariant set $\mathcal{S}_\tau$ for all time.

Chapter 3

Related Work

The idea of improving the control of a system in an online manner is not a recent one. This section will describe some early approaches of “learning control”. There are two main strategies that researchers have been following. Firstly, learning methods can be employed to directly learn a control for a given system. Secondly, data can be used to improve the model of the system online.

An early example of the first approach is Iterative Learning Control [12]. In Iterative Learning Control, the controller incrementally improves its performance of a repetitive task. The control action is modified based on the control action of previous iterations and the deviation from a desired trajectory. However, this technique is constrained to a predefined repetitive reference trajectory with fixed durations of iterations and fixed initial conditions for each iteration.

A different example for an early contribution to this matter is the article of Wang et. al. [13] on stable adaptive fuzzy control. The proposed method assumes that the state space is of a special form

$$x^{(n)} = f(x, \dot{x}, ..., x^{(n-1)}) + bu, \qquad y = x \tag{3.1}$$

with an unknown continuous function f and an unknown constant b. A way is then presented on how to update the singleton parameters θ of a fuzzy controller

$$u_c(x) = \theta^T \varepsilon(x) \tag{3.2}$$

with fuzzy basis functions $\varepsilon(x)$. Assuming boundaries on b and f, a supervisory controller that ensures stability is further introduced. The approach can only be applied to the above explained restricted class of state space models but is not limited to one trajectory. By incorporating the supervisory controller, it realizes some early form of “safe learning for control”.

The approach of learning the system model instead of a direct control has for instance been pursued in the field of Indirect Adaptive Control [12]. The adaptation of control parameters is here done by firstly estimating parameters of the model and subsequently computing control parameters based on that model. However, the estimation of parameters implicates that only a certain class of functions is considered. If the target function is not well modelled within that class, the control performance might still be poor. For this reason, Gaussian processes that are flexible enough to model a variety of different functions have been proven to be a valuable tool for non-linear modelling [8].

An approach that combines the indirect learning of system dynamics with the direct learning of control has been proposed in [1]. This thesis is largely based on the method proposed in that article. The approach assumes an unknown, additive, state-dependent disturbance $d(x)$. The state space is assumed to be of the form:

$$\dot{x} = h(x) + g(x)u + d(x), \tag{3.3}$$

where $h(x)$ and $g(x)$ are the known parts of the system dynamics. Assuming upper bounds on the disturbance, a safe set can be calculated using Hamilton-Jacobi-Isaacs (HJI) reachability analysis. Starting from a given conservative upper bound, the disturbance estimate is iteratively improved, leading to an increasing safe set. For states inside the safe set, a control is found with the learning algorithm Policy Gradient via the Signed Derivative (PGSD). Otherwise, a safe control that attempts to drive the system to the safest possible state is applied. In the objective function of the PGSD, a term is included that punishes switching between the safe and the learning control. This is done in order to reduce the number of times that the system reaches the border of the safe set. An issue with this approach is that exploration is not handled explicitly. This means that the algorithm does not encourage the acquisition of new knowledge about the state space. On the contrary, it discourages the system from taking exploratory actions towards the borders of the safe set by punishing switching between the safe and learning control. This conflicts with the goal of enlarging the safe set as the disturbance estimate will remain uncertain in regions where few samples have been gathered.

Because the approaches presented in [1] and [13] both realise some form of safe learning, it might be interesting to compare the fundamentally different techniques they employ. Compared to [13], the method presented in [1] poses less restrictions on the form of the state space. However, the way safety is ensured in this approach is a lot more complex than the computation of the supervisory control in [13]. The estimation of the disturbance and subsequent computation of a safe set is theoretically complex and computationally expensive. Both approaches have some critical assumptions: While the method in [13] relies on knowledge of the bounds of f and b, that of [1] assumes the knowledge of upper and lower bounds of d. Both assumptions might be hard to satisfy in practice. Finally, in the approach presented in [1], a control is learned via an learning algorithms that can be designed for performance. In [13], the update of the parameter vector θ is fixed.

Chapter 4

Solution Architecture

The thesis focuses on implementing a safe learning controller for an inverted pendulum. Initially, a Markov Decision Process model of the pendulum system is obtained by essentially discretizing the state and action space, assigning rewards to the discrete states, and calculating transition probabilities between the states. As in [1], a system with an unknown additive state-dependent disturbance $d(x)$ is assumed. Additionally, conservative initial disturbance bounds, the upper bound $\overline{d}_0$ and the lower bound $\underline{d}_0$, are assumed to be known initially. The bounds will be iteratively updated with a GP model. Based on the initial disturbance estimate and a safe set, the backwards reachable set, which should be avoided in order to never leave the safe set, is calculated with HJI reachability analysis. This calculation gives rise to a safe region within the state space within which the learning controller can operate freely. Additionally, the safe set calculations output a safe controller that should be applied at the borders of this set. Based on that, the chosen reinforcement learning algorithm can learn a policy by choosing actions and receiving subsequently information about the reward and the state transition associated with that action. If the chosen action would cause the system to leave the safe region, the safe controller acts and brings the system back into the safe set. The chosen reinforcement learning algorithm is a modified version of Delayed Q-Learning introduced in Section 2.1.

While the learning controller acts, data samples are recorded and subsequently fed into the GP model. The GP estimates a less conservative bound for the disturbance so that subsequently a larger safe set can be calculated with HJI reachability analysis. This procedure is sketched in Figure 4.1. The estimated safe set is fed into the safe learning controller that learns a policy through interaction with the system. The samples recorded during learning are used to get a better estimate of the disturbance thus resulting in a less restrictive safe control.

More precisely, the same control scheme is explained in Figure 4.2. In this figure the colours blue and red are used to underline the two control loops that run in parallel: The blue loop is the safety loop, which estimates the disturbance, calculates a safe set (and a safe controller on that basis), and checks every action that the

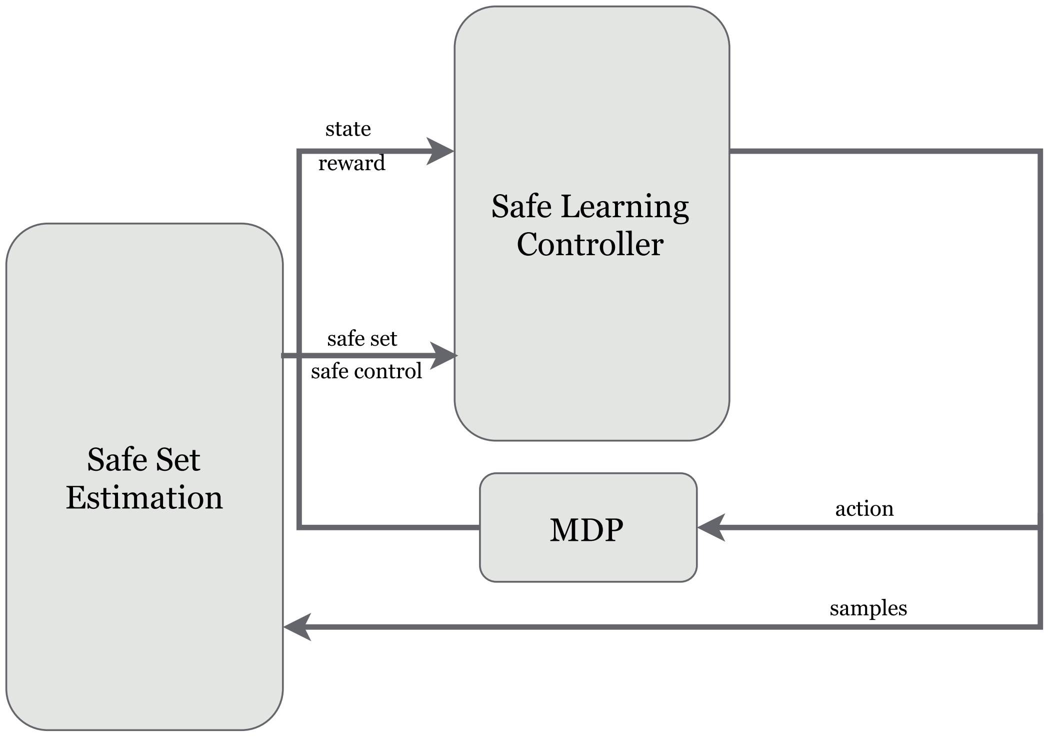


Figure 4.1: Rough Control Setup.

learning controller wants to take. The red loop is the learning control loop where the reinforcement learning controller chooses an action based on its policy, and subsequently receives some feedback from the process. Based on the received reward, the controller updates its policy. For each action chosen by the learning controller, the safe controller performs a check if that action would violate the boundaries of the safe set. If that is the case, the action is not executed but replaced by a safety-preserving action. This very rough sketch will be further explained in Section 5.

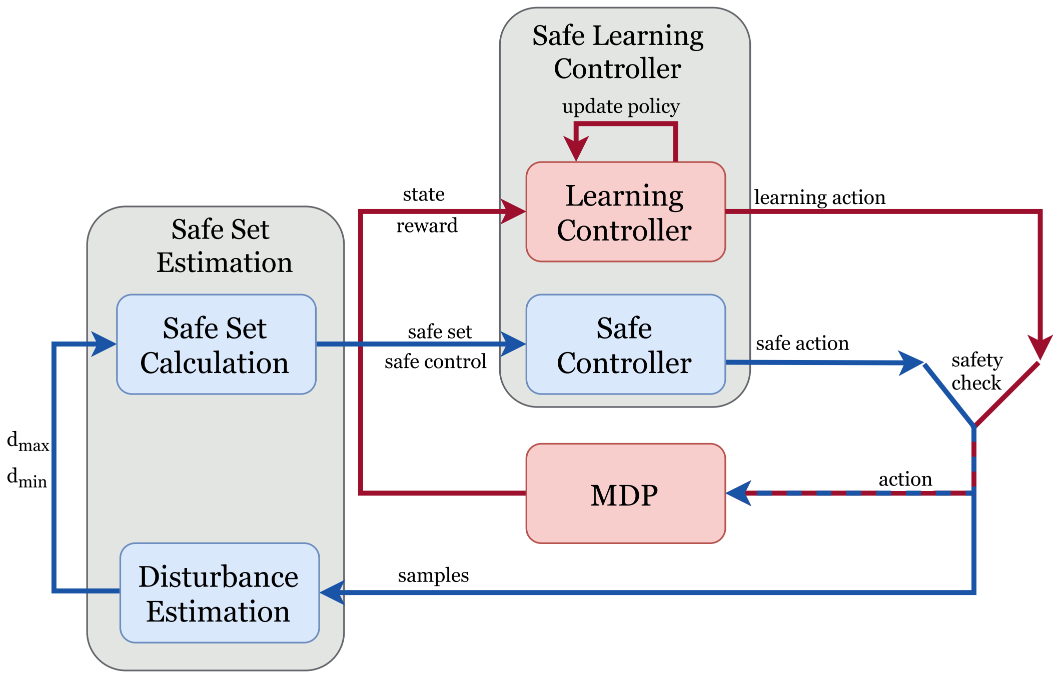


Figure 4.2: Detailed Control Setup.

