Technische Universität Dresden

Fakultät Elektrotechnik und Informationstechnik

Institut für Regelungs- und Steuerungstheorie

Diplomarbeit

Bestärkendes Lernen zur Steuerung und Regelung nichtlinearer dynamischer Systeme

vorgelegt von: Max Pritzkoleit

geboren am: 4. September 1991 in Arolsen, jetzt Bad Arolsen

zum Erlangen des akademischen Grades

Diplomingenieur

(Dipl.-Ing.)

Betreuer:	Dr.-Ing. C. Knoll
Verantwortlicher Hochschullehrer:	Prof. Dr.-Ing. habil. Dipl.-Math. K. Röbenack
Tag der Einreichung:	19.06.2019

Technische Universität Dresden
Fakultät Elektrotechnik und Informationstechnik
Institut für Regelungs- und Steuerungstheorie
Prof. Dr.-Ing. habil. K. Röbenack

RST
Institut für
Regelungs- und Steuerungstheorie

Aufgabenstellung zur Diplomarbeit
für Herrn cand. ing. Max Pritzkoleit

Bestärkendes Lernen zur Steuerung und Regelung nichtlinearer dynamischer Systeme

Das sog. „bestärkende Lernen“ (engl. „reinforcement learning“) ist eine Teildisziplin des maschinellen Lernens. Aus regelungstheoretischer Sicht kann dieser Ansatz als adaptive Optimalsteuerung interpretiert werden. Eine zentrale Rolle spielen dabei die Belohnungs-, Wert- und Strategie-Funktion (reward-, value-, policy-function), welche mit den inkrementellen Kosten, den minimalen Restkosten und dem optimalen Eingang korrespondieren. In den meisten Fällen sind diese Funktionen nicht bekannt und müssen „erlernt“, d. h. geeignet approximiert werden. Mögliche Regressionsmethoden dafür basieren auf künstlichen neuronalen Netzen (KNN) oder Gausschen Prozessen (GP).
Ziel der Diplomarbeit ist der Entwurf von Regelungseinrichtungen für nichtlineare dynamische Systeme verschiedener Komplexität auf Basis des bestärkenden Lernens. Neben der Stabilisierung einer Ruhelagen ist die Überführung zwischen vorgegebenen Ruhelagen interessant, insbesondere, wenn Methoden der linearen Regelungstheorie dafür unzulänglich sind. Weiterhin ist von Interesse, wie Prozesskenntnisse und etablierte Entwurfsverfahren im Lernprozess berücksichtigt werden können.

Im Einzelnen ergeben sich folgende Teilaufgaben:

- Literaturrecherche zu regelungstechnischen Anwendungen des bestärkenden Lernens und Reproduktion ausgewählter Ergebnisse
- Untersuchung verschiedener Approximationsmethoden (z. B. GP, KNN)
- Anwendung auf neue Beispielsysteme (thermodynamische Gebäudemodelle, unteraktuierte Systeme)
- Optional: Anwendung auf Versuchsstand (z. B. Schwebekörper)
- Dokumentation der Ergebnisse.

Bearbeitungszeitraum:	01. 11. 2018 – 11. 04. 2019
Prüfer :	Prof. Dr.-Ing. habil. Dipl.-Math. K. Röbenack, Dr.-Ing. J. Winkler
Betreuer:	Dr.-Ing. C. Knoll

Prof. Dr.-Ing. Bernet Vors. des Prüfungsausschusses	Prof. Dr.-Ing. habil. Dipl.-Math. K. Röbenack Verantwortlicher Hochschullehrer

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage dem Prüfungsausschuss der Fakultät Elektrotechnik und Informationstechnik eingereichte Diplomarbeit zum Thema

Bestärkendes Lernen zur Steuerung und Regelung nichtlinearer dynamischer Systeme

selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen sind, wurden als solche kenntlich gemacht.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistungen von folgenden Personen erhalten:

Dr.-Ing. Carsten Knoll

Weitere Personen waren an der geistigen Herstellung der vorliegenden Arbeit nicht beteiligt.

Dresden, 19.06.2019 Max Pritzkoleit

Kurzfassung

In der vorliegenden Arbeit wird das bestärkende Lernen im Kontext der Steuerung und Regelung nichtlinearer dynamischer Systeme untersucht. Es werden zunächst die Grundlagen der stochastischen Optimalsteuerung sowie des maschinellen Lernens, die für die Betrachtungen dieser Arbeit relevant sind, erläutert. Anschließend werden die Methoden des bestärkenden Lernens im Kontext der datenbasierten Steuerung und Regelung dargelegt, um anschließend auf drei Methoden des tiefen bestärkenden Lernens näher einzugehen. Der Algorithmus Deep-Deterministic-Policy-Gradient (DDPG) wird zum Gegenstand intensiver Untersuchungen an verschiedenen mechanischen Beispielsystemen. Weiterhin erfolgt der Vergleich mit einem klassischen Ansatz, bei dem die zu bewältigenden Steuerungsaufgaben mit einer modellbasierten Trajektorienberechnung, die auf dem iterativen linear-quadratischen Regler (iLQR) basiert, gelöst werden. Mit dem iLQR können zwar alle Steuerungsaufgaben erfolgreich bewältigt werden, aber für neue Anfangswerte muss das Problem erneut gelöst werden. Bei DDPG hingegen wird ein Regler erlernt, der das zu steuernde dynamische System – aus nahezu beliebigen Anfangswerten – in den gewünschten Zustand überführt. Nachteilig ist jedoch, dass der Algorithmus sich auf hochgradig nichtlineare Systeme bisher nicht anwenden lässt und eine geringe Dateneffizienz aufweist.

