Safe Learning for Control

CAROLINE HEIDENREICH

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

Chapter 1

Introduction

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 should be controlled. As models usually reflect only parts of the reality, those approaches often suffer from poor model accuracy. Learning the control online 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 problems is however that those methods learn in a trial-and-error fashion and are therefore not designed to satisfy constraints during the learning process. 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.

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 [13]. 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

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

where T describes the horizon of the problem (that 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 then is 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.

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.

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

which means that a state with higher future rewards 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

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

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 one with the optimal state-value function

$$V^*(s) = \max_\pi V^\pi(s) \qquad \pi(s)^* \in \arg\max_\pi V(s)^\pi \tag{2.4}$$

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

$$V^*(s) = \max_a Q(s,a) \tag{2.5}$$

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 shortly 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:

$$V_{k+1} = \max_a \left[r(s,a) + \gamma \sum_{s'} T(s,a,s')V_k(s') \right]. \tag{2.6}$$

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 directly a policy π. As the policy converges usually 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 and obtain the optimal policy by solving that model. 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 from the policy followed. All algorithms explained below except from Delayed Q-Learning are off-policy algorithms.

In the following section some Reinforcement algorithms will be introduced. A classical algorithm in the field of reinforcement learning, known as Q-learning, was presented by Watkins in 1989 [16]. In its simplest form the learning update of this algorithm is formulated as

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \left[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right]. \tag{2.7}$$

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

$$Q_{t+1}(s, a) = Q_t(s, a) \qquad \forall (s, a) \neq (s_t, a_t). \tag{2.8}$$

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 [6]

$$\sum_{t=1}^{\infty} \alpha_t = \infty \tag{2.9}$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty \tag{2.10}$$

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.

$$Q_0 = V_{max} = \sum_{k=0}^{\infty} \gamma^k \max_{s,a} r(s,a) = \frac{\max_{s,a} r(s,a)}{1-\gamma}. \tag{2.11}$$

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 [3]. 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

$$\begin{aligned} Q_{t+1}(s_t,a_t) = {} & Q_t(s_t,a_t) + \alpha_k \left[r_{t+1} + \gamma \max_a Q_{t-1}(s_{t+1},a) - Q_t(s_t,a_t) \right] \\ & + (1-\alpha_k) \left[r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - r_{t+1} + \gamma \max_a Q_{t-1}(s_{t+1},a) \right]. \end{aligned} \tag{2.12}$$

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 [3].

Another variation of Q-learning is the Delayed Q-learning algorithm for the online problem to obtain PAC-MDP guarantee [12]. 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

$$Q_{t+1}(s_t,a_t) = \frac{1}{m} \sum_{i=1}^{m} \left(r_{k_i} + \gamma \max_a Q_{k_i}(s_{k_i},a) \right) + \varepsilon \tag{2.13}$$

where m and ε are inputs to the algorithm. For such 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. The authors then provide an upper bound that holds with high probability 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 [12].

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

$$Q_\theta(s,a) = \sum_{i=1}^{n} \theta_{i,a}\varphi_i(s) = \theta^T\varphi \tag{2.14}$$

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

$$\varphi_i(s) = \exp\left(\frac{-||s-m_i||^2}{2\sigma_i^2}\right) \tag{2.15}$$

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. The standard deviation of the basis functions for instance determines over which distance generalization takes place. 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:

$$\delta_t = r_{t+1} + \gamma \max_a Q_\theta(s_{t+1},a) - Q_\theta(s_t,a_t) \tag{2.16}$$

$$\theta_{\bullet,a_{t+1}} = \theta_{\bullet,a_t} + \alpha \cdot \delta_t \cdot \varphi_t(s_t). \tag{2.17}$$

A detailed convergence analysis of this method can be found in [14].

2.2 Gaussian Processes for Regression

The following introduction to Gaussian processes is based on [9] and [10]. In the latter, Gaussian processes are defined to be a "collection of random variables, any finite number of which have (consistent) joint Gaussian distributions". Gaussian processes 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, Gaussian processes are over functions and defined by a mean function and a covariance function.

Gaussian processes can be applied for regression in the following manner. A Gaussian process

$$f(x) \sim GP(m(x), \kappa(x,x')) \tag{2.18}$$

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

$$\begin{pmatrix} \mathbf{f}(\mathbf{X}) \\ \mathbf{f}(\mathbf{X}^*) \end{pmatrix} = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right) \tag{2.19}$$

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

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f}) = \mathcal{N}(\mathbf{f}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \tag{2.20}$$

$$\boldsymbol{\mu}_* = \boldsymbol{\mu}(\mathbf{X}_*) + \mathbf{K}_*^T\mathbf{K}^{-1}(\mathbf{f} - \boldsymbol{\mu}(\mathbf{X})) \tag{2.21}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T\mathbf{K}^{-1}\mathbf{K}_*. \tag{2.22}$$

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

$$\mathbf{K}_y = \mathbf{K} + \sigma_y^2\mathbf{I}_N \tag{2.23}$$

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 Gaussian process 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:

$$\kappa(x, x') = \sigma_f^2 \exp(-\frac{1}{2l^2}(x - x')^2). \tag{2.24}$$

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

$$L(y|X, \theta) = \log p(y|x, \theta) = \log \mathcal{N}(0, \mathbf{K}_y) \tag{2.25}$$

$$= -\frac{1}{2}\log|\mathbf{K}_y| - \frac{1}{2}y^T\mathbf{K}_y^{-1}y - \frac{n}{2}\log(2\pi). \tag{2.26}$$