Chapter 5

Implementation

In this chapter, the implementation details of the approach outlined in Section 4 will be discussed. Starting from the modelling of the system as a Markov Decision Process, the implementation of the reinforcement learning algorithm, the disturbance estimation with Gaussian processes, and finally the safe set calculation with HJI reachability analysis will be described.

5.1 Markov Decision Process Model

Consider a damped inverted pendulum system with mass m, length l and friction coefficient b. The states of the system are the pendulum angle x_1 and angular velocity x_2. The system is disturbed by an additive state-dependent disturbance $d(x)$. The dynamics are described by [14]:

$$\dot{x_1} = x_2 \tag{5.1}$$

$$\dot{x_2} = \frac{1}{ml^2}u + \frac{g}{l}\sin(x_1) - \frac{b}{m}x_2 + d(x). \tag{5.2}$$

All constants are assumed to be positive. Modelling the inverted pendulum system as a MDP means to define the quintuple (S, A, T, r, γ) as given in Section 2.1. Defining the finite set of states S and the finite set of actions A implies discretisation over the intervals $[x_{\min}, x_{\max}]$ and $[u_{\min}, u_{\max}]$, where x_1 is a circular state and should be wrapped if the discretisation interval is larger than its period 2π. The number of discretisation steps impacts the convergence speed of the reinforcement learning algorithm and was chosen around $n = 20$ for each state dimension. The transition probabilities T have been approximated by simply sampling the discrete transitions under a chosen time step h. Given a specific action and state, the subsequent state has been simulated 100 times. The probability $p(s'|s, a)$ of state s' is then computed as the number of transitions to s' from the chosen state and action divided by the number of total transitions. It is hereby important that the time step h is large enough to actually allow transitions. If h is too small, the system will always stay in state s regardless of the chosen action. For the present system, a time step around $h = 0.2\,\mathrm{s}$ has proven to give good results. The reward function for state s, $r(s)$, is defined as

$$r(s) = e^{-\frac{\|s\|^2}{\sigma^2}}, \tag{5.3}$$

where σ is a constant defining how narrow the function is. Defining the coordinate system with $x_1 = 0$ being on the top, this reward corresponds to the goal of keeping the pendulum still and upright as the states with the smallest x are rewarded the most. The discount factor γ is chosen to be 0.9. This choice weighs future rewards slightly less than present rewards. One reason for not weighing all rewards the same is that the convergence time increases if the discount factor is chosen larger. Furthermore, the pendulum is being reset as soon as the state values are outside some margin around the border of the MDP. This is done as part of the algorithm, because learning can only take place inside the MDP. Therefore, a reward far in the future might never be obtained as the pendulum might be reset earlier and should be weighed less.

5.2 Reinforcement Learning

To our learning agent (the inverted pendulum controller), the reward and transition probabilities are not known beforehand. The agent acts in an unknown environment and aims at maximising an external reward. We therefore apply reinforcement learning to learn the optimal policy.

The chosen algorithm resembles largely the Delayed Q-Learning algorithm introduced in Section 2.1 with the update rule (2.16). The only difference lies in the handling of the algorithm inputs ε and m. The batch size m determines how often the update of a state-action pair's Q-value is attempted until a successful update of any Q-value has to occur for the update to be attempted again. This batch size has been replaced by the state-action-dependent batch size $M(s, a)$ that is increased each time when the update of the state-action pair (s, a) has been tried $M(s, a)$ times. The update rule is

$$M_{t+1}(s_t, a_t) = \min\{\lceil 1.02 M_t(s_t, a_t)\rceil + 1, 500\} \tag{5.4}$$
$$M_{t+1}(s, a) = M_t(s, a) \qquad \forall (s, a) \neq (s_t, a_t) \tag{5.5}$$

The threshold ε for an update to be admitted has been made adaptive such that ε after $M(s, a)$ attempted updates increases to

$$\varepsilon_{t+1} = \min\{1.1\varepsilon_t, \varepsilon_{\text{target}}\}, \tag{5.6}$$

where $\varepsilon_{\text{target}}$ is an input parameter. The initialization of the Q-values is done optimistically according to (2.14). The learning rate is chosen to be $\alpha = \frac{1}{v+1}$ where v is the number of times that the current state action pair has been visited. This learning rate leads to a faster convergence than $\alpha = \frac{1}{n+1}$ with iteration count n and fulfills still the convergence criteria posed in (2.12). Without any safety-preserving

Algorithm 1 MODIFICATION OF DELAYED Q-LEARNING

```
Require: $\mathcal{S}, \mathcal{A}, \gamma$, and $R_{\max}$, $m_0$, $\varepsilon_{\text{target}}$.
  for all $(s, a)$ do
    $Q(s,a) \leftarrow R_{\max}/(1-\gamma)$   // optimistic initialization of $Q$-values
    $U(s,a) \leftarrow 0$                     // per-batch cumulative $Q$-values for $(s,a)$
    $B(s,a) \leftarrow 0$                     // beginning time-step of attempted update for $(s,a)$
    $C(s,a) \leftarrow 0$                     // per-batch counter for $(s,a)$
    $M(s,a) \leftarrow m_0$                   // batch size for $(s,a)$
    $L(s,a) \leftarrow$ TRUE                  // the learning flag
  end for
  $t_{\text{update}} \leftarrow 0$            // time-step of the most recent $Q$-value change
  $\varepsilon \leftarrow 10^{-4}$            // threshold for admitting $Q$-value updates
  for $t \geq 1$ do
    Observe the current state $s_t$. Take action $a_t \in \text{argmax}_{a \in \mathcal{A}} Q(s_t, a)$, receive reward $r_t$,
    and go to a next state $s_{t+1}$.
    if $B(s_t, a_t) \leq t_{\text{update}}$ then
      $L(s_t, a_t) \leftarrow$ TRUE
    end if
    if $L(s_t, a_t)$=TRUE then
      if $C(s_t, a_t) = 0$ then
        $B(s_t, a_t) \leftarrow t$
      end if
      $C(s_t, a_t) \leftarrow C(s_t, a_t) + 1$
      $U(s_t, a_t) \leftarrow U(s_t, a_t) + r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a)$
      if $C(s_t, a_t) = M(s_t, a_t)$ then
        $q \leftarrow U(s_t, a_t)/M(s_t, a_t)$
        if $|Q(s_t, a_t) - q| \geq \varepsilon$ then
          $Q(s_t, a_t) \leftarrow q$   // update if $Q$-value changes significantly
          $t_{\text{update}} \leftarrow t$
        else if $B(s_t, a_t) > t_{\text{update}}$ then
          $L(s_t, a_t) \leftarrow$ FALSE
        end if
        $U(s_t, a_t) \leftarrow 0$
        $C(s_t, a_t) \leftarrow 0$
        $M(s_t, a_t) \leftarrow \min\{\lceil 1.02 M(s_t, a_t) \rceil + 1, 500\}$ //increase batch size
        $\varepsilon \leftarrow \min\{1.1\varepsilon, \varepsilon_{\text{target}}\}$ // increase the threshold
      end if
    end if
  end for
```

controller, this algorithm converges to the optimal policy but does not guarantee any constraint satisfaction. Figure 5.1 shows the learned policy after a test run with 100, 000 steps in comparison with the optimal policy that has been found with policy iteration. The colours correspond to certain action values. As the actual values are not of importance, no legend matching colours to values has been provided. It can be seen that the estimated policy is relatively accurate, however the learning algorithm violates the boundaries of the MDP no less than 8986 times. As the algorithm learns the optimal policy, the number of constraint violations is decreasing. This can be seen in Figure 5.2, where a histogram of the constraint violations is shown. However, safety can never be guaranteed in this framework. To prevent occurrence of these violations, a safe controller will be introduced in the next section.

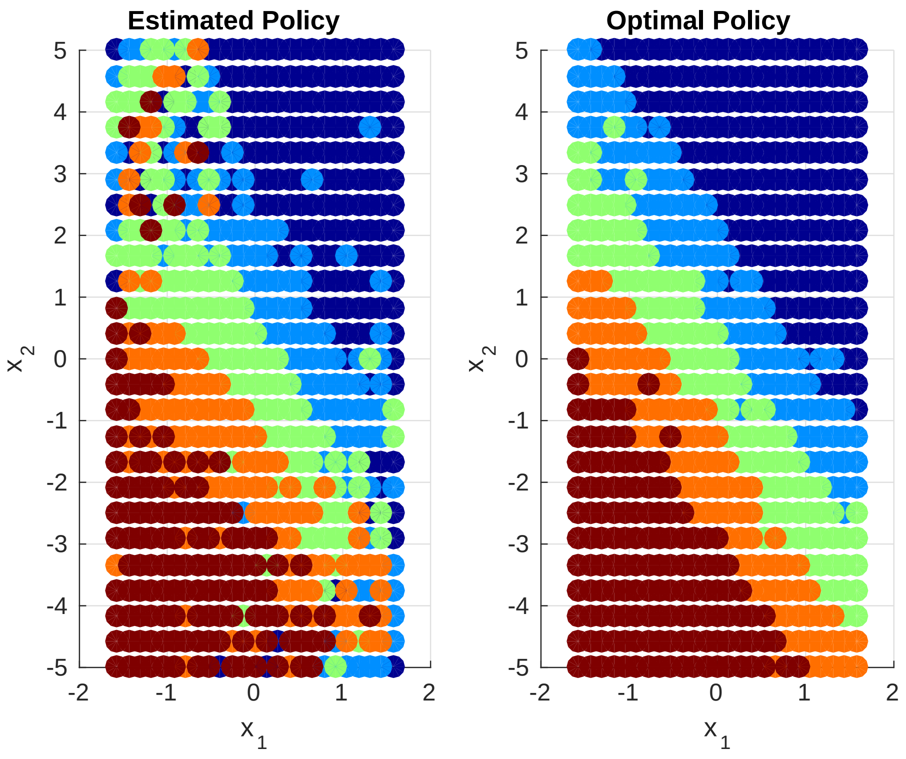


Figure 5.1: The learned policy versus the optimal policy.