Abstract

In this thesis, the application of reinforcement learning for the control of nonlinear dynamic systems is researched. At first, the relevant principles of stochastic optimal control and machine learning are layed out. Afterwards, reinforcement learning is embedded in the context of optimal control. Three methods of deep reinforcement learning are analyzed. A particular algorithm, namely DDPG, is chosen for further studies on a variety of mechanical systems. Furthermore, the reinforcement learning approach is compared to a model-based trajectory optimization method, called iterative linear-quadratic regulator (iLQR). All control problems can be successfully solved with the trajectory optimization approach, but for new initial conditions, the problem has to be solved again. In contrast, with DDPG a *global* feedback controller is learned, that can drive the controlled system in the desired state. Disadvantageous is the poor data efficiency and the lack of applicability to highly nonlinear systems.

Inhaltsverzeichnis

Abkürzungsverzeichnis

BP	Backpropagation
CNN	Faltungsnetzwerk (engl. convolutional neural network)
DDP	differenzielle dynamische Programmierung
DDPG	Deep-Deterministic-Policy-Gradient
DP	dynamische Programmierung
DPG	Deterministic-Policy-Gradient
DQN	Deep-Q-Network
iLQR	iterativer linear-quadratischer Regler
KNN	künstliches neuronales Netzwerk
LQR	linear-quadratischer Regler
MAE	mittlerer absoluter Fehler (engl. mean absolute error)
MEP	Markow-Entscheidungsproblem
MLP	mehrschichtiges Perzeptron (engl. multilayer perceptron)
MPC	modellprädiktive Regelung (engl. model predictive control)
MSBE	Mean-Squared-Bellman-Error
MSE	mittlerer quadratischer Fehler (engl. mean squared error)
NFQ	Neural-Fitted-Q-Iteration
NMPC	nichtlineare modellprädiktive Regelung (engl. nonlinear model predictive control)
PG	Policy-Gradient
PILCO	Probabilistic-Inference-for-Learning-Control
ReLU	Rectified-Linear-Unit
RNN	rekurrentes neuronales Netzwerk (engl. recurrent neural network)

SGD stochastisches Gradientenabstiegsverfahren (engl. stochastic gradient descent)

SPG Stochastic-Policy-Gradient

TD Temporal-Difference

Symbole und Notation

a	Skalar
$\boldsymbol{a}$	Vektor $(a_0, \ldots, a_n)^{\mathrm{T}} \in \mathbb{R}^n$
$\dot{a}, \dot{\boldsymbol{a}}$	Zeitableitung
A	Matrix
$\boldsymbol{A}_{[1:n]}$	Folge von n Vektoren $\{\boldsymbol{a}_1, ..., \boldsymbol{a}_n\}$
$\mathbb{R}$	menge der reellen Zahlen
$\mathbb{R}^{\geq 0}$	Menge der nichtnegativen reellen Zahlen
$\mathbb{Z}$	Menge der ganzen Zahlen
$x \sim p(x)$	Zufallsvariable x ausgewählt aus der Verteilung $p(x)$
$\mathbb{E}_{x\sim p(x)}\{x\}$	Erwartungswert der Zufallsvariable x, $\mathbb{E}_{x\sim p(x)}\{x\} = \sum_x p(x)x$ bzw. $\mathbb{E}_{x\sim p(x)}\{x\} = \int p(x)x\mathrm{d}x$
$\mathcal{N}(\mu, \sigma)$	Normalverteilung mit Mittelwert μ und Standardabweichung σ
$\mathcal{U}(a, b)$	Gleichverteilung mit der Wahrscheinlichkeit $\frac{1}{b-a}$ im Intervall $[a, b]$
$^{[\cdot]}$	hochgestellter Index, der eine Schicht eines KNN bezeichnet
$^{(\cdot)}$	hochgestellter Index, der einen Datenpunkt bezeichnet
N	Zeithorizont
k	diskreter Zeitindex
$\boldsymbol{x}_k$	Zustand zum Zeitpunkt k
$\boldsymbol{x}_*$	Ruhelage
$\boldsymbol{u}_k$	Eingang oder Steuerung zum Zeitpunkt k
$\boldsymbol{w}_k$	Störgröße zum Zeitpunkt k
$\boldsymbol{\tau}$	Trajektorie
$p(\boldsymbol{\tau})$	Wahrscheinlichkeit für das Auftreten der Trajektorie $\boldsymbol{\tau}$
$\mathbb{X}_k$	Zustandsraum
$\mathbb{X}^-$	Menge nichtzulässiger Zustände
$\mathbb{X}^+$	Menge von Zielzuständen
$\mathbb{U}_k$	Eingangssraum