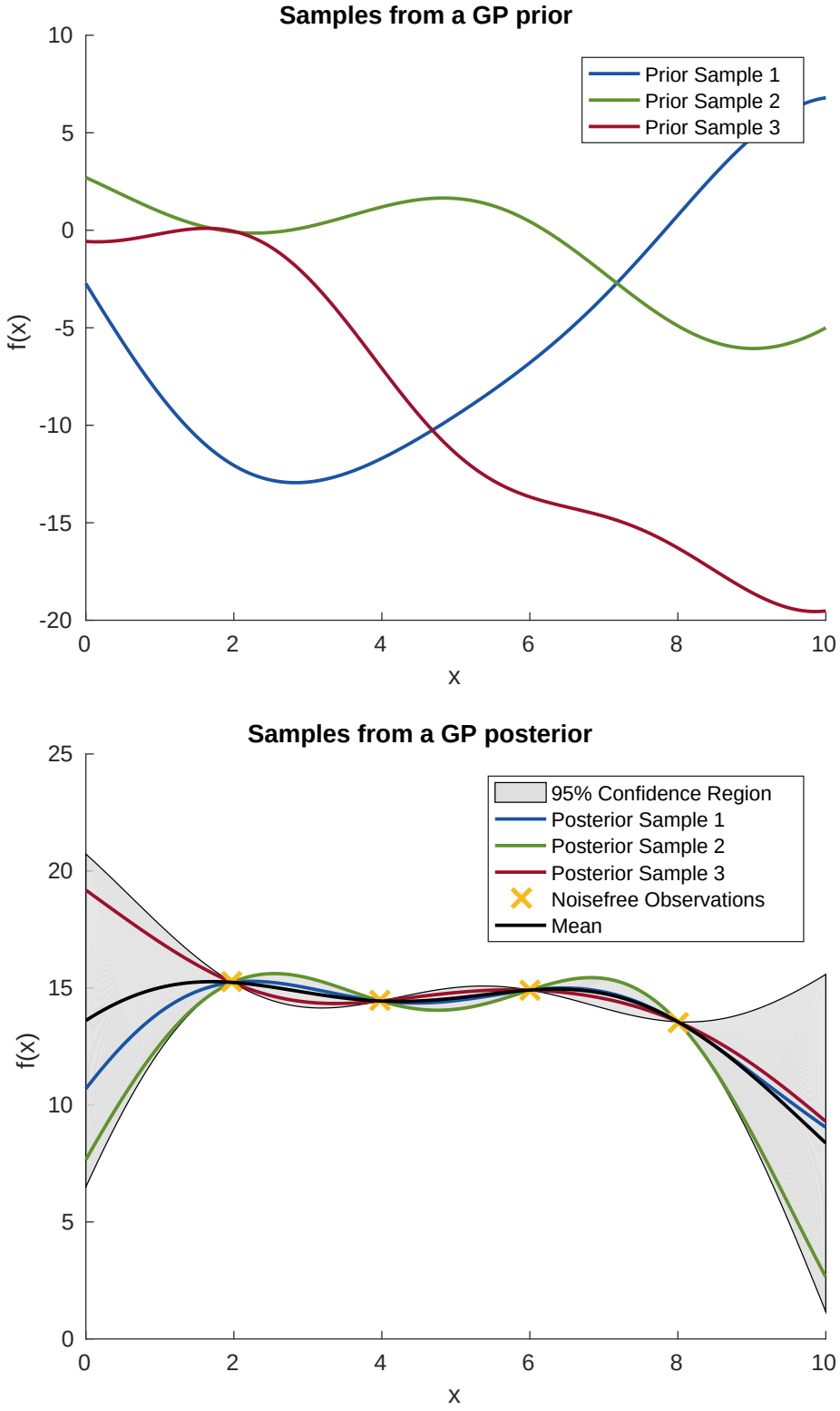

Figure 2.1: The upper figure shows samples from a Gaussian prior with a squared exponential kernel. The lower figure shows samples from the Gaussian posterior conditioned on four noisefree observations (yellow). Additionally, the mean function (black) and the $\pm 2\sigma$-confidence region (grey) is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

where 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 means that a trade-off between complexity and data-fit is automatically introduced and does not need to be tuned manually [10]. Maximising the likelihood function can be done based on the derivative

$$\frac{\partial L}{\partial \theta_i} = \frac{1}{2} \text{ trace} \left(\mathbf{K}_y^{-1} \frac{\partial \mathbf{K}_y}{\partial \theta_i} \right) + \frac{1}{2} y^T \mathbf{K}_y^{-1} \frac{\partial \mathbf{K}_y}{\partial \theta_i} \mathbf{K}_y^{-1} y \tag{2.27}$$

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 safety within the learning framework. Both explanation and implementation are based on Ian M. Mitchell's Level Set Toolbox [8] and doctoral thesis [7].

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 $t \in [-\tau, 0]$. The applied method does not only output the set $\mathcal{S}_\tau$ but gives also rise to a control strategy u_{safe} that guarantees to keep the system within $\mathcal{S}_\tau$. Furthermore, this control strategy only acts at the boundaries of the safe set so that any control action can be applied within the safe set.

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

$$\mathcal{T}_0 = \{x \in \mathbb{R}^n | g(x) \leq 0\}. \tag{2.28}$$

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

$$\frac{\partial}{\partial t}\phi(x,t) + \min\left[0, H\left(x, \frac{\partial}{\partial x}\phi(x,t)\right)\right] = 0, \qquad \forall t \in [-T, 0], \forall x \in \mathbb{R}^n \tag{2.29}$$

$$\phi(x, 0) = g(x), \qquad \forall x \in \mathbb{R}^n \tag{2.30}$$

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

$$H(x,p) = \max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} p^T f(x, u, d). \tag{2.31}$$

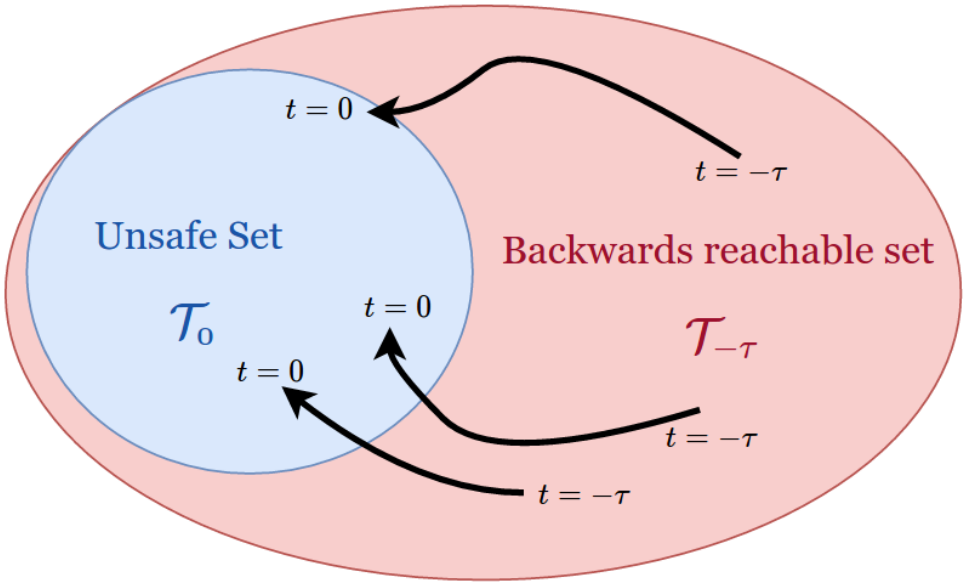

Figure 2.2: The backwards reachable set $\mathcal{T}_\tau$ is the set from which trajectories can reach the unsafe set within time τ.

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 $\mathcal{T}_\tau$:

$$\mathcal{T}_\tau = \{x \in \mathbb{R}^n | \phi(x, t) \leq 0\}, \qquad \forall \tau \in [0, T]. \tag{2.32}$$

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 (2.31) given by

$$u_{\text{safe}} = \arg\max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} p^T f(x, u, d) \tag{2.33}$$

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 online is not a recent one. There are two main strategies that researchers have been following. Firstly, data can be used to improve the model of the system online. This approach has already been pursued in the field of Indirect Adaptive Control. More recently, Gaussian processes have been proven to be a valuable tool for nonlinear modelling [9]. Secondly, learning methods can also be used to directly learn a control for a given system. An early example of this approach is Iterative Learning Control [2]. While this technique is constrained to a predefined repetitive reference trajectory, Reinforcement Learning techniques such as Q-learning can be applied to a broad range of control problems [13].

An example for an early contribution to this matter is the article of Wang et. al. on stable adaptive fuzzy control [15]. The proposed method assumes that the state space is on a special form

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

with a 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 framework but realizes some early form of "safe learning for control".

Further algorithms have been proposed in recent years. This thesis will mainly focus on the approach proposed in [1]. The approach assumes an unknown state-dependent disturbance $d(x)$ that is added to the known parts of the system dynamics $h(x)$ and $g(x)$:

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

Assuming upper bounds on the disturbance a safe set can be calculated using Hamilton-Jacobi-Isaacs (HJI) reachability analysis. Starting from the 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 state possible is applied. In the objective function of the PGSD a term is included to reduce switching between the safe and the learning control. Compared to [15], this approach poses less restrictions on the form of the state space. An issue with the approach in [1] 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. This conflicts with the goal of increasing the safe set as the disturbance estimate will remain uncertain in regions with no or few samples.