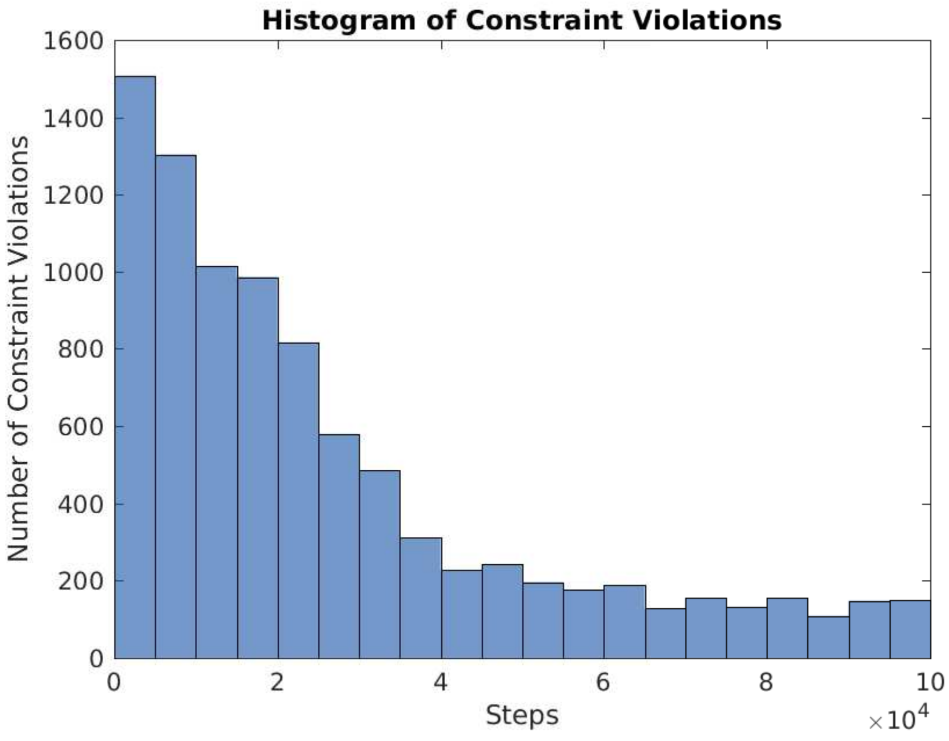


Figure 5.2: Evolution of the constraint violations with increasing number of steps.

5.3 Safe Set Computations based on Reachability Analysis

The general idea behind safe set computations has been described in Section 2.3. In this section the concrete implementation of such computations for the case of the inverted pendulum system will be described. The Hamiltonian in this case becomes

$$H(x,p) = \max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} p^T \begin{pmatrix} x_2 \\ \frac{1}{ml^2}u + \frac{g}{l}\sin(x_1) - \frac{b}{m}x_2 + d \end{pmatrix}. \tag{5.7}$$

Determining the optimizers to (5.7) can be done easily for non-linear systems whose inputs enter linearly. That means that the dynamics can be written on the form

$$f(x,u,d) = f^x(x) + \mathbf{F}^u(x)u + \mathbf{F}^d(x)d. \tag{5.8}$$

This is the case for the inverted pendulum system with

$$f^x(x) = \begin{pmatrix} x_2 \\ \frac{g}{l}\sin(x_1) - \frac{b}{m}x_2 \end{pmatrix}, \tag{5.9}$$

$$\mathbf{F}^u(x) = \begin{pmatrix} 0 \\ \frac{1}{ml^2} \end{pmatrix}, \tag{5.10}$$

$$\mathbf{F}^d(x) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{5.11}$$

According to [10], the input maximising (5.7) and the disturbance minimizing the equation are then given by

$$u^*(x,p) = \begin{cases} \underline{U}, & \text{if } \sum_{j=1}^{n} p_j \mathbf{F}_j^u(x) \leq 0; \\ \overline{U}, & \text{otherwise,} \end{cases} \tag{5.12}$$

$$d^*(x,p) = \begin{cases} \overline{D}, & \text{if } \sum_{j=1}^{n} p_j \mathbf{F}_j^d(x) \leq 0; \\ \underline{D}, & \text{otherwise.} \end{cases} \tag{5.13}$$

$\overline{D}$ and $\underline{D}$ describe the input constraints $u \in \mathcal{U} = [\underline{U}, \overline{U}]$ and disturbance bounds $d \in \mathcal{D} = [\underline{D}, \overline{D}]$.

Hence, the Hamiltonian can be written as

$$H(x,p) = p_1 x_2 + p_2 \left(\frac{g}{l}\sin(x_1) - \frac{b}{m}x_2 + \frac{1}{ml^2}u^* + d^* \right). \tag{5.14}$$

$H(x,p)$ as given above must be coded into an existing function prototype within the toolbox. To calculate the derivative $p = \frac{\partial}{\partial x}\phi(x,t)$, the Level Set toolbox employs the Lax-Friedrichs approximation

$$\widehat{H}(x, p^+, p^-) \triangleq H\left(x, \frac{p^- + p^+}{2}\right) - \frac{1}{2}\alpha^T(x)(p^+ - p^-) \tag{5.15}$$

where p^+ and p^- are respectively left and right side approximations of p, and where $H(x,p)$ is given in (5.14). The function $\alpha(x)$ must also be implemented within the provided function prototype. $\alpha(x)$ is given to be

$$\alpha_i(x) = \max_{p \in \mathcal{I}} \left| \frac{\partial H}{\partial p_i} \right|, \tag{5.16}$$

with $\mathcal{I}$ being the hypercube containing all values that p takes over the computational domain. The calculation can be done with the following over-approximation

$$\alpha_j(x) \leq |f_j^x(x)| + |\mathbf{F}_j^u| U_{\max} + |\mathbf{F}_j^d| D_{\max}, \tag{5.17}$$

which for the present system reduces to

$$\alpha_1 \leq |x_2|, \tag{5.18}$$

$$\alpha_2 \leq \left| \frac{g}{l}\sin(x_1) - \frac{b}{m}x_2 \right| + \frac{1}{ml^2}U_{\max} + D_{\max}. \tag{5.19}$$

Having specified the functions for calculating $H(x,p)$ and $\alpha(x)$, the Level Set toolbox can be adapted for the present problem. Specifically, a function has been written that takes as inputs an estimate for the disturbance bounds $\underline{D}$ and $\overline{D}$, input constraints $\underline{U}$ and $\overline{U}$, an initial safe set $\mathcal{S}_0$, a time horizon τ, and the state space of the MDP. For each state in the MDP, whether it is safe under time horizon τ and a safe control $u^*(s)$. During the learning process, one can then apply the safe control as soon as the system hits the border of the safe set. To illustrate this, Figure 5.3 shows the evolution of a safe set calculation over the time horizon $\tau = 30\,\mathrm{s}$. The first subplot shows the initial safe set $\mathcal{S}_0$ as defined by the function input. Over time, the safe set is shrinking to the set $\mathcal{S}_\tau$ shown in the last subplot. Trajectories starting from within this set are guaranteed to remain in $\mathcal{S}_0$ for $t = [0, \tau]$. It can easily be seen that the set converges already within the first seconds such that trajectories that are safe for $t = [0, 10]$ are also safe for the whole time horizon.

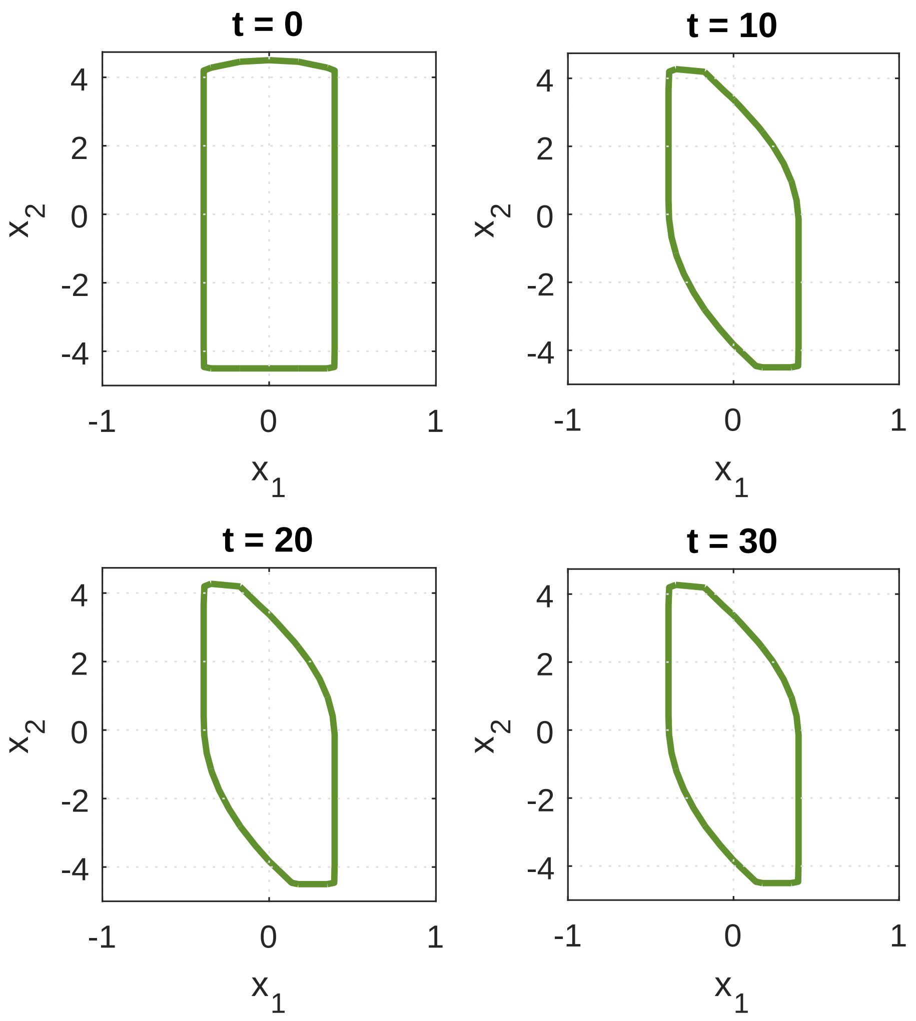


Figure 5.3: Example for the evolution of a safe set during the time horizon.