$\mathbb{U}(\boldsymbol{x}_k)$	Eingangsraum in Abhängigkeit von $\boldsymbol{x}_k$
$\mathbb{W}_k$	Raum der Störgröße $\boldsymbol{w}_k$
Π	Menge der zulässigen Rückführungen
$f_d(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k)$	zeitdiskrete stochastische nichtlineare Systemdynamik
$f_d(\boldsymbol{x}_k, \boldsymbol{u}_k)$	zeitdiskrete deterministische nichtlineare Systemdynamik
$f_c(\boldsymbol{x}_k, \boldsymbol{u}_k)$	zeitkontinuierliche deterministische nichtlineare Systemdynamik
$p(\boldsymbol{x}_{k+1}\|\boldsymbol{x}_k, \boldsymbol{u}_k)$	Wahrscheinlichkeit für $\boldsymbol{x}_{k+1}$ ausgehend von $\boldsymbol{x}_k$ bei Anwenden der Steuerung $\boldsymbol{u}_k$
π	Rückführung
π^*	optimale Rückführung
$\mu_k(\boldsymbol{x}_k)$	deterministische Rückführung
$\pi(\boldsymbol{u}_k\|\boldsymbol{x}_k)$	stochastische Rückführung
$\pi_\varepsilon(\boldsymbol{u}_k\|\boldsymbol{x}_k)$	*ε-greedy* Rückführung
$\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)$	parametrierte deterministische Rückführung
$\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k\|\boldsymbol{x}_k)$	parametrierte stochastische Rückführung
α	Lernrate bei Temporal-Difference (TD) Lernen und Q-learning
β	Schrittweite bei Gradientenabstiegsverfahren
$\boldsymbol{\theta}, \boldsymbol{\phi}$	Parmetervektoren von Funktionsapproximatoren
ε	Wahrscheinlichkeit dafür, eine zufällige Steuerung in einer *ε-greedy* Rückführung zu wählen
γ	Diskontierungsfaktor
τ	Target-Update-Rate bei DDPG
c_k	inkrementelle Kosten
c_N	Endkosten
J	Gesamtkosten
V^*	optimale Restkosten
V^π	Restkosten einer Rückführung π
Q^π	Bewertungsfunktion einer Rückführung π
$\hat{V}^\pi$	tabulare Repräsentation der Restkosten einer Rückführung π
$\hat{Q}^\pi$	tabulare Repräsentation der Bewertungsfunktion einer Rückführung π
$\hat{V}^\pi_{\boldsymbol{\phi}}$	Funktionsapproximation der Restkosten einer Rückführung π
$\hat{Q}^\pi_{\boldsymbol{\phi}}$	Funktionsapproximation der Bewertungsfunktion einer Rückführung π
$\mathcal{D}$	Datensatz
$\mathcal{B}$	Minibatch eines Datensatzes $\mathcal{D}$

Bemerkung zur Notation In der vorliegenden Arbeit wird entgegen der im deutschen Sprachraum üblichen Schreibweise der Punkt „.“ als Dezimaltrennzeichen verwendet. Hintergrund ist die Konsistenz zu automatisch generierten Achsenbeschriftungen in Abbildungen und die Eindeutigkeit bei der Angabe von Vektoren und Mengen, bei denen das Komma das Trennzeichen zwischen Komponenten bzw. Elementen darstellt.

Kapitel 1

Einführung

In den letzten Jahren konnten bemerkenswerte Fortschritte auf dem Gebiet des maschinellen Lernens erreicht werden. Besonders in der Sprach- und Bildverarbeitung [72, 31, 18] sowie der künstlichen Intelligenz [60, 43] und Robotik [53, 34] hat sich dieser Fortschritt bemerkbar gemacht. Dies wird zum Einen durch die Verfügbarkeit großer Datenmengen und entsprechender Rechenleistung ermöglicht, zum Anderen hat die Entwicklung von entsprechenden freien Software-Bibliotheken [47, 38, 10] für (tiefes)[1] maschinelles Lernen (engl. deep learning) diesen Prozess beschleunigt.

Eine der Schwierigkeiten des maschinellen Lernens bestand in der Merkmalsextraktion (engl. feature extraction), auf welche beim tiefen maschinellen Lernen größtenteils verzichtet werden kann. Durch die Anwendung des tiefen maschinellen Lernens ist es bspw. möglich, Motorsteuersignale direkt aus Pixeldaten eines Bildverarbeitungssystems zu generieren, um die Manipulationsaufgabe eines Industrieroboters zu bewältigen [33].

Ein weiterer Faktor, der die Popularität des (tiefen) maschinellen Lernens beflügelt, ist die Tatsache, dass wissenschaftliche Meilensteine auf dem Gebiet der künstlichen Intelligenz ein besonderes Maß an Aufmerksamkeit in der öffentlichen Wahrnehmung erregen [71, 58, 15].