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 single states. As in [1], a system with an unknown state-dependent disturbance that is added to the known part of the system dynamics is assumed. Additionally, a conservative bound on the disturbance is known initially that will be iteratively updated with a Gaussian Process model. Based on the initial disturbance estimate and a safe set the backwards reachable set that 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. At the borders of this set the safe controller that is also calculated in the HJI framework can then be applied to keep the system within in the safe region. 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 yield 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 Gaussian Process model. The Gaussian Process estimates a less conservative bound for the disturbance such that a larger safe set can then be calculated with the HJI 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 during the learning recorded samples are used to get a better estimate of the disturbance that results in a less restrictive safe control. In more detail, 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, that estimates the disturbance, calculates a safe set and a safe controller on that basis and checks every action that the learning controller wants

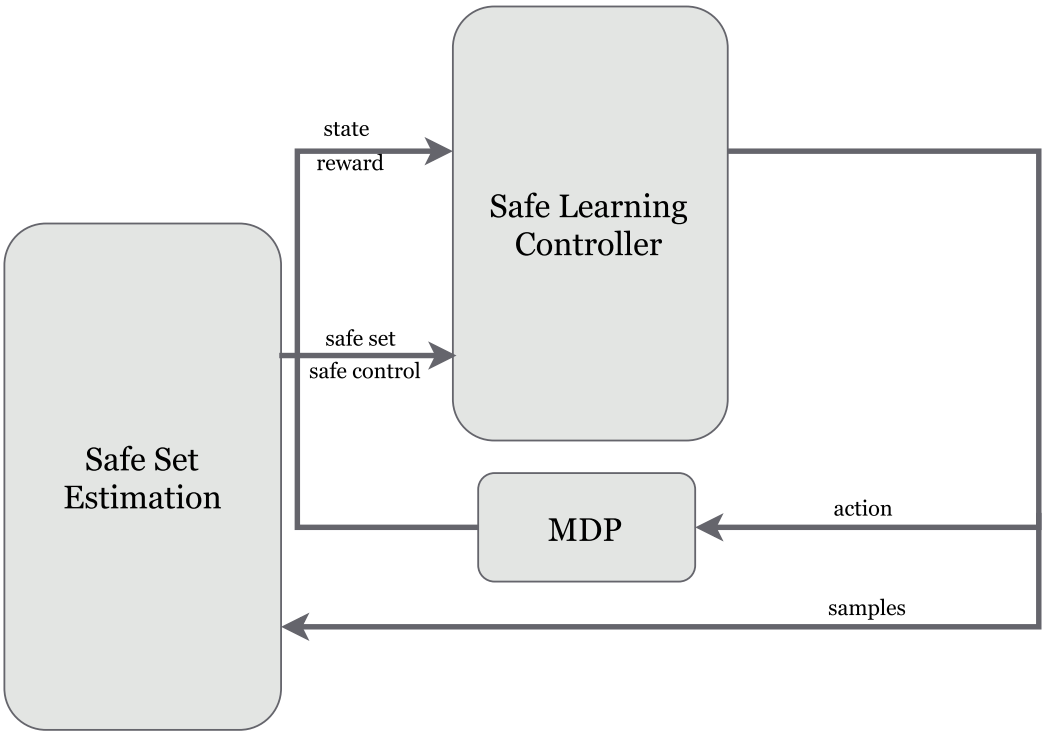

Figure 4.1: Rough Control Setup.

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 that the learning controller chooses, 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 explained further in section 5

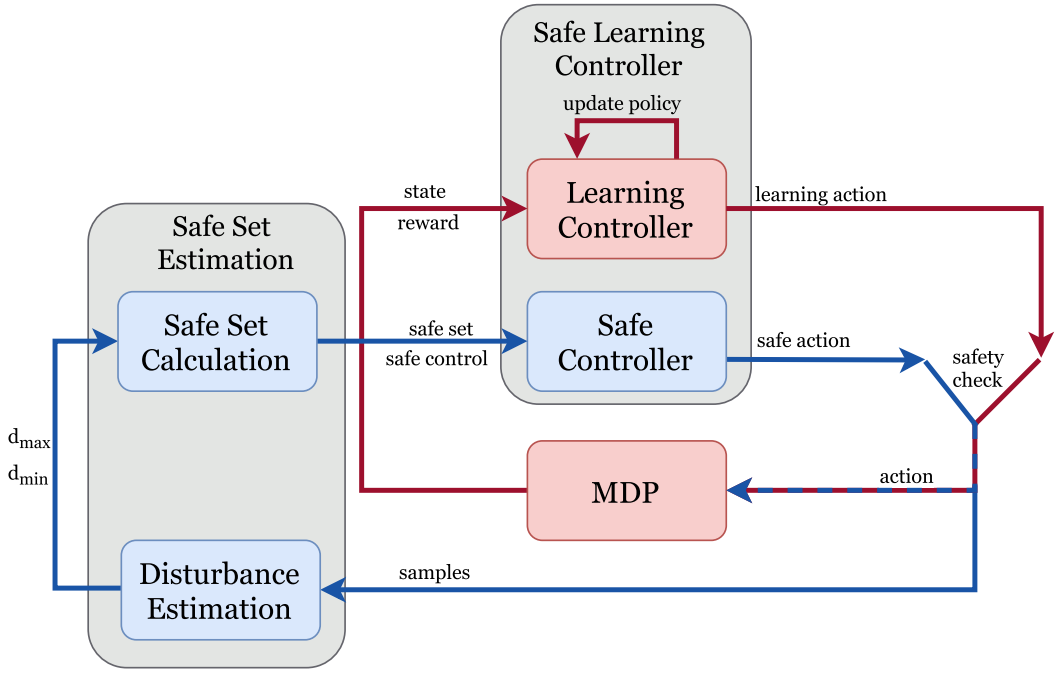

Figure 4.2: Detailed Control Setup.

Chapter 5

Implementation

In this section I will discuss the implementation details of the approach outlined in section 4. Starting from the modelling of the system as Markov Decision Process, I will then describe the implementation of the Reinforcement Learning algorithm, the disturbance estimation with Gaussian Processes and finally, the safe set calculation with HJI reachability analysis.

5.1 Markov Decision Process Model

The dynamics of the damped inverted pendulum system are described by [4]:

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

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

where m is the mass of the pendulum, l is the length of the pendulum, g is the gravity constant and b is the friction coefficient. All constants can be assumed to be positive. Modelling the inverted pendulum system as a MDP means to define the quintuple (S, A, T, r, γ) as defined in 2.1. Defining the finite set of states S and the finite set of actions A implies discretization 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 discretization interval is larger than its period 2π. The number of discretization steps is important for 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 important that the time step h is large enough to actually allow transitions. It h is too small the system will always stay in state s regardless of the chosen action. For the system we consider the time step should lie around $h = 0.2$.

The reward function $r(s)$ is defined as

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

where σ is a constant defining how narrow the reward function is. Defining the coordinate system with $x_1 = 0$ being on the top, this reward corresponds to the goal to keep 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.

5.2 Reinforcement Learning

On the obtained MDP Reinforcement Learning was then applied to learn an optimal control policy. The chosen algorithm is a slightly modified version of the in section 2.1 introduced Delayed Q-Learning.

In large parts the algorithm resembles the Delayed Q-Learning algorithm that was introduced in 2.1 with the update rule (2.13). 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

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

The initialization of the Q-values is done optimistically according to (2.11). 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 (2.9). Without any safety-preserving 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 100000 steps in comparison to the optimal policy that has been found with policy iteration. The colours correspond to certain action values. 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. The number of constraint violations is decreasing over time as can be seen in figure 5.2 because the algorithm learns the optimal policy but safety can never be guaranteed. To prevent those violations from happening, a safe controller will be introduced in the next section.