To verify the calculations, simulations of system trajectories have been done for both, initial conditions within the safe set $\mathcal{S}_\tau$ and outside that set. The results are depicted in Figure 5.4. Trajectories are marked with a red dot at the initial condition and a blue dot at the final value. The safety-preserving controller manages to keep all trajectories with initial conditions inside $\mathcal{S}_\tau$ within the safe set for the whole simulation time. The simulation time has been chosen to be smaller than the time horizon $\tau = 30\,\mathrm{s}$ to keep the computation time low and the figure clean. The result holds however for longer simulation times. Furthermore, the figure indicates that the safe set is an under-approximation of the true safe set as the trajectories with initial conditions close to $\mathcal{S}_\tau$ can be stabilized too. This is expected as the calculated reachable set is an over-approximation of the true reachable set.

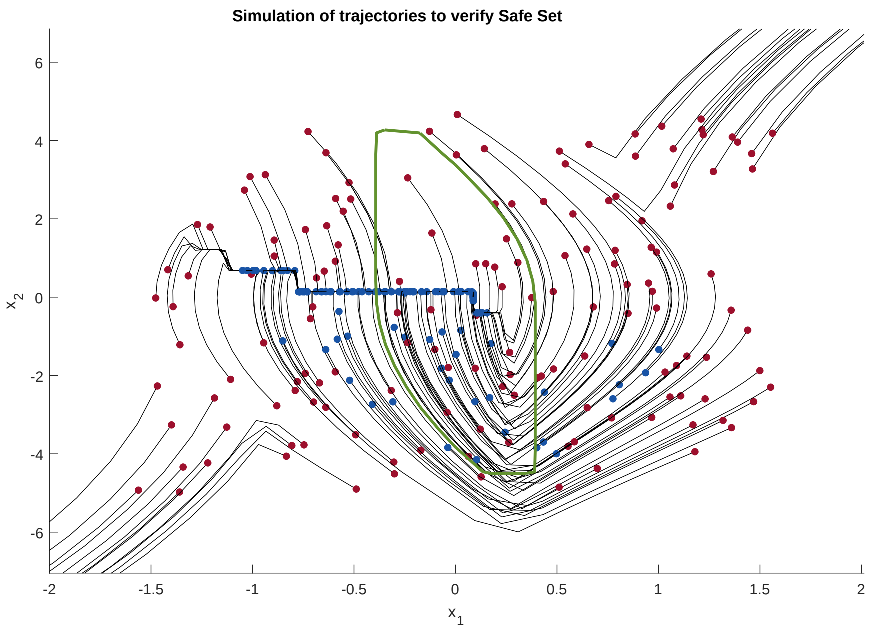


Figure 5.4: Simulations to verify the control invariance of the safe set.

It is worth mentioning that the safe set calculations assume continuous dynamics. It is therefore important to keep the time step h small enough so that the error through discretisation does not become too large. This conflicts with the requirement of Section 5.1 which necessitates the time step to be large in order to allow transitions between states. With a very small time step learning can not take place as no state transitions can be made. On the other hand, a large time step jeopardises safety as the system possibly violates the border of the safe set between two samples and it is then not guaranteed that the safe control can bring the system back into the borders again. Optimally, one would find a structured way to shrink the safe set dependent on the time step in order to account for the error caused by discretisation. This falls beyond the scope of this thesis. This problem is tackled otherwise by introducing a "safety loop" that runs faster than the actual learning loop. For instance, if the time step required from the MDP is $h_{\text{learn}} = 0.12\,\text{s}$, but the time step for the safety-preserving controller should be maximal $h_{\text{safe}} = 0.02\,\text{s}$, the safety loop will run six times faster than the learning loop. In each safety iteration the evolution of the system is simulated. If the system violates the boundaries of the safe set, the safe control is applied. Otherwise the learning control (that is constant over the six safety iterations) is applied. This is portrayed in Figure 5.5. During the fourth iteration, the safe controller detects a violation of the safe set boundaries and applies the safe control that brings the system back inside the boundaries. For the sake of illustration, the time steps are chosen very large in comparison to the drawn safe set. Obviously, one would not choose the time step h_{learn} so large that the whole safe set can be crossed within one time step.

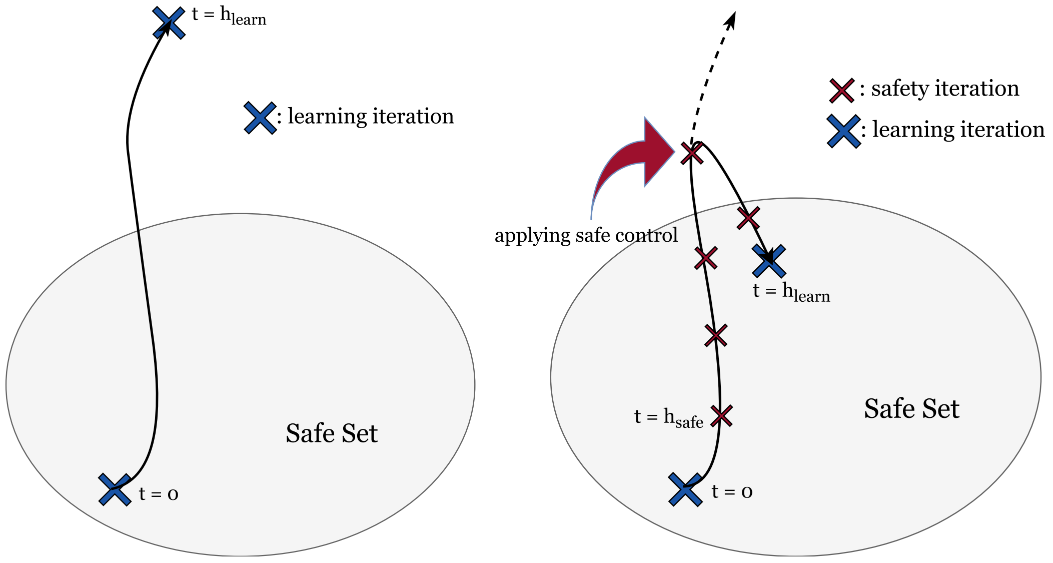


Figure 5.5: Illustration of the problem with a large time step h_{learn}. At $t = h_{\text{learn}}$ the system is without safety check (left) already far outside the safe set so that it cannot be guaranteed to be brought back again. On the right hand side, the same example is shown with a faster running safety loop. Each h_{safe} a safety check is performed so that the violation of the safe set can be detected and acted against earlier.

5.4 Disturbance Estimation with Gaussian processes

In this section, the disturbance estimation with GP regression will be described. As stated above, the safe set calculations with HJI reachability analysis rely on a given bound for the disturbance. This bound is initially picked in a conservative manner but should be updated as soon as data from the real system is available. Given some measurements of the state x collected during the learning process, disturbance samples can be obtained by calculating:

$$\hat{d}(x) = \dot{x}_2 - \left(\frac{1}{ml^2} u + \frac{g}{l} \sin(x_1) - \frac{b}{m} x_2 \right). \tag{5.20}$$

As the continuous derivative $\dot{x}_2$ is not available, an approximation with the forward difference is calculated:

$$\dot{x}_2 \approx \frac{x_{2,k+1} - x_{2,k}}{h}, \tag{5.21}$$

where $x_{2,k+1}$ and $x_{2,k}$ are two consecutive samples and h is the sample time. For this approximation to be accurate, it is important that the time step h is small.

Therefore, the samples are recorded within the faster running safety loop, so that h from the above equation is given as $h = h_{\text{safe}}$. In order to obtain estimates for d for all values of x and not only for the sampled values, a GP model is employed. GP also have the advantage to provide not only an estimate, but with the standard deviation also an uncertainty measure, so that we can pick the bounds of $d(x)$ to be

$$\underline{d}(x) = \mu(x) - 3\sigma(x) \tag{5.22}$$

$$\overline{d}(x) = \mu(x) + 3\sigma(x). \tag{5.23}$$

Here, $\mu(x)$ is the mean of the estimate and $\sigma(x)$ is the standard deviation. The chosen probabilistic bounds for $\overline{d}$ and $\underline{d}$ give a confidence interval of 99.7%. However, one should consider that the numerical differentiation introduces an error that is not accounted for in this confidence interval. This did not pose any problem in the algorithm, because h_{safe} has been picked sufficiently small to ensure safety. Nevertheless, it might be interesting to account for the error due to numerical differentiation and discrete state space in a more structured way when calculating the safe set.

Another important question is how to choose sample points that can be fed to the GP regression. As the regression is computationally expensive, it is not possible to feed all recorded samples into the GP. One should rather choose around 1000 points that accurately reflect the whole state space. The question then is how to ensure that the sample points are evenly spread over the whole state space, so that a good estimate over the whole state space can be obtained. To achieve this, it is not enough to randomly pick samples because the samples will concentrate around the equilibrium point as soon as a good control is learned. Instead, a set of randomly chosen points that cover the whole state space is used as a grid. The samples from the system that lie closest to those points are chosen so that it can be ensured that the points around the equilibrium are not over-represented. This procedure is illustrated in Figure 5.6. One should notice that this is a rather complicated and artificial way of achieving a good spread of the samples. Instead of cherry picking the samples, one should rather encourage the system to actually explore the edges of the safe set. This issue has been dealt with in Section 5.5. Furthermore, all chosen samples still will lie within the safe set, as the system is required to never leave this set. However, by getting a less conservative disturbance estimate at the borders of the safe set, it is possible to subsequently increase the safe set and therefore the region where samples can be collected.