Es wird auf eine lange Tradition von Ereignissen zurückgeblickt, bei denen bisher ausschließlich dem Menschen vorbehaltene kognitive Leistungen durch intelligente Systeme übertroffen wurden.

Im Jahr 1997 wurde der Schachweltmeister Garri Kasparow von dem von *IBM* entwickelten Programm *Deep Blue* in einem Spiel aus sechs Partien geschlagen [9]. Fast 20 Jahre später im Jahr 2016 wurde ein ähnliches Ergebnis für das traditionelle chinesische Spiel Go erzielt, bei dem das von *Google DeepMind* entwickelte Programm *AlphaGo* Lee Seedol, einen weltweit führenden Go-Spieler, in einer Partie aus fünf Spielen zur Aufgabe zwang [60].

[1] *Tief* bezieht sich auf die Topologie der verwendeten KNN.

Im Jahr 2011 konnte die künstliche Intelligenz *IBM Watson* in dem US-amerikanischen Fernsehquiz *Jeopardy* gegen seine menschlichen Gegner triumphieren [37]. Bei *Jeopardy* wird den Kandidaten eine Antwort gegeben, zu der sie die entsprechende Frage formulieren müssen.

Diesen Programmen ist gemein, dass sie das bestärkende Lernen nutzen.

1.1 Bestärkendes Lernen

Das bestärkende Lernen (engl. reinforcement learning) wird neben dem überwachten (engl. supervised) und unüberwachten (engl. unsupervised) Lernen als dritte Teildisziplin des maschinellen Lernens angesehen und beschäftigt sich mit Problemen der sequenziellen Entscheidungsfindung. Es ist theoretisch und historisch mit der optimalen Steuerung eng verzahnt und wird auch als direkte adaptive Optimalsteuerung interpretiert [67].

Die aktuellen Fortschritte des (tiefen) bestärkenden Lernens sowie der Zusammenhang mit der optimalen Steuerung machen die Untersuchung aus Sicht der Regelungs- und Steuerungstheorie und werfen somit zwei Fragen auf: Inwieweit eignen sich die Methoden des bestärkenden Lernens für die Steuerung und Regelung nichtlinear dynamischer Systeme und wie schlagen sie sich im Vergleich zu klassischen Methoden der Regelungs- und Steuerungstheorie?

Ein weiterer Faktor der das bestärkende Lernen für die Regelungs- und Steuerungstheorie interessant macht, sind die steigenden Anforderungen, die an Regelungseinrichtungen – auch bedingt durch neue Anwendungen – gestellt werden. So sollen Roboter künftig komplexe Manipulationsaufgaben lösen und sich adaptiv an neue Situationen anpassen. Es scheint zudem sinnvoll – mit Bezug auf die Entwicklungen im Rahmen von der sog. *Industrie 4.0* – vorhandene Daten zu nutzen.

Viele Ergebnisse des bestärkenden Lernens beruhen lediglich auf Simulationsstudien, da sich durch die geringe Dateneffizienz vieler Methoden eine Anwendung auf realen Systemen schwierig gestaltet. Dennoch wurde das bestärkende Lernen schon erfolgreich auf reale regelungstechnische Problemstellungen angewandt [13, 22, 27, 55].

1.2 Aufbau der Arbeit

Die vorliegende Arbeit befasst sich mit dem bestärkenden Lernen als Methode zur Steuerung und Regelung nichtlinearer dynamischer Systeme.

In Kapitel 2 werden die theoretischen Grundlagen der optimalen Steuerung sowie des maschinellen Lernens beschrieben. Es werden grundlegende Konzepte herausgearbeitet, die für das Verständnis der weiterführenden Kapitel nötig sind.

In Kapitel 3 und Kapitel 4 werden die untersuchten Methoden des (tiefen) bestärkenden Lernens erläutert. Ein besonderes Augenmerk wurde darauf gelegt, das bestärkende Lernen schlüssig in den Kontext der optimalen Steuerung einzubetten und so den Zusammenhang mit dieser zu verdeutlichen.

In Kapitel 5 wird anschließend der iterative linear-quadratische Regler (iLQR) – ein fortgeschrittenes Verfahren zur Steuerung und Regelung nichtlinearer dynamischer Systeme – als Referenzmethode eingeführt.

In Kapitel 6 folgt ein ausführlicher Vergleich der untersuchten Methoden an verschiedenen mechanischen Beispielsystemen sowie eine Auswertung der Ergebnisse. Abschließend werden in Kapitel 7 die Ergebnisse der Arbeit zusammengefasst und mögliche aufbauende Untersuchungsgegenstände dargelegt.