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
        $U(s_t,a_t) \leftarrow 0$
        $C(s_t,a_t) \leftarrow 0$
        $M(s_t,a_t) \leftarrow \min\{\lceil 1.02M(s_t,a_t)\rceil + 1, 500\}$ // increase batch size
        $\varepsilon \leftarrow \min\{1.1\varepsilon, \varepsilon_{\text{target}}\}$ // increase the threshold
      end if
    end if
  end for

5.3 Safe Set Computations based on Reachability Analysis

The general idea behind safe set computations has been described in chapter 2.3. In this section the concrete implementation on the inverted pendulum system will

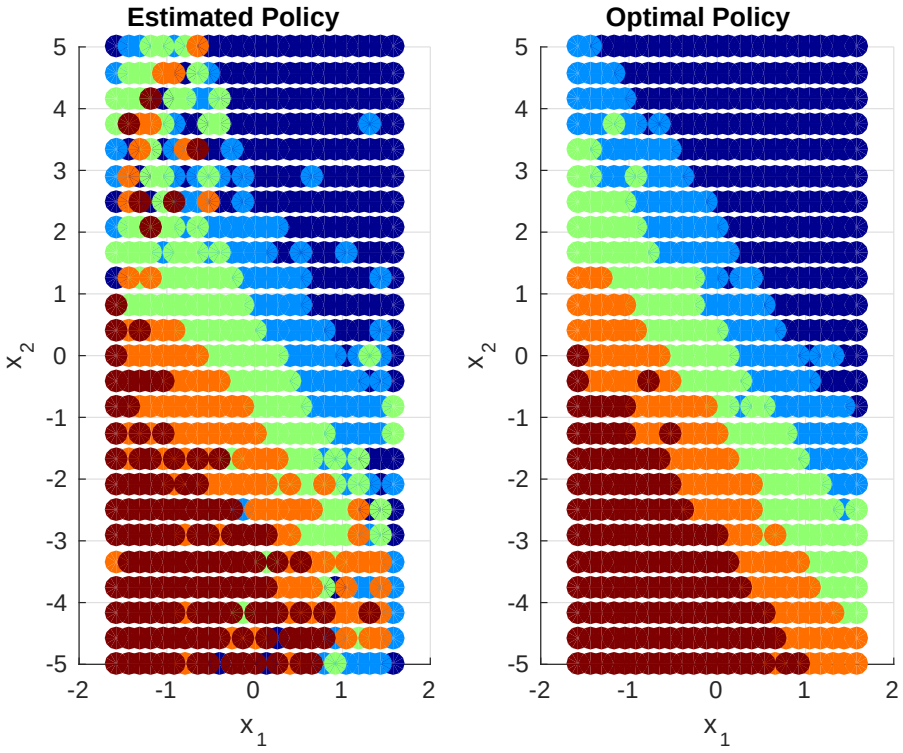

Figure 5.1: Comparison between the learned policy and the with policy iteration determined optimal policy.

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

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

This is the case for the inverted pendulum system with

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

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

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

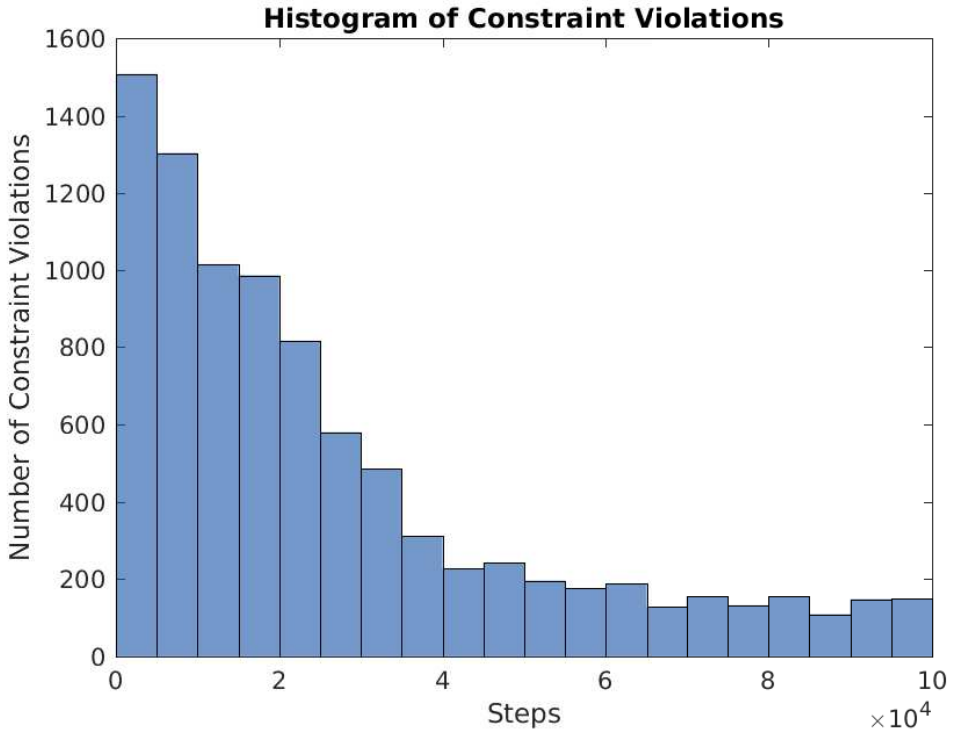

Figure 5.2: Evolution of the constraint violations over .

According to [8], the input maximising (5.7) and the disturbance minimizing the equation are then given by

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

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

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

$$U^{\max} = \max(|\underline{U}|, |\overline{U}|) \tag{5.14}$$

$$D^{\max} = \max(|\underline{D}|, |\overline{D}|), \tag{5.15}$$

u^* and d^* can be explicitly written as

$$u^* = \frac{1}{ml^2} U^{\max} \text{sign}(p_2) \tag{5.16}$$

$$d^* = -D^{\max} \text{sign}(p_2). \tag{5.17}$$

This results in the Hamiltonian

$$H(x,p) = p_1 x_2 + p_2 \left(\frac{g}{l} \sin(x_1) - \frac{b}{m} x_2 \right) + |p_2| \left(\frac{1}{ml^2} U^{\max} - D^{\max} \right) \tag{5.18}$$

that needs to 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

$$\hat{H}(x,p^+,p^-) \triangleq H\left(x, \frac{p^- + p^+}{2}\right) - \frac{1}{2}\alpha^T (p^+ - p^-) \tag{5.19}$$

where p^+ and p^- are left and right side approximations of p. $H(x,p)$ is given by (5.18).

$$\alpha_i(x) = \max_{p \in \mathcal{I}} \left| \frac{\partial H}{\partial p_i} \right| \tag{5.20}$$

with $\mathcal{I}$ being the hypercube containing all values that p takes over the computational domain needs to be implemented within the provided function prototype. This can be done with the over-approximation

$$\alpha_j(x) \leq |f_j^x(x)| + |\mathbf{F}_j^u| U^{\max} + |\mathbf{F}_j^d| D^{\max} \tag{5.21}$$

that for the present system resolves to

$$\alpha_1 \leq |x_2| \tag{5.22}$$

$$\alpha_2 \leq \left| \frac{g}{l} \sin(x_1) - \frac{b}{m} x_2 \right| + \frac{1}{ml^2} U^{\max} + D^{\max}. \tag{5.23}$$

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 and outputs for each state in the MDP if it is safe or not under time horizon τ and a safe control u^* for each state. 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 $\tau = 30s$. 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 $\mathcal{S}_\tau$ that is shown in the last subplot. Trajectories that start 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.