We then have a set of evenly spread disturbance samples that can be used for GP regression. The regression is done with the GPML toolbox [15] with a zero mean function and a squared exponential kernel as given in (2.27).

Figure 5.7 aims at illustrating the results from a GP regression for a constant disturbance $d = 2\,\text{N}$ after $48,000$ recorded samples of which 1000 are fed into the GP. The figure shows 1000 input samples (blue) that are widely spread within the

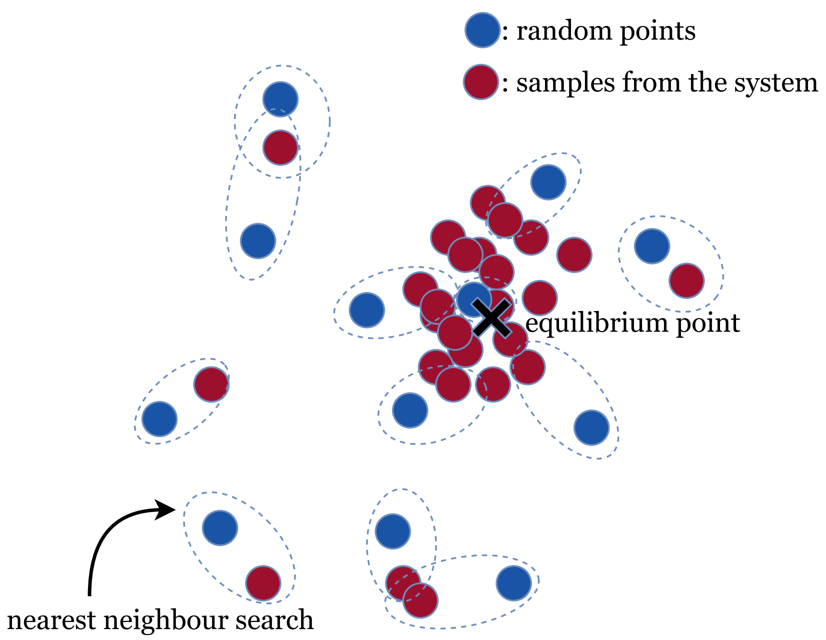
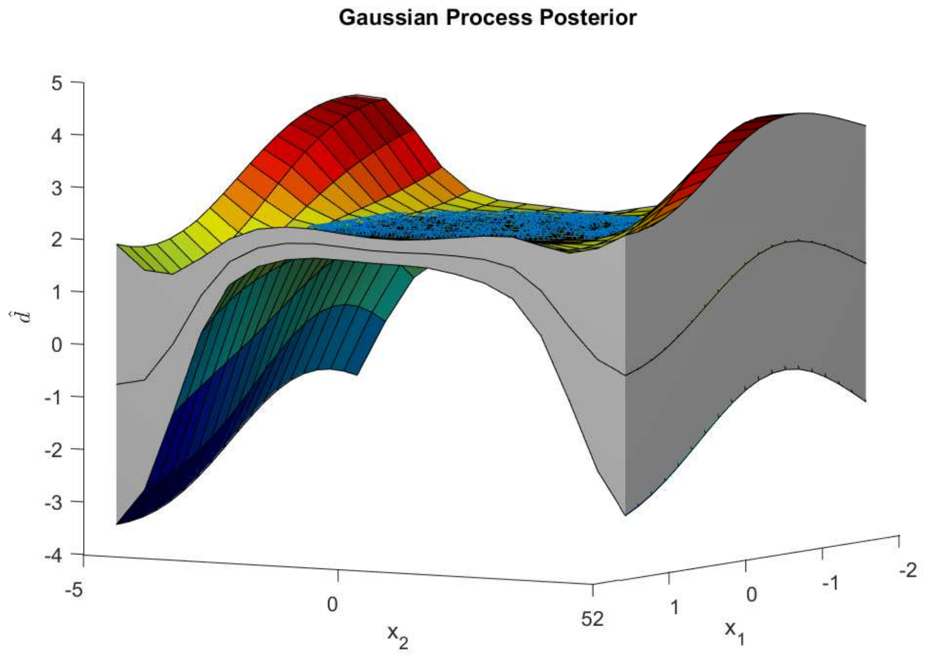


Figure 5.6: Choice of samples (red) by employing a grid (blue) and taking the nearest neighbours to the grid points in order to ensure that the samples cover the whole state space.

safe set. The upper and lower planes bound the $\pm 3\sigma$ confidence interval that is additionally shaded in grey. The plane "sandwiched" in the middle between the two outer planes is the mean, i.e. the disturbance estimate that the GP outputs. In the sliced lower plot, it can be seen that the estimate in the region around the origin is very accurate and the uncertainty is low.

5.5 Exploration

While this thesis mainly follows the approach presented in [1], this section deals with a topic that has not been treated in that approach: Exploration. Exploration describes the need of visiting the whole state space in order to find the optimal strategy instead of only exploiting readily observed parts of the state space. In the inverted pendulum system, this describes for instance the need for visiting the borders of the safe set in order to get a better disturbance estimate and potentially enlarge the safe set instead of only staying close to the origin. Furthermore, the learning of a control policy for all states requires that all states are visited a sufficient number of times. In Section 5.4, the need for exploration has been dealt with in a rather unorthodox way. By setting up a grid and doing a nearest neighbour search, it has been ensured that the samples cover the whole state space. However, this is not exploration, as one does not encourage the system to visit the borders of the safe set, but rather hopes for that the system does so automatically. For this reason,



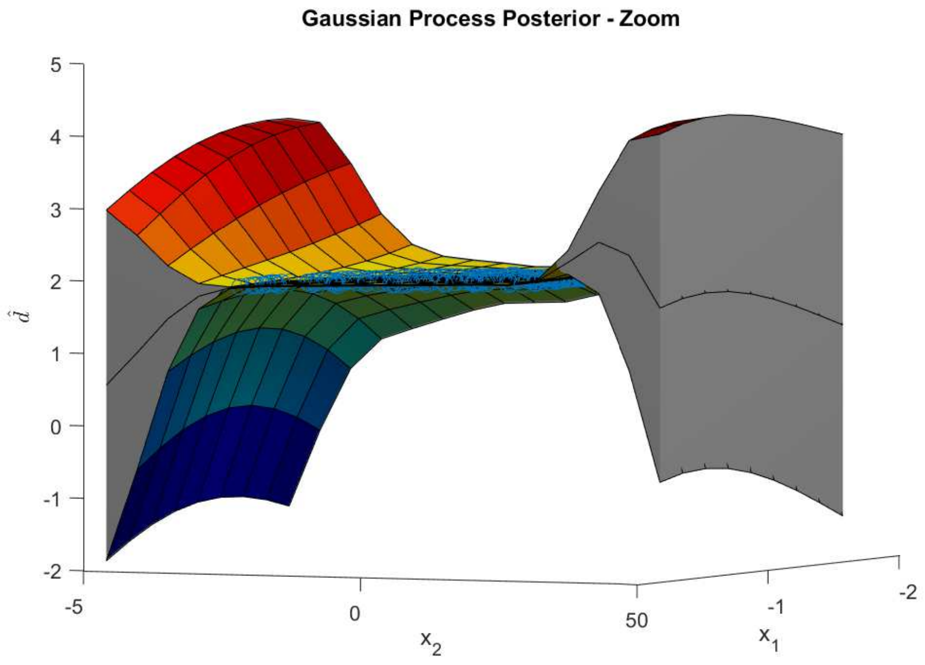


Figure 5.7: Disturbance estimation with GP regression for a disturbance. The lower and upper planes bound the $\pm 3\sigma$ confidence interval (shaded in grey). The samples that serve as input to the GP are shown as blue circles. The mean plane can be seen as a thin line in the middle of the confidence interval. The lower figure depicts a slice through the middle of the figure to show the uncertainty in the middle of the surface.

a more essential way of doing exploration will be presented in this section. The approach chosen is called Incremental Q-learning and aims at a de-randomisation of the ϵ-greedy policy as explained in Section 2.1 [16]. Instead of choosing random actions with probability ϵ, a greedy policy with respect to the Q-values plus a promotion term $A(s,a)$ is chosen. $A(s,a)$ is initialized to zero and updated in the following way:

$$A_{t+1}(s,a) = \begin{cases} 0, & \text{if } s_t = s, a_t = a; \\ A_t(s,a) + \Phi_t(\#(s,a)), & \text{if } s_t = s, a_t \neq a; \\ A_t(s,a), & \text{if } s_t \neq s, a_t \neq a. \end{cases} \tag{5.24}$$

The term $\Phi_t(\#(s,a))$ is the promotion function that depends on the number of times the system visited state s without executing action a since the last time it executed a from s. The promotion function chosen in the following is $\Phi_t(i) = \frac{1}{i+1}$. This setup promotes state-action pairs that have not been visited often and therefore it encourages exploration.

Chapter 6

Experimental Results

6.1 Experimental Setup

In this chapter, we numerically examine the approach described in Chapter 4. The inverted pendulum system has been modelled as a MDP with $x_1 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and $x_2 \in [-5\,\frac{\text{rad}}{s}, 5\,\frac{\text{rad}}{s}]$. The state space has been discretised with 19 steps in each dimension and the action space has been discretised with 5 steps on the range $u \in [-25\,\text{N}, 25\,\text{N}]$. All MDP parameters can also be found in table 6.1.

	min	max	# steps
x_1	-$\frac{\pi}{2}$	$\frac{\pi}{2}$	19
x_2	$-5\,\frac{\text{rad}}{s}$	$5\,\frac{\text{rad}}{s}$	19
u	$-25\,\text{N}$	$25\,\text{N}$	5

Table 6.1: MDP parameters.