Kapitel 2

Theoretische Grundlagen

Die Basis für die Betrachtungen der vorliegenden Arbeit sind nichtlineare dynamische Systeme mit diskreter Zeitentwicklung

$$\boldsymbol{x}_{k+1} = f_d(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k), \quad k = 0, ..., N-1, \tag{2.1}$$

mit diskretem Zeitindex k, Zustand $\boldsymbol{x}_k \in \mathbb{X}_k \subset \mathbb{R}^n$, Eingang oder Steuerung $\boldsymbol{u}_k \in \mathbb{U}(\boldsymbol{x}_k) \subset \mathbb{U}_k \subset \mathbb{R}^m$, sowie einer Störgröße $\boldsymbol{w}_k \in \mathbb{W}_k \subset \mathbb{R}^d$ zum Zeitpunkt k. Die Horizontlänge N bestimmt, wie oft eine Steuerung angewandt wird. Die Abbildung $f_d : \mathbb{X}_k \times \mathbb{U}_k \times \mathbb{W}_k \to \mathbb{X}_{k+1}$ beschreibt, wie sich der Systemzustand $\boldsymbol{x}$ in der Zeit entwickelt [4, S. 2ff].

Der Eingang $\boldsymbol{u}_k$ ist beschränkt auf eine nichtleere Teilmenge $\mathbb{U}(\boldsymbol{x}_k) \subset \mathbb{U}_k$, die von $\boldsymbol{x}_k$ abhängt. Die Folge $\boldsymbol{w}_0, \boldsymbol{w}_1, ..., \boldsymbol{w}_{N-1}$ wird als unkorreliert angenommen.

Alternativ kann (2.1) auch als bedingte Verteilung

$$\boldsymbol{x}_{k+1} = \boldsymbol{w}_k, \tag{2.2}$$

mit

$$\boldsymbol{w}_k \sim p(\cdot|\boldsymbol{x}_k, \boldsymbol{u}_k), \quad k = 0, 1, ..., N-1, \tag{2.3}$$

dargestellt werden, welche die Wahrscheinlichkeit für eine Transition vom Zustand $\boldsymbol{x}_k$ in den Folgezustand $\boldsymbol{x}_{k+1}$ bei Anwenden der Steuerung $\boldsymbol{u}_k$ angibt [4, S. 13]. Es wird vorausgesetzt, dass der Gradient $\nabla_{\boldsymbol{u}_k} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)$ existiert und so wie (2.3) stetig in $\boldsymbol{x}_k$ und $\boldsymbol{u}_k$ ist.

Bemerkung 2.1. *Dass an dieser Stelle zwei Ausdrücke (2.1) und (2.3) für die Systemdynamik gegeben sind, hängt damit zusammen, dass je nach Problemstellung eine der beiden Darstellungen zu bevorzugen ist, um mathematisch konsistente und übersichtliche Ausdrücke zu erhalten. Im regelungstechnischen Kontext wird (2.1) gegenüber (2.3) bevorzugt, wohingegen (2.3) im Kontext des bestärkenden Lernens breite Verwendung findet.*

Das System (2.1) erfüllt per Konstruktion die Markow-Eigenschaft, welche besagt, dass $\boldsymbol{x}_{k+1}$ nur von Größen des vorherigen Zeitschritts k abhängt [59]:

$$p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k) = p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_0, \boldsymbol{u}_0, ..., \boldsymbol{x}_k, \boldsymbol{u}_k), \quad k = 0, ..., N-1. \tag{2.4}$$

Die Trajektorie

$$\boldsymbol{\tau} := \{\boldsymbol{X}_{[0:N-1]}, \boldsymbol{U}_{[0:N-1]}\} \tag{2.5}$$

des Systems (2.1), mit

$$\boldsymbol{X}_{[0:N-1]} := \{\boldsymbol{x}_0, ..., \boldsymbol{x}_{N-1}\} \tag{2.6}$$

und

$$\boldsymbol{U}_{[0:N-1]} := \{\boldsymbol{u}_0, ..., \boldsymbol{u}_{N-1}\} \tag{2.7}$$

beschreibt den zeitlichen Verlauf der Zustands- und Eingangsgrößen.

Die multivariate Verteilung

$$p(\boldsymbol{\tau}) = p(\boldsymbol{x}_0) \prod_{k=0}^{N-1} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.8}$$

gibt die Wahrscheinlichkeit für das Auftreten einer Trajektorie $\boldsymbol{\tau}$ an [32].

2.1 Stochastische Optimalsteuerung

Im Folgenden werden die notwendigen Grundlagen der stochastischen Optimalsteuerung für die Betrachtungen dieser Arbeit erläutert. Der **deterministische Fall** ist gegeben indem (2.1) durch das deterministische Äquivalent

$$\boldsymbol{x}_{k+1} = f_d(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad k = 0, ..., N-1 \tag{2.9}$$

ersetzt wird. Dadurch werden die Ausdrücke der folgenden Betrachtung ebenfalls deterministisch. Die Bildung des Erwartungswertes geht dann in einen Ausdruck mit Gleichungsnebenbedingungen über.