Moreover, to verify the calculations simulations of system trajectories have been done for both, initial conditions within the safe set $\mathcal{S}_\tau$ and without that set. The results can be seen in figure 5.4. Trajectories are marked with a red dot at the initial

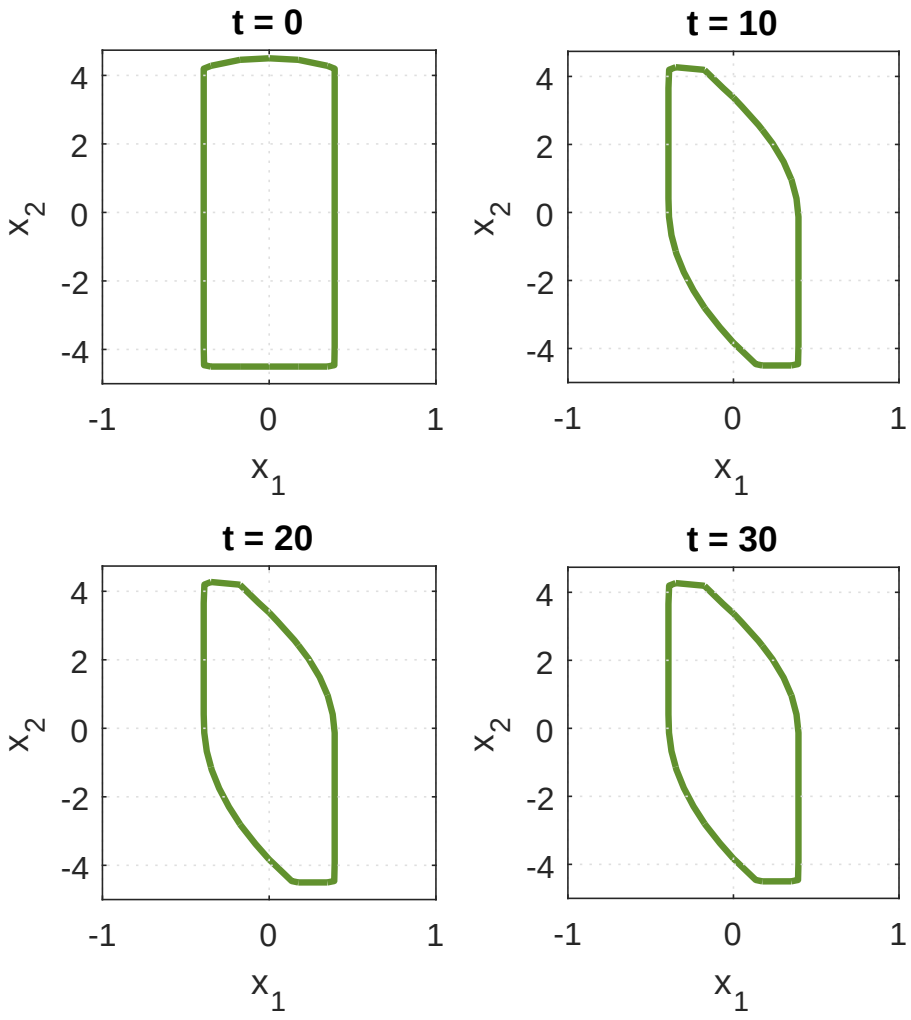

Figure 5.3: Example for the evolution of a safe set during the time horizon.

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 complete simulation time. The simulation time has been chosen to be smaller than the time horizon τ to keep the computation time low and the figure overseeable. The result holds however also for longer simulation times. Furthermore, the figure indicates that the safe set is an under-approximation of the true safe set as also the trajectories with initial conditions close to $\mathcal{S}_\tau$ can be stabilized. Yet, no definite conclusion about that can be drawn from simulations.

Another important remark is, that the safe set calculations assume continuous dynamics. It is therefore important to keep the time step h small so that the error

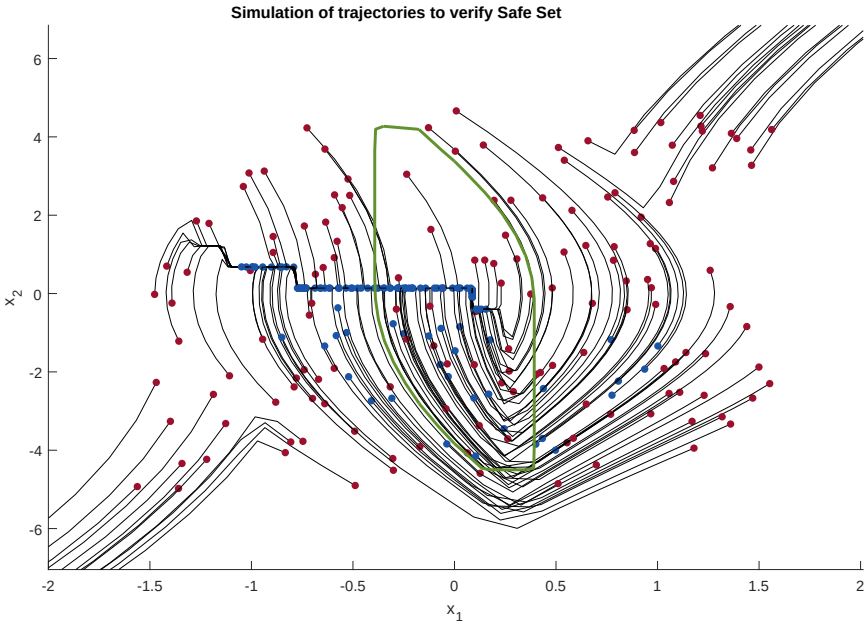

Figure 5.4: Simulations to verify that all trajectories that start.

through discretization does not become too large. This conflicts with the requirement from section 5.1 according to which the time step should 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, a large time step can lead to instability 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 that 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 discretization. This was however not within the scope of this thesis. This problem can be solved 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 $h_{\text{learn}} = 0.12s$ but the time step for the safety-preserving controller should be maximal $h_{\text{safe}} = 0.02s$, 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 can be seen 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} that large that the whole safe set can be crossed within one time step.

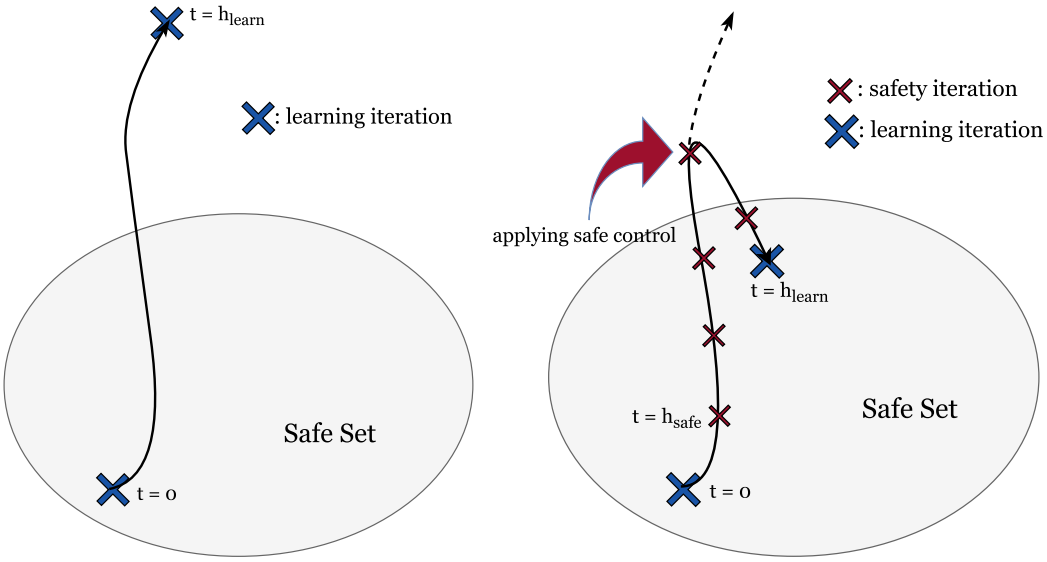

Figure 5.5: Illustration of the problem with a large time step h_{learn}. At $t = h_{\text{learn}}$ the system is without safety check (left) already far outside the safe set so that it cannot be guaranteed to be brought back again. On the right hand side, the same example is shown with a faster running 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 it will be explained how the disturbance estimation with Gaussian Processes works. 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 are available. Given data points that have been collected during the learning process, disturbance samples can be obtained by calculating:

$$\hat{d}(x) = \dot{x}_2 - \left(\frac{1}{ml^2} u + \frac{g}{l} \sin(x_1) - \frac{b}{m} x_2 \right). \tag{5.24}$$

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

$$\dot{x}_2 \approx \frac{x_{2_{k+1}} - x_{2_k}}{h}, \tag{5.25}$$

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 Gaussian Process model is employed. Gaussian Processes 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

$$\underline{d}(x) = \mu(x) - 3\sigma(x) \tag{5.26}$$

$$\overline{d}(x) = \mu(x) + 3\sigma(x), \tag{5.27}$$

where $\mu(x)$ 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 problems in the algorithm, because h_{safe} has been picked very small anyway to ensure safety, however, it might be interesting to account for the error due to numerical differentiation and discrete time and 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 Gaussian Process regression. As the regression is computationally expensive, it is not possible to feed all recorded samples into the Gaussian Process. 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 (as soon as a good control is learned) concentrate around the equilibrium point. Instead, a set of randomly chosen points that covers 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 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 Gaussian Process regression. The regression is done with the GPML toolbox [11] with a zero-mean function and a squared exponential kernel as given in (2.24).