The time step for the learning loop is chosen as $h_{\text{learn}} = 0.2\,\text{s}$. The design parameters for the learning algorithm can be looked up in table 6.2.

steps per iteration	10,000
h_{learn}	0.2 s
γ	0.9
m_0	5
$\varepsilon_{\text{target}}$	0.01
ε_0	0.0001

Table 6.2: Learning parameters.

The true disturbance introduced to the system is chosen to be $d = 0.5\sin(5x_1)$ and initially bounded conservatively with $\overline{d}_0 = 10\,\mathrm{N}$ and $\underline{d}_0 = -10\,\mathrm{N}$. The safe region $\mathcal{S}_0$ is a rectangle centered around the origin with width 0.9π in x_1-direction and width $10\frac{\mathrm{rad}}{s}$ in x_2-direction. In each iteration the safe set is calculated for $\tau = 40\,\mathrm{s}$. The safe set parameters are collected in table 6.3.

	min	max
$x_{1,\mathrm{safe}}$	-0.45π	0.45π
$x_{2,\mathrm{safe}}$	$-5\,\frac{\mathrm{rad}}{s}$	$5\,\frac{\mathrm{rad}}{s}$
d_0	$-10\,\mathrm{N}$	$10\,\mathrm{N}$
h_{safe}	$0.005\,\mathrm{s}$	
τ	$40\,\mathrm{s}$	

Table 6.3: Safe Set parameters.

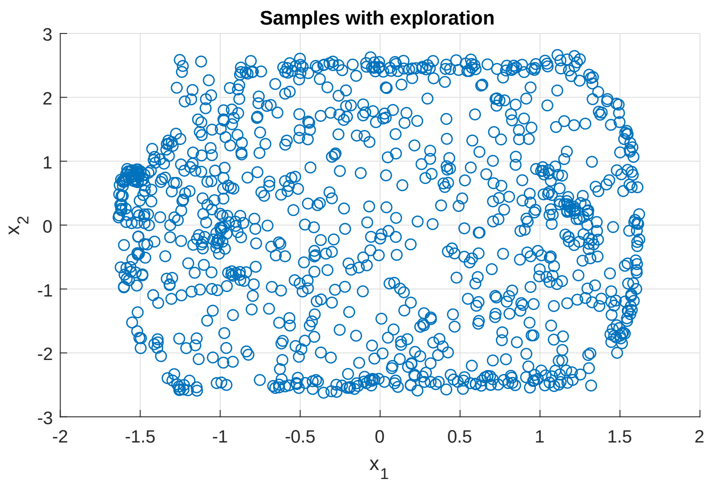
In the following part, the results from four learning iterations with each 10,000 steps will be presented. After each iteration, a new disturbance estimation and safe set calculation is done. As described in Section 5.5, it is crucial for the disturbance and policy estimation that all states within the safe set are visited sufficiently often. A method for structured exploration, Incremental Q-learning, has been introduced. In this chapter, the results from incorporating exploration will be compared to those obtained from the approach without structured exploration. The four iterations take around 80 s to terminate. The exact times for each iteration can be found in 6.4. The remainder of the elapsed time is due to the calculation of the MDP and result plotting.

	Iteration 1	Iteration 2	Iteration 3	Iteration 4
HJI analysis	9.02	5.71	5.64	5.45
RL	8.47	8.49	8.36	8.37
GP regression	4.20	3.68	3.57	3.52
Total	82.23			

Table 6.4: Execution times for each iteration and in total. All times are given in seconds.

6.2 Policy Learning

To begin with, we look at the distribution of 1000 random samples drawn from the recorded samples of all four iterations. Figure 6.1 shows the sampling results of the algorithm with exploration compared to the one without exploration. It can clearly be seen that the samples are a lot more wide-spread if exploration is done. Without exploration, the learning algorithm always decides for the action which is the most promising in order to keep the system close to the origin. This causes concentration of samples around this point that might inhibit both the disturbance estimation and the policy learning at the edges of the safe set.



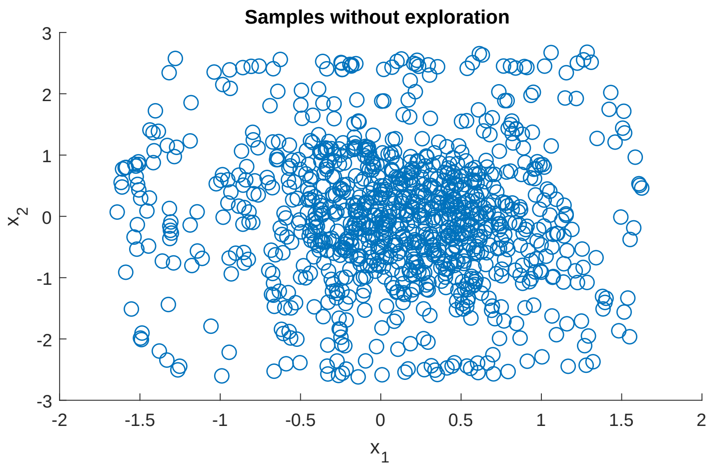


Figure 6.1: Random sampling with and without Incremental Q-learning. In the upper figure, the spread of the samples is clearly better, whereas in the lower figure most samples are concentrated around the origin.

It is therefore interesting to examine whether the Incremental Q-learning, as described in Section 5.5, yields a better estimation of the policy. The results of this comparison are depicted in Figure 6.2. The figure shows the estimated policy from each state, where the respective colors correspond to values of u. The quality of the estimated policy can be decided upon by comparing to the optimal policy which has been found with Policy Iteration (left figure). Clearly, the results are better for the approach which incorporates exploration (middle figure). This can especially seen at the edges of the safe set, where the estimated policy without exploration is very inaccurate (right figure).

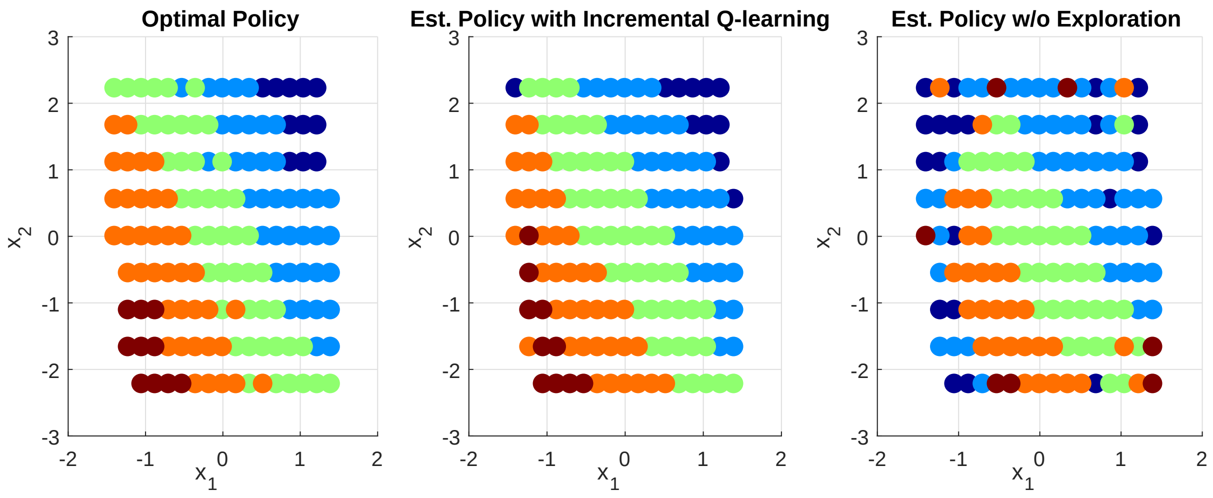


Figure 6.2: Policy estimated with Incremental Q-learning, ϵ-greedy exploration and without exploration. After the same number of learning iterations, the algorithm with Incremental Q-learning achieves considerably better results above all at the edges of the safe set.

To quantify the findings from Figure 6.2, the absolute error between the estimated and the optimal policy is illustrated in Figure 6.3. In addition to the algorithm without exploration and with Incremental Q-learning, the error with ϵ-greedy exploration is shown. It becomes apparent that the Incremental Q-learning approach takes slightly longer to converge but ends up with a remarkably better result. One can further conclude that the deterministic approach to exploration, Incremental Q-learning, pays off in comparison to the randomised ϵ-greedy exploration.

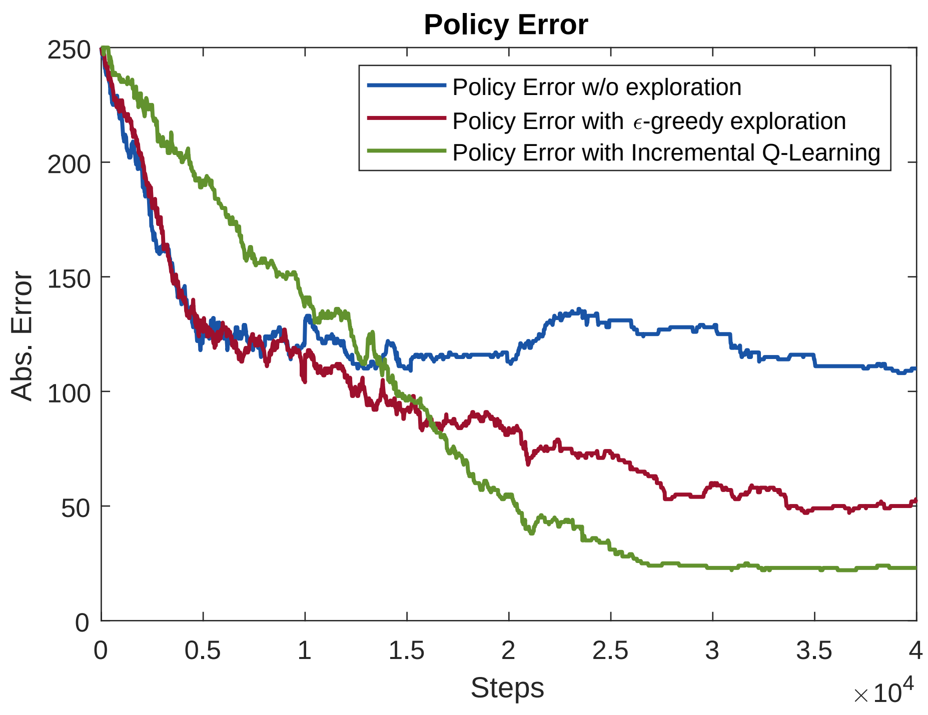


Figure 6.3: Policy error with Incremental Q-learning, ϵ-greedy eploration and without exploration. The approach with Incremental Q-learning converges slightly slower but the final error without exploration is about four times higher than the error of the Incremental Q-learning algorithm.