2.1.1 Grundlagen der optimalen Steuerung

Das System (2.1) akkumuliert ausgehend von einer gegebenen Verteilung des Anfangszustands $p(\boldsymbol{x}_0)$ Kosten C, die durch eine additive Kostenfunktion $c_k : \mathbb{X}_k \times \mathbb{U}_k \to \mathbb{R}^{\geq 0}$

$$C(\boldsymbol{X}_{[0:N]}, \boldsymbol{U}_{[0:N-1]}) = c_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} c_k(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{2.10}$$

mit den Endkosten $c_N(\boldsymbol{x}_N)$ definiert sind.

Die zu erwartenden Restkosten für eine gegebene Eingangsfolge $\boldsymbol{U}_{[0:N-1]}$, sowie einen Anfangszustand $\boldsymbol{x}$ sind gegeben durch

$$J(\boldsymbol{x}, \boldsymbol{U}_{[0:N-1]}) = \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau})} \left\{ c_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) \middle| \boldsymbol{x}_0 = \boldsymbol{x} \right\}. \tag{2.11}$$

Der Erwartungswert wird hier auf (2.10) angewandt, um –hinsichtlich des stochastischen Charakters von (2.1) – einen sinnvoll zu optimierenden Ausdruck zu gewährleisten [4, S. 3].

Definition 2.1 (Rückführung)**.** *Eine Rückführung bzw. ein Rückführ- oder Regelgesetz (engl. control policy)*[1] *ist eine Folge von Funktionen*

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{2.12}$$

wobei μ_k den Zustand $\boldsymbol{x}_k$ auf eine Steuerung

$$\boldsymbol{u}_k = \mu_k(\boldsymbol{x}_k) \tag{2.13}$$

abbildet. Die Rückführung ist zulässig, wenn $\mu_k(\boldsymbol{x}_k) \in \mathbb{U}(\boldsymbol{x}_k)$ für alle $\boldsymbol{x}_k \in \mathbb{X}_k$ und k gilt [4].

Weiterhin kann π auch stochastischen Charakter haben und als bedingte Verteilung über den Zustand

$$\boldsymbol{u}_k \sim \pi(\boldsymbol{u}_k|\boldsymbol{x}_k) \tag{2.14}$$

gegeben sein. Die deterministische Rückführung (2.13) kann in der Form (2.14) dargestellt werden:

$$\pi(\boldsymbol{u}_k|\boldsymbol{x}_k) = \begin{cases} 1, & \text{wenn} \quad \boldsymbol{u}_k = \mu_k(\boldsymbol{x}_k), \\ 0, & \text{sonst.} \end{cases} \tag{2.15}$$

Für die Rückführung π wird eine Verteilung $p_\pi(\boldsymbol{\tau})$ eingeführt, die die Wahrscheinlichkeit für das Auftreten einer Trajektorie $\boldsymbol{\tau}$ angibt [49]:

$$p_\pi(\boldsymbol{\tau}) = p(\boldsymbol{x}_0) \prod_{k=0}^{N-1} \pi(\boldsymbol{u}_k|\boldsymbol{x}_k) p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k). \tag{2.16}$$

Definition 2.2 (Bewertungsfunktion Q^π)**.** *Die Bewertungsfunktion (engl. Q-function, action-value function) Q^π für eine gegebene Rückführung π gibt die zu erwartenden Restkosten für ein gegebenes Zustands-Eingangspaar $(\boldsymbol{x}, \boldsymbol{u})$ an [49]:*

$$Q^\pi(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{E}_{\boldsymbol{\tau}\sim p_\pi(\boldsymbol{\tau})} \left\{ c_N(\boldsymbol{x}_N) + \sum_{i=k}^{N-1} c_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \middle| \boldsymbol{x}_k = \boldsymbol{x}, \boldsymbol{u}_k = \boldsymbol{u} \right\} \tag{2.17}$$

[1] In der englischsprachigen Literatur wird oft auch nur von *policy* gesprochen.

Definition 2.3 (Restkosten V^π)**.** *Die zu erwartenden Restkosten (engl. value function) V^π für eine gegebene Rückführung π ausgehend von einem Anfangszustand $\boldsymbol{x}$ sind durch*

$$V^\pi(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{u}_k \sim \pi(\cdot|\boldsymbol{x}_k)} \left\{ Q^\pi(\boldsymbol{x}_k, \boldsymbol{u}_k) \middle| \boldsymbol{x}_0 = \boldsymbol{x} \right\} \tag{2.18}$$

gegeben [49].