Figure 5.7 aims at illustrating the results from a Gaussian Process regression for a disturbance $d = 2$ after 48000 recorded samples of which 1000 are fed into the Gaussian Process. The figure shows the 1000 input samples (blue) that are widely spread within the safe set. The upper and lower plane 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

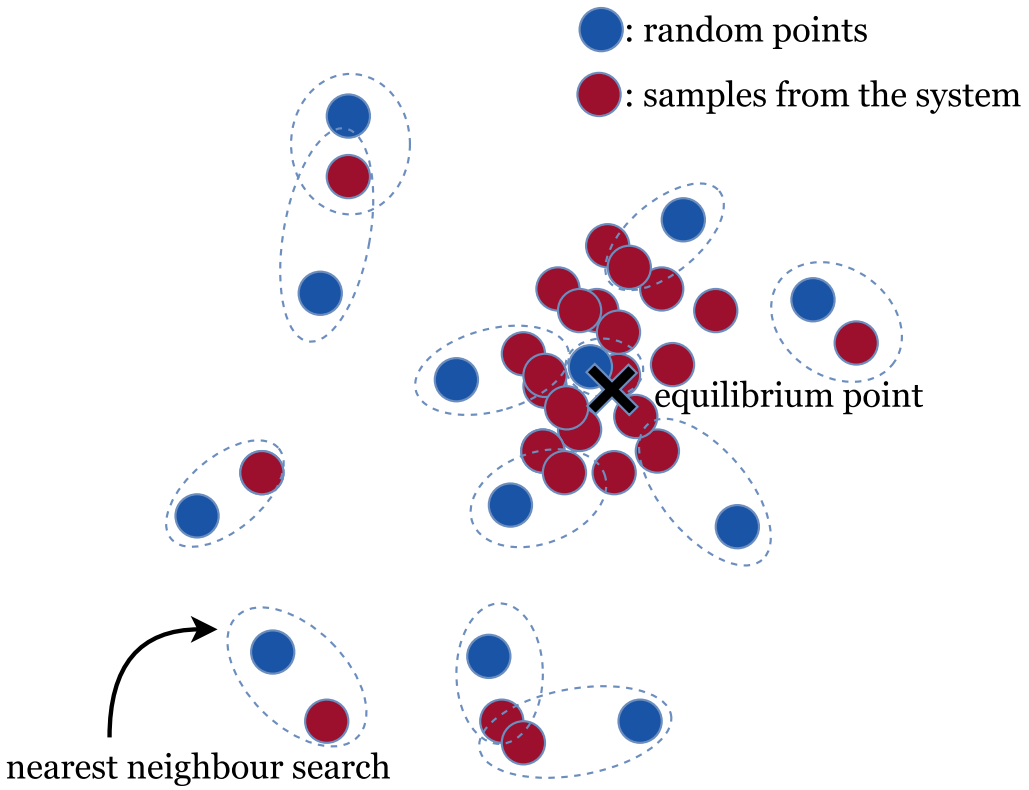

Figure 5.6: Choice of samples (red) by employing a grid (blue) and taking the nearest neighbours to the grid points in order to ensure that the samples cover the whole state space.

Gaussian Process outputs. 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 already obtained knowledge of the state space. In the inverted pendulum system this describes for instance the need of also visiting the borders of the safe set in order to get a better disturbance estimate and potentially increase the safe set instead of only staying close to the origin. In section 5.4, the need of 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

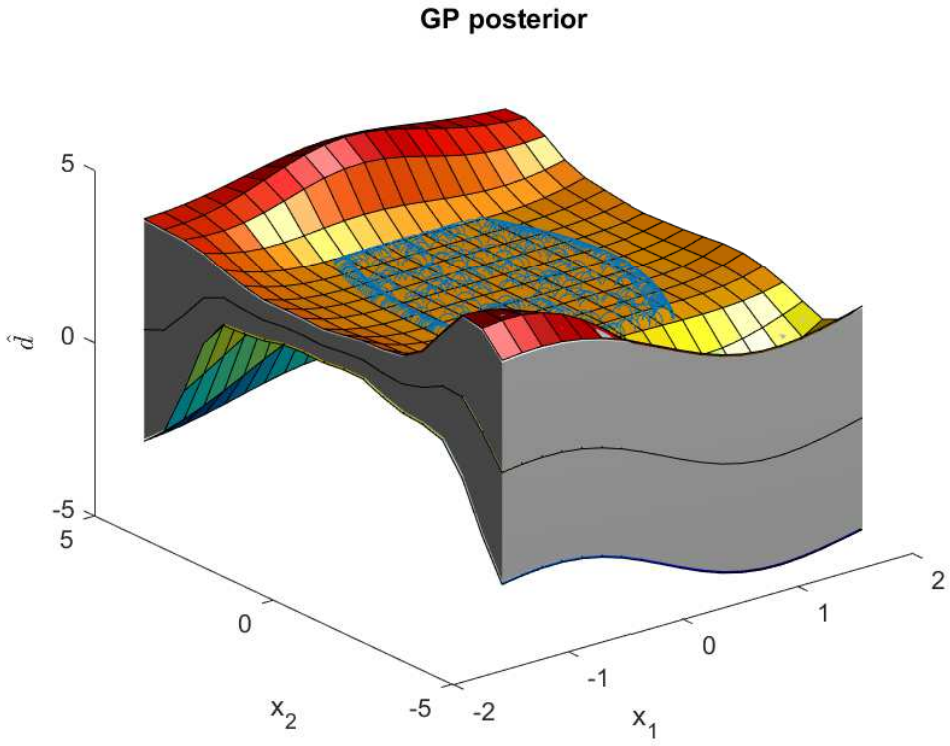

Figure 5.7: Disturbance estimation with GP regression for a disturbance. The lower and upper plane bound the $\pm 3\sigma$ confidence interval (shaded in grey). The samples that serve as input to the GP are shown as blue circles. The mean plane can be seen as a thin line in the middle of the confidence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

the system does so automatically. For this reason, a more structured way of doing exploration will be presented in this section and compared to the results without exploration. The approach chosen is called incremental Q-learning and aims at a de-randomisation of the in section 2.1 explained ϵ-greedy policy [5]. Instead of choosing random actions with probability ϵ, a greedy policy is chosen with respect to the Q-values plus a promotion term $A(s,a)$. $A(s,a)$ is initialized to zero and updated in the following way:

$$A_{t+1}(s,a) = \begin{cases} 0, & \text{if } s_t = s, a_t = a; \\ A_t(s,a) + \Phi_t(\#(s,a)), & \text{if } s_t = s, a_t \neq a; \\ A_t(s,a), & \text{if } s_t \neq s, a_t \neq a. \end{cases} \tag{5.28}$$