However, in the long run, pure exploration might not be the most desirable strategy. After all states have been visited sufficiently often, one would wish to exploit the most promising strategy. In the present system, this means that after sufficient exploration one would wish to keep the pendulum stable around the origin. This can for example be realised by switching off the exploration after all states have been visited 100 times. It is therefore interesting to investigate, if the exploration algorithm can guarantee that all states are visited a sufficient number of times. Figure 6.4 shows the number of visits of the 10 least visited states with and without exploration.

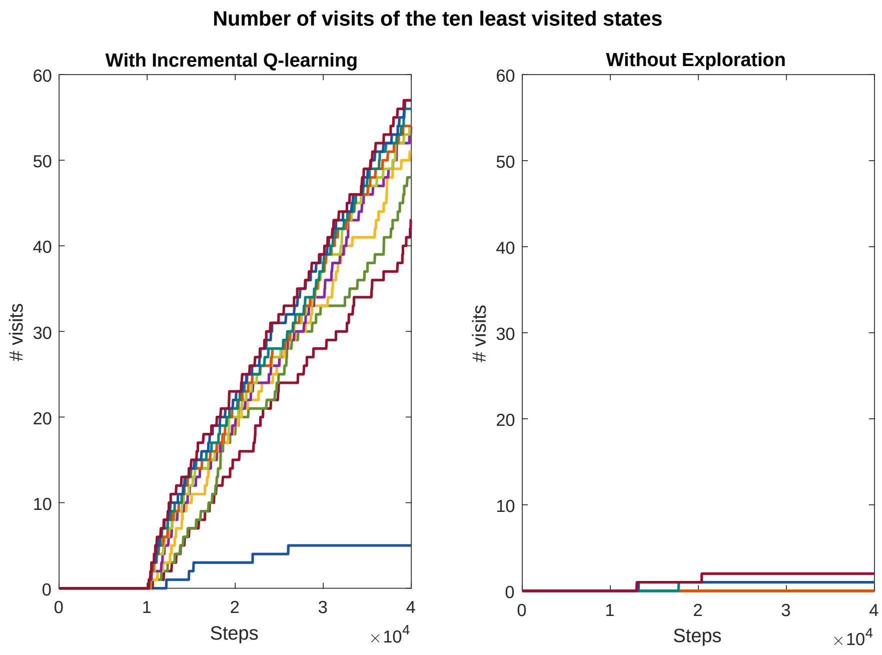


Figure 6.4: Number of visits of the 10 least visited states with and without exploration respectively.

Without exploration, several states remain unvisited for all $40,000$ steps. None of the 10 least visited state is visited more than ten times. With Incremental Q-learning, the system passes all states within the first $20,000$ steps. Except from one, all states count more than 40 visits within the observed $40,000$ steps. This indicates that it might indeed be possible to switch off the Incremental Q-learning after a certain number of iterations. A suitable criterion could be identified by examining how often a state approximately needs to be visited to eliminate the policy error. Yet, the topic of policy convergence has not yet been well covered as current results mostly focus on the convergence results of state values.

6.3 Disturbance Estimation and Safe Set Computations

In this section, the influence of exploration on the disturbance estimation will be examined. Figure 6.5 shows the result from the disturbance estimation with GP regression after each iteration with the above described setup. As only states within the safe set can be visited, no estimate of a policy outside that set can be done. Therefore, only policy values within the safe set are compared. The left and right half of the figure show the result without exploration and Incremental Q-learning respectively. For both algorithms, it can be seen that the disturbance estimation improves after each iteration. This is due to the fact that the system visits wider regions of the state space with an increasing number of steps. Therefore, the samples taken from the recorded data points are spread out wider as more steps are recorded. Furthermore, it becomes evident that the disturbance estimation indeed is better for the approach with exploration. In the case without exploration, the samples are more concentrated in the front half of the coordinate system so that the disturbance estimate is good in that region but almost constant in the rear half and outside the safe set. In the case with exploration, the sinusoidal shape of the disturbance is clearly visible from the third iteration onwards. The disturbance estimate is quite accurate for all states within the safe set.

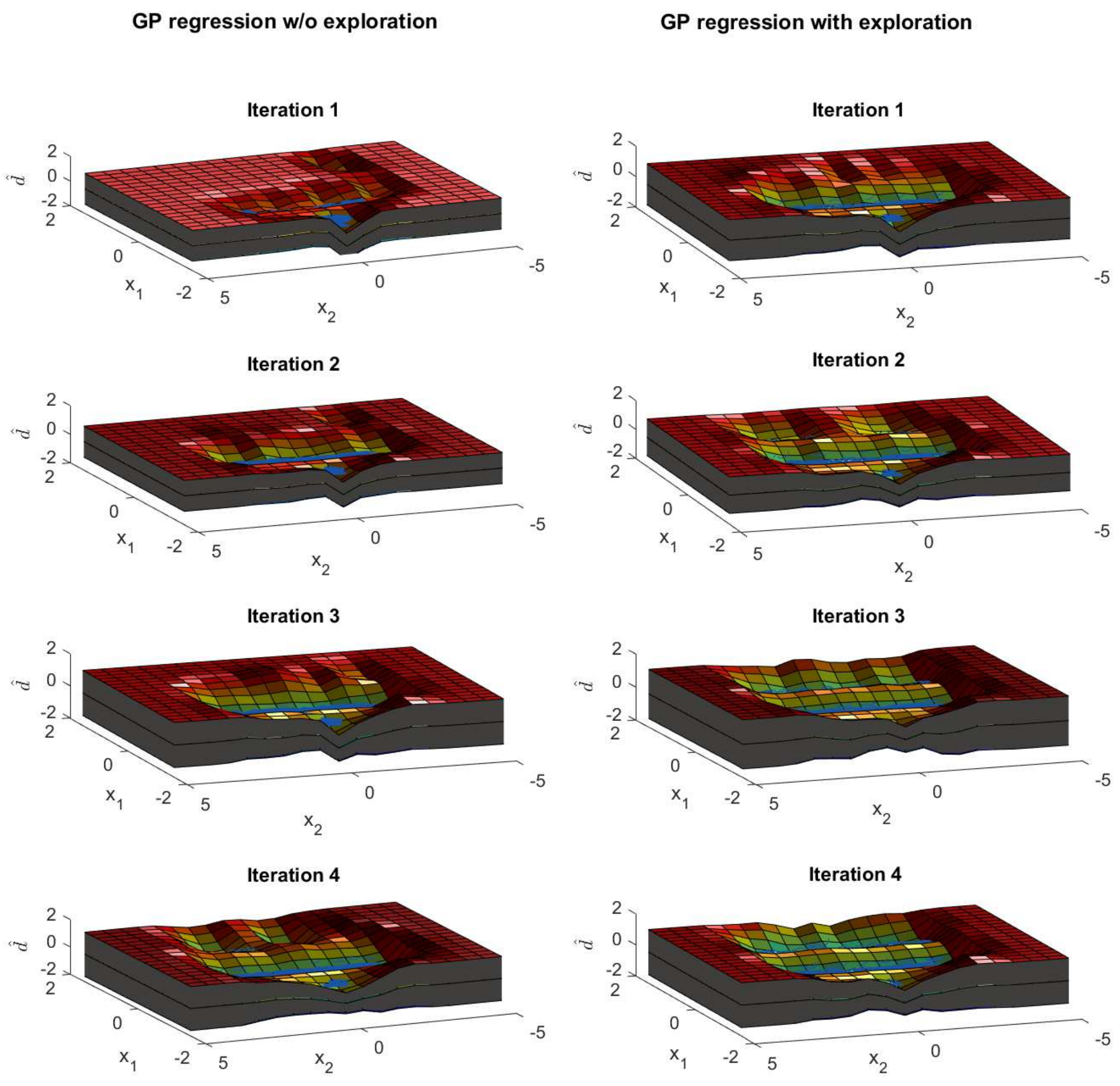


Figure 6.5: GP regression in four iterations with each 10,000 steps.

Figure 6.6 shows the safe set computations for all four iterations. During the first iteration, the conservative initial bound is used for the safe set calculations and gives a relatively small initial safe set. By calculating a better disturbance estimate in the GP regression, the safe set grows already largely in the second iteration.

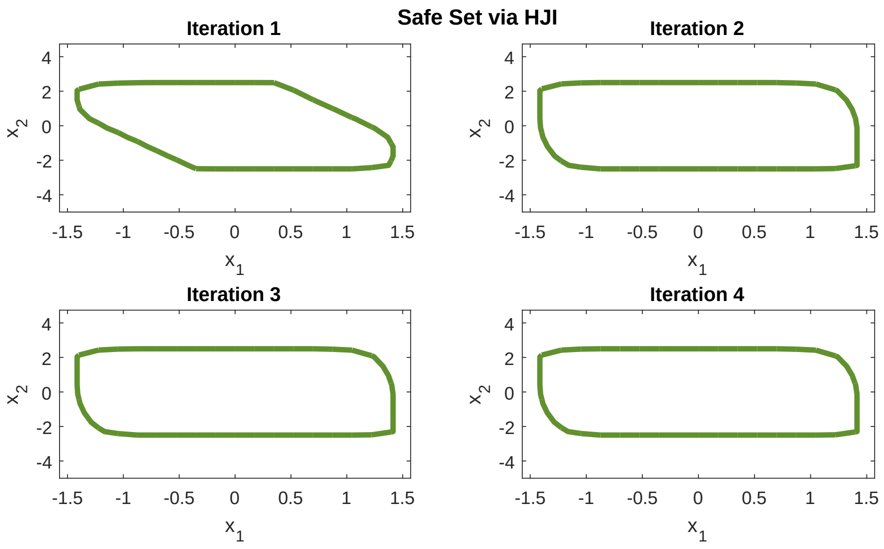


Figure 6.6: Safe set computation in four iterations. The better disturbance estimate causes an increase in the safe states already after the first iteration.