Das Problem der optimalen Steuerung

Das Problem der optimalen Steuerung wird auf zwei Weisen formuliert und entsprechend gelöst. Zur Lösung des Problems der optimalen Steuerung ist entweder

1. die optimale Eingangsfolge $\boldsymbol{U}^*_{[0:N-1]} = \{\boldsymbol{u}^*_0, ..., \boldsymbol{u}^*_{N-1}\}$ [19, S. 82]:

$$\begin{aligned}
\min_{\boldsymbol{U}_{[0:N-1]}} \quad & J(\boldsymbol{x}_0, \boldsymbol{U}_{[0:N-1]}) && \text{(2.19a)}\\
\text{u.B.v.} \quad & \boldsymbol{x}_{k+1} = f_d(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k), \quad k = 0, ..., N-1, && \text{(2.19b)}\\
& \boldsymbol{u}_k \in \mathbb{U}(\boldsymbol{x}_k), && \text{(2.19c)}\\
& \text{geg. } \boldsymbol{x}_0, && \text{(2.19d)}
\end{aligned}$$

   oder

2. die optimale Rückführung π^* zu bestimmen, welche die zu erwartenden Restkosten V_π minimiert:

$$V^*(\boldsymbol{x}_0) = \min_{\pi \in \Pi} V^\pi(\boldsymbol{x}_0), \tag{2.20}$$

   wobei π^* Element des Funktionenraumes der zulässigen Rückführungen Π ist. Die Minimierung führt auf die minimalen bzw. optimalen Restkosten (engl. optimal value function) [4, S. 14].

Modellprädiktive Regelung (engl. model predictive control) (MPC)

Die MPC ist eine fortgeschrittene Methode der Steuerungs- und Regelungstheorie, bei der in jedem Zeitschritt k das verkürzte Optimierungsproblem (2.19) über N_p Zeitschritte gelöst wird [20]. Mit N_p ist der Prädiktionshorizont bezeichnet. Der erste Wert $\boldsymbol{u}^*_0$ der bestimmten optimalen Eingangsfolge $\boldsymbol{U}^*_{[0:N_p-1]}$ wird auf das System (2.1) angewandt, woraufhin eine Zustandstransition erfolgt. Der Folgezustand wird als neuer Anfangswert genutzt, um erneut das Optimierungsproblem zu lösen. Im Gegensatz zur naiven Anwendung der Steuerfolge $\boldsymbol{U}^*_{[0:N_p-1]}$ wird so der Einfluss von Störgrößen und Modellunbestimmtheiten kompensiert.

Algorithmus 1 Modellprädiktive Regelung MPC

Anfangswert $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$
for $k = [0, ..., N-1]$ **do**
  $\boldsymbol{U}^*_{[0:N_p-1]}$ aus (2.19) mit $\boldsymbol{x}_0 := \boldsymbol{x}_k$ und Prädiktionshorizont N_p ▷ Optimierung.
  $\boldsymbol{u}_k := \boldsymbol{u}^*_0$
  $\boldsymbol{x}_{k+1} \sim p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)$
end for

2.1.2 Dynamische Programmierung (DP)

Die dynamische Programmierung (DP) basiert auf dem Optimalitätsprinzip von Bellman, welches besagt, dass sich jede optimale Lösung von (2.20) aus optimalen Teillösungen zusammensetzt. Die Grundidee der DP ist es, das Problem in Teilprobleme zu unterteilen, welche einfacher gelöst werden können und diese Teillösungen zur Gesamtlösung zusammenzusetzen. In der Informatik wird dieser Ansatz als Teile-und-herrsche-Verfahren bezeichnet [12, S. 28].

Satz 1 (Optimalitätsprinzip von Bellman)**.** *Sei* $\pi^* = \left\{\mu_0^*, \mu_1^*, ..., \mu_{N-1}^*\right\}$ *die optimale Rückführung für* (2.18)*. Betrachtet man das Teilproblem von* (2.11)*, bei dem ausgehend vom Zustand* $\boldsymbol{x}_i$ *die Restkosten*

$$J(\boldsymbol{x}, \boldsymbol{U}_{[i:N-1]}) = \mathbb{E}_{\boldsymbol{\tau} \sim p(\boldsymbol{\tau})} \left\{ c_N(\boldsymbol{x}_N) + \sum_{j=i}^{N-1} c_j(\boldsymbol{x}_j, \boldsymbol{u}_j) \middle| \boldsymbol{x}_i = \boldsymbol{x} \right\}$$

minimiert werden sollen, so ist die verkürzte Rückführung $\left\{\mu_i^*, \mu_{i+1}^*, ..., \mu_{N-1}^*\right\}$ *optimal für das Teilproblem [4, S. 18].*

Auf Basis von Satz 1 lässt sich der Algorithmus der dynamischen Programmierung herleiten, der in Satz 2 aufgeführt ist.

Satz 2 (Dynamische Programmierung (DP))**.** *Für jeden Anfangszustand* $\boldsymbol{x}_0$*, sind die minimalen Restkosten* $V^*(\boldsymbol{x}_0)$ *gleich den Restkosten* $V_0(\boldsymbol{x}_0)$ *des folgenden Algorithmus. Dieser berechnet ausgehend von* $k = N$ *bis zu* $k = 0$ *– für alle* $\boldsymbol{x}_k \in \mathbb{X}_k$ *– die minimalen Restkosten für eine gegebene Randbedingung* $c_N(\boldsymbol{x}_N)$*:*

$$V_N(\boldsymbol{x}_N) = c_N(\boldsymbol{x}_N) \tag{2.21a}$$

$$V_k(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathbb{U}(\boldsymbol{x}_k)} c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) + \mathbb{E}_{\boldsymbol{x}_{k+1} \sim p(\cdot|\boldsymbol{x}_k, \boldsymbol{u}_k)} \left\{ V_{k+1}(\boldsymbol{x}_{k+1}) \right\}, k = 0, 1, ..., N-1. \tag{2.21b}$$