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

Results

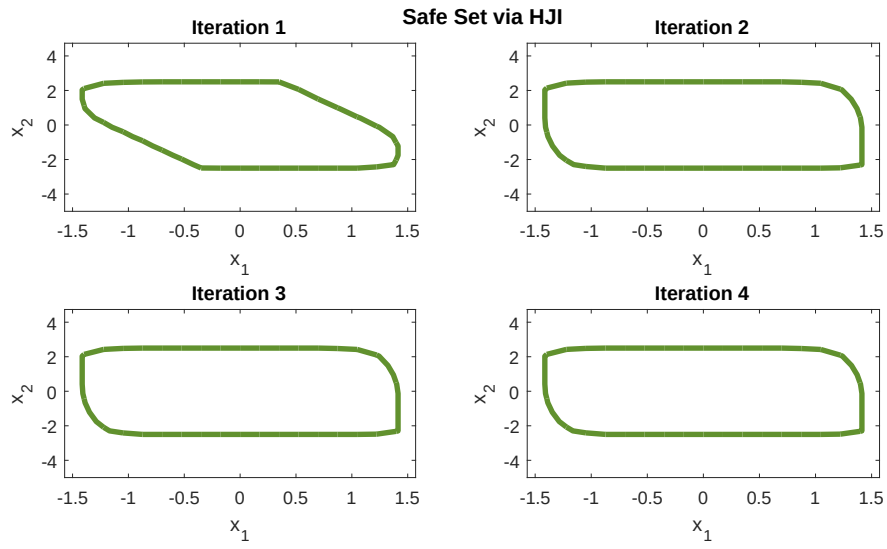

Figure 6.1: Safe Set Computation in four iterations. Clearly, the safe set already converges after the first iteration.

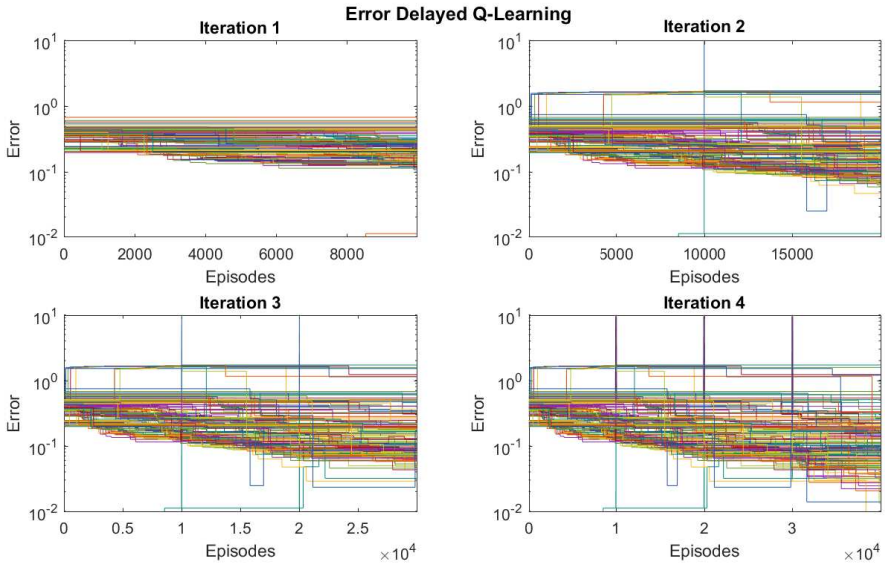

Figure 6.2: Logarithmic Error of the learned values compared to the values computed by Value Iteration. Each iteration consists of 10000 steps.

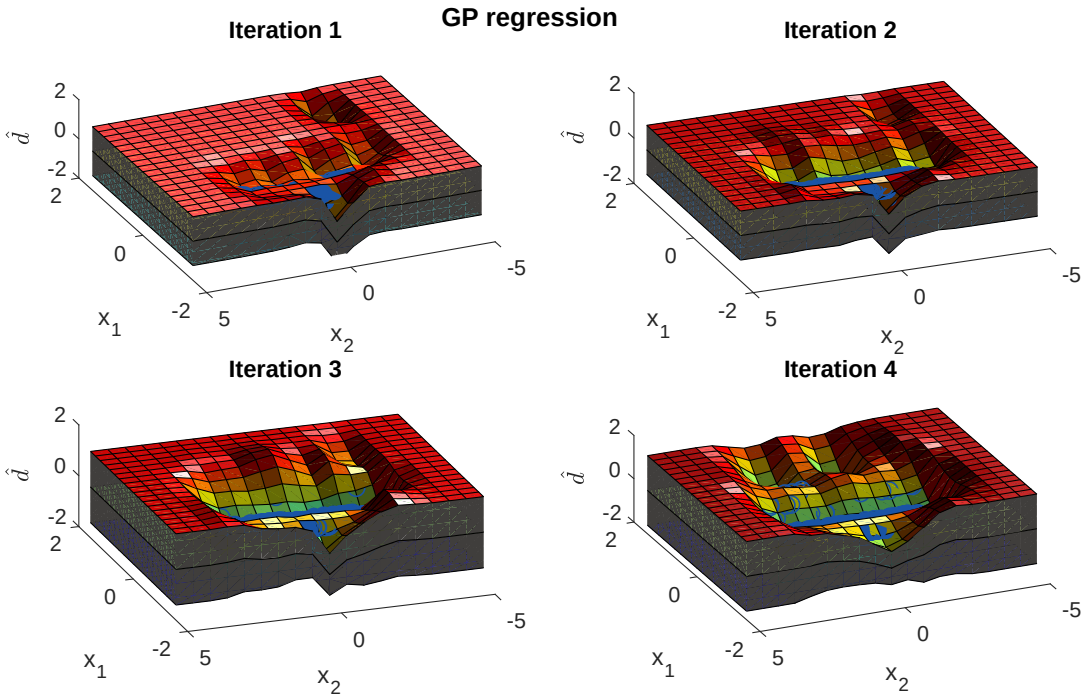

Figure 6.3: Gaussian Process regression in four iterations.

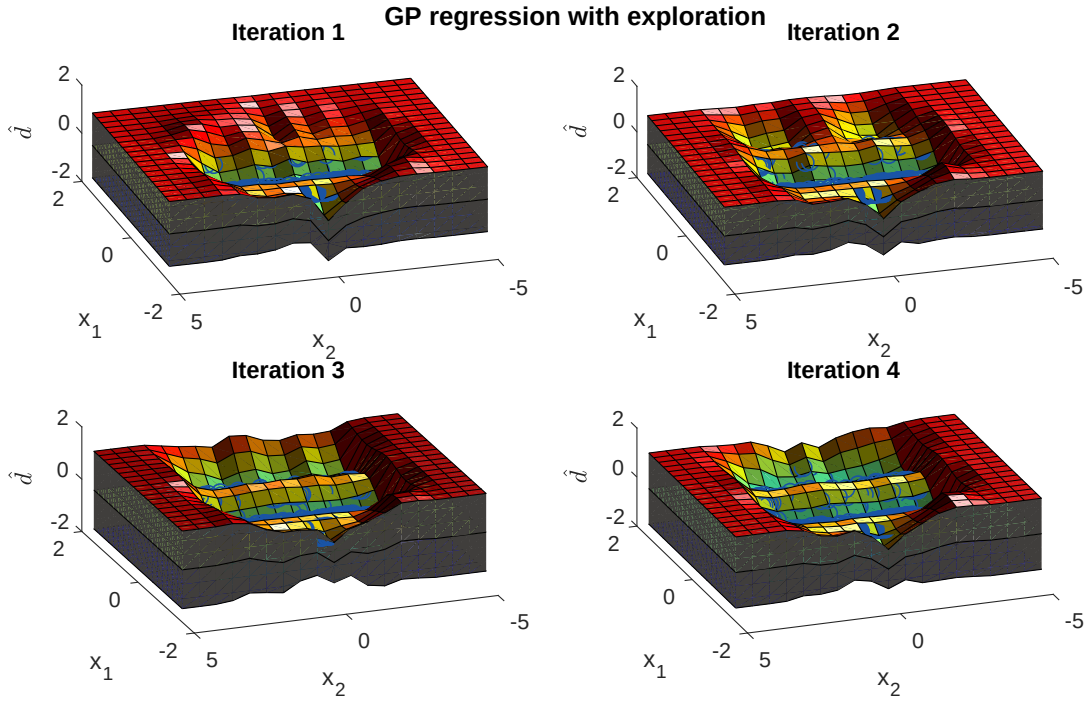

Figure 6.4: Gaussian Process regression in four iterations with incremental Q-learning.