Interestingly, the safe set resulting from the disturbance estimate is the same without and with exploration. This could be attributed to the fact that the safe set is an over-approximation of the true safe set that not necessarily increases as the disturbance bounds decrease. This claim is supported by the fact that the safe set remains the same even if a disturbance of $d = 0\,\mathrm{N}$ is assumed.

6.4 Simulations

Finally, simulation results from both approaches are portrayed in Figure 6.7. The depicted trajectories show the last $500,000$ samples of the test run with four iterations that have been recorded with $h_{\text{safe}} = 0.005\,\mathrm{s}$. It becomes clear that the algorithm with exploration causes the system to explore the whole state space and repeatedly hits the borders of the safe set. Since the safety check is not performed continuously, the system leaves the safe set at some points, but can each time be brought back into the set. In the right half of the figure it is apparent that the approach without exploration manages to keep the system close to the origin.

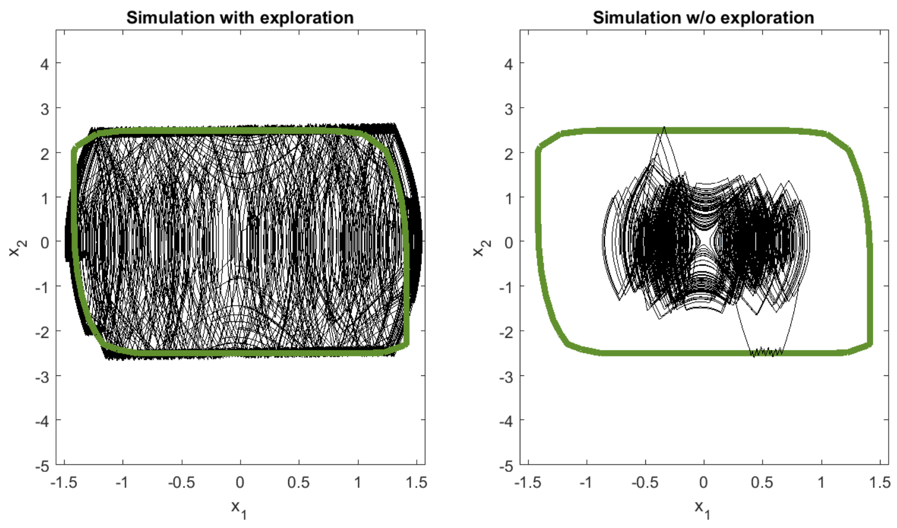


Figure 6.7: Simulation of the system with Incremental Q-learning versus simulation without exploration.

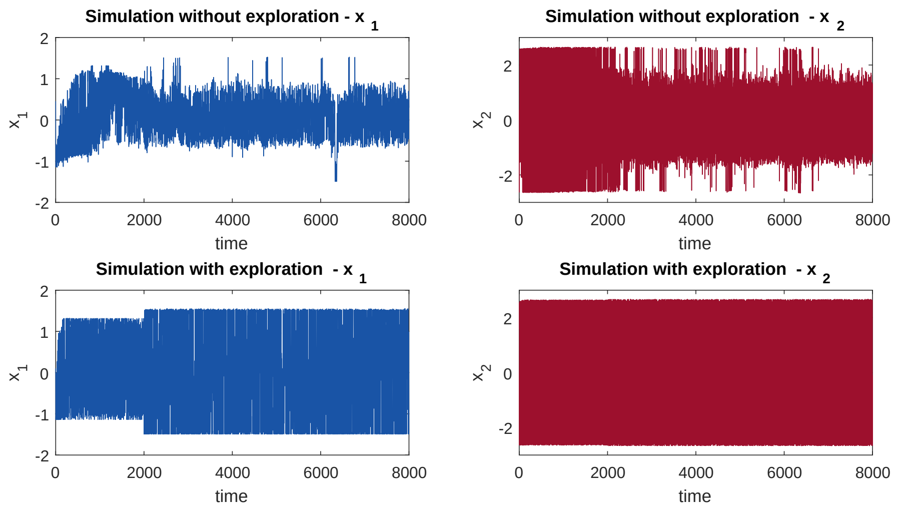


Figure 6.8: Simulation of the system with Incremental Q-learning versus simulation without exploration.

Chapter 7

Conclusions and Future Work

In this chapter, the results of the thesis and possibilities for future improvements will be discussed.

7.1 Conclusions

This thesis aims at providing a way of safely learning the control for a system. Applying reinforcement learning methods to control tasks is a promising approach to overcome the strong dependence of traditional model-based control methods on accurate models. Model-free reinforcement learning techniques such as Q-learning have been extensively studied and proven their value in various applications. However, the problem of how to satisfy constraints during the learning process has not yet been addressed yet. Therefore, Hamilton-Jacobi-Isaacs (HJI) reachability analysis has been incorporated into the algorithm to ensure safety during the learning process. By modelling the unknown parts of the state-space model as an unknown additive disturbance with known conservative bounds, this method provides a way to calculate a safe set. Within this set the system can move freely as long as a safe control is applied at the edges. Combining reinforcement learning and HJI analysis provides a way to safely learn a control for a system with uncertain dynamics. To update the initial conservative bound on the disturbance, we then iteratively estimate the disturbance on the basis of collected data points. To this end, we apply Gaussian Processes that not only provide an estimate for the disturbance but also a measure for how certain this estimate is. Gaussian Process (GP) regression is a non-parametric regression method that is ideally suited for our purpose, since it is crucial to not under-approximate the disturbance. By updating the disturbance with a safety margin based on the certainty of the estimate, we can precisely say how good our safety guarantee is.

To make all methods run smoothly, we decided to run the algorithm on two different time-scales. The safety loop runs with a very small sample time to ensure that every constraint violation will be detected immediately. The reinforcement learning algorithm requires a longer time step to work properly.

Compared to the approach presented in [1], the method has been extended to incorporate exploration. In Chapter 6, we extensively compared the approaches with exploration and without exploration. We find that both, policy learning and disturbance estimation, can be considerably improved by encouraging exploration. This holds especially for the edges of the safe set that will barely be visited without exploration.

7.2 Future Work

In the following section, some points that could be interesting to improve in the future will be listed.

Integration of Learning and Safety Loop. Firstly, the collaboration of the different methods is not satisfactory in some regards. The safety loop of the algorithm solely passes an estimate of the disturbance bounds to the learning loop and runs apart from that completely separate. It would be preferable to have a joint design of safety loop and learning loop, i.e. to pass some more information from the safety loop to the learning loop to warm-start the Reinforcement Learning algorithm. For instance, the policy could be initialized to the safe control from the safety algorithm or conclusions about transition probabilities could be drawn from the recorded samples. Furthermore, the fact that the algorithm runs on two different time scales could render the algorithm suboptimal. Finding a way to run both loops on the same time scale might increase the efficiency of the algorithm substantially. Yet, we do not know at this point, how this could be achieved.

Recursive Disturbance Estimation. Secondly, the disturbance estimation could possibly be improved. The GP regression is implemented batchwise with batches of 1000 samples each. This is due to the computational complexity of the method. The disturbance estimation might be improved considerably by employing a recursive method that takes all recorded samples into account. It is however not trivial how to implement GP regression in a recursive manner because the method relies on re-computing the covariances for each new input.

Guarantee to always stay in the Safe Set. Finally, the most significant improvement could be made by guaranteeing that the system never leaves the safe set. The two main problems that cause the system to sporadically leave the safe set in the current algorithm are known: Firstly, the numerical differentiation of the recorded data points leads to a faulty disturbance estimate, which distorts the safe set. Secondly, the safety check is not performed continuously which allows the system to leave the safe set in between two samples. The problem could for instance be tackled by shrinking the safe set with regard to the sampling time.

In summary, we remark that ideas from this thesis could serve as a good starting point for future research. Even though some adjustments, i.e. a closer integration of the different methods, should be made, the approach of safe learning for control applications is really interesting and could overcome some of the limitations of traditional control.

Bibliography

[1] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin, "Reachability-based safe learning with gaussian processes," in *Conference on Decision and Control , IEEE 53rd Annual Conference*, pp. 1424–1431, 2014.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[3] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[4] T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," *Neural computation*, vol. 6, no. 6, pp. 1185–1201, 1994.

[5] M. G. Azar, R. Munos, M. Ghavamzadeh, H. J. Kappen, *et al.*, "Speedy Q-learning.," in *Neural Information Processing Systems*, pp. 2411–2419, 2011.

[6] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "PAC model-free reinforcement learning," in *International Conference on Machine Learning*, pp. 881–888, 2006.

[7] J. N. Tsitsiklis, B. Van Roy, *et al.*, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[8] K. P. Murphy, *Machine learning: A probabilistic perspective.* MIT press, 2012.

[9] C. E. Rasmussen, *Gaussian processes for Machine Learning.* MIT press, 2006.

[10] I. M. Mitchell, "A toolbox of level set methods," *Department of Computer Science, University British Columbia, Vancouver, BC, Canada*, 2004.

[11] I. M. Mitchell, *Application of level set methods to control and reachability problems in continuous and hybrid systems.* University Microfilms, 2003.

[12] K. J. Åström and B. Wittenmark, *Adaptive control.* Courier Corporation, 2013.

[13] L.-X. Wang, "Stable adaptive fuzzy control of nonlinear systems," *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 2, pp. 146–155, 1993.

[14] K. Doya, "Reinforcement learning in continuous time and space," *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.

[15] C. E. Rasmussen and H. Nickisch, "Gaussian processes for machine learning (GPML) toolbox," *Journal of Machine Learning Research*, vol. 11, pp. 3011–3015, Dec. 2010.

[16] E. Even-Dar and Y. Mansour, "Convergence of optimistic and incremental Q-learning," in *Neural Information Processing Systems*, pp. 1499–1506, 2001.