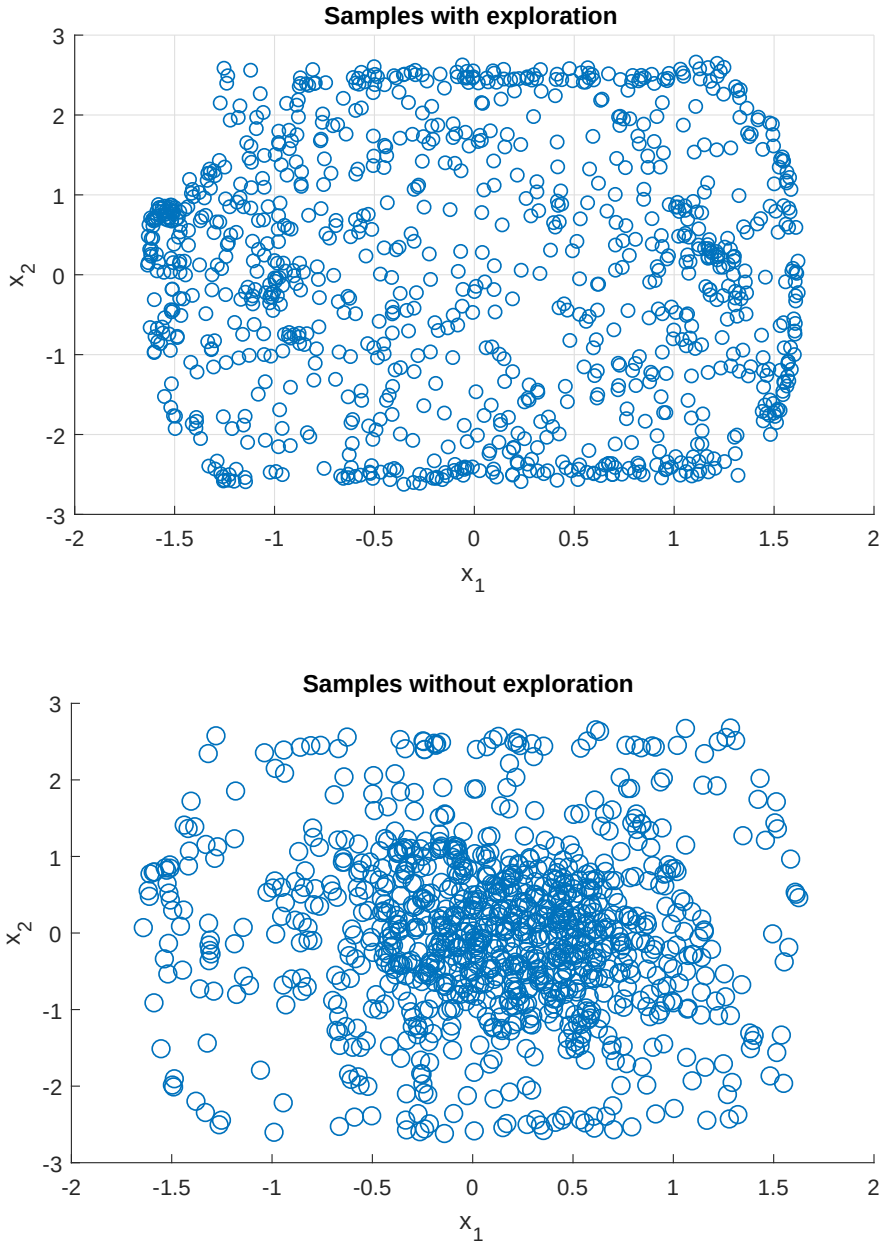

Figure 6.5: Random sampling with and without incremental Q-learning. In the upper figure, the spread of the samples is clearly better, whereas in the lower figure most samples are concentrated around the origin.

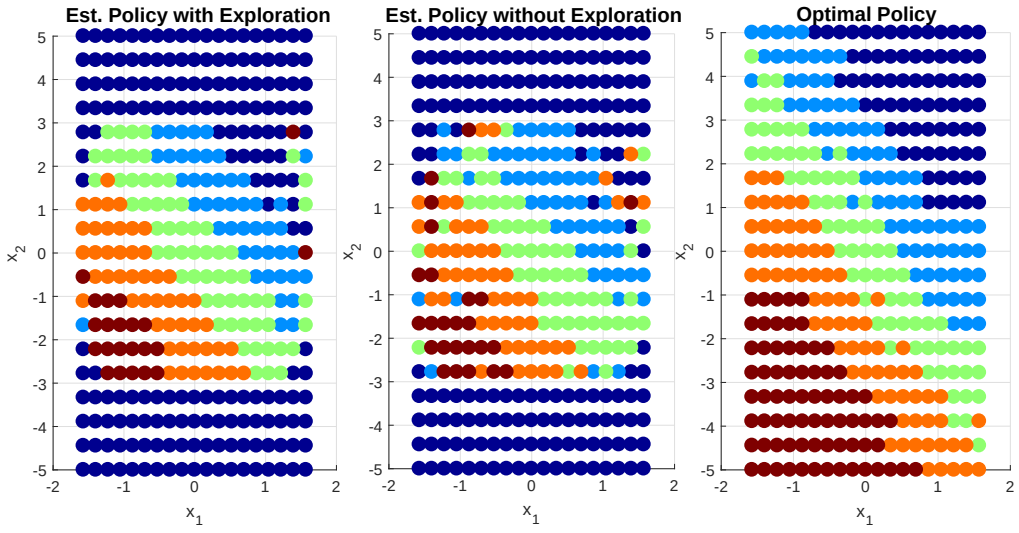

Figure 6.6: Policy estimation with and without exploration. After the same number of learning iterations, the algorithm with exploration achieves considerably better results above all at the edges of the safe set.

Chapter 7

Conclusions

- it works!
- difficult to combine different methods
- alternative to Q-learning?
- batch-method for GP suboptimal
- better way to share information between methods (e.g. exploit all information from HJI and GP)

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 *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 1424–1431. IEEE, 2014.

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

[3] M. G. Azar, R. Munos, M. Ghavamzadeh, H. J. Kappen, et al. Speedy q-learning. In *NIPS*, pages 2411–2419, 2011.

[4] K. Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

[5] E. Even-Dar and Y. Mansour. Convergence of optimistic and incremental q-learning. In *NIPS*, pages 1499–1506, 2001.

[6] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201, 1994.

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

[8] I. M. Mitchell. A toolbox of level set methods. *Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada, http://www. cs. ubc. ca/˜ mitchell/ToolboxLS/toolboxLS. pdf, Tech. Rep. TR-2004-09*, 2004.

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

[10] C. E. Rasmussen. Gaussian processes for machine learning. 2006.

[11] C. E. Rasmussen and H. Nickisch. Gaussian processes for machine learning (gpml) toolbox. *J. Mach. Learn. Res.*, 11:3011–3015, Dec. 2010.

[12] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888. ACM, 2006.

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

[14] J. N. Tsitsiklis, B. Van Roy, et al. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

[15] L.-X. Wang. Stable adaptive fuzzy control of nonlinear systems. *IEEE Transactions on fuzzy systems*, 1(2):146–155, 1993.

[16] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.