Wenn weiterhin $\boldsymbol{u}_k^* := \mu_k^*(\boldsymbol{x}_k)$ *der Minimierer von* (2.21b) *für alle* $\boldsymbol{x}_k$ *und* k *ist, dann ist die Rückführung* $\pi^* = \left\{\mu_0^*, \mu_1^*, ..., \mu_{N-1}^*\right\}$ *optimal [4, S. 23].*

2.1.3 Markow-Entscheidungsproblem (MEP)

Definition 2.4 (MEP)**.** *Ein MEP ist ein Tupel* $\mathcal{M} = (\mathbb{X}, \mathbb{U}, \mathcal{T}, c, \gamma)$*. Dabei bezeichnet* $\mathbb{X}$ *einen Zustandsraum,* $\mathbb{U}$ *einen Eingangsraum,* $\mathcal{T}$ *einen Transitionsoperator,* c *ein Kostenfunktional und* γ *einen Diskontierungsfaktor [59].*

Ein MEP bildet einen sehr allgemein gehaltenen Rahmen für die Lösung von Problemen zur sequenziellen Entscheidungsfindung mit Unbestimmtheiten. Prinzipiell kann jede regelungstechnische Problemstellung als MEP formuliert werden.

Bemerkung 2.2 (Partiell beobachtbares MEP)**.** *Bei partiell beobachtbaren MEP wird davon ausgegangen, dass nur der Ausgang* $\boldsymbol{y}_k$ *bekannt ist und so nur eine indirekte Kenntnis über den Zustand* $\boldsymbol{x}_k$ *vorliegt. Der Zusammenhang von* $\boldsymbol{x}_k$ *und* $\boldsymbol{y}_k$ *wird durch die bedingte Verteilung*

$$h(\boldsymbol{y}_{k+1}|\boldsymbol{x}_{k+1}, \boldsymbol{u}_k) \tag{2.22}$$

definiert [59]. In diesem Fall kann die optimale Rückführung π^**, im Gegensatz zu der optimalen Rückführung eines MEP, auch stochastisch sein [62]. In dieser Arbeit werden nur MEP betrachtet, es wird also davon ausgegangen, dass der Zustand* $\boldsymbol{x}_k$ *bekannt ist. In der Regelungs- und Steuerungstheorie werden im Fall der partiell beobachtbaren MEP modellbasierte Schätzer (bspw. das Kalman-Filter) eingesetzt, um den Zustand* $\boldsymbol{x}_k$ *zu rekonstruieren.*

Lösung eines Markow-Entscheidungsproblem

Ein MEP ist gelöst, wenn die optimale Restkostenfunktion $V^*(\boldsymbol{x}_k)$ bzw. die optimale Bewertungsfunktion $Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k)$ bestimmt ist. Aus dieser lässt sich die optimale Rückführung π^* ableiten [59]:

$$\pi^*(\boldsymbol{u}_k|\boldsymbol{x}_k) = \begin{cases} 1, & \text{wenn} \quad \boldsymbol{u}_k = \arg\min_{\boldsymbol{u}_k \in \mathbb{U}(\boldsymbol{x}_k)} Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k), \\ 0 & \text{sonst.} \end{cases} \tag{2.23}$$

Alternativ kann π^* auch aus V^* berechnet werden:

$$\mu^*(\boldsymbol{x}_k) = \arg \min_{\boldsymbol{u}_k \in \mathbb{U}(\boldsymbol{x}_k)} \left(c(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \mathbb{E}_{\boldsymbol{x}_{k+1} \sim p(\cdot|\boldsymbol{x}_k, \boldsymbol{u}_k)} \{ V^*(\boldsymbol{x}_{k+1}) \} \right). \tag{2.24}$$

V^* und π^* können bspw. mit der in Abschnitt 2.1 beschriebenen DP bestimmt werden. Dazu muss jedoch der Transitionsoperator $\mathcal{T}$, also die Dynamik des Systems (2.1) bekannt sein.

Die Rückführung (2.23) wird als *greedy policy* – also gierige[2] Rückführung – bezeichnet, da sie Q minimiert. Die optimale Rückführung π^* ist somit deterministisch. Dass π im Kontext des bestärkenden Lernen als bedingte Verteilung über den Zustand $\pi(\boldsymbol{u}_k|\boldsymbol{x}_k)$ formuliert wird, stellt eine notwendige Verallgemeinerung dar. Die Rückführung π ist während des Lernvorgangs probabilistisch, um den Zustandsraum des Systems zu erkunden.

Eine häufig verwendete Rückführung ist die sog. *ε-greedy* Rückführung $\pi_\varepsilon(\boldsymbol{u}_k|\boldsymbol{x}_k)$, die mit der Wahrscheinlichkeit $(1-\varepsilon)$ die Rückführung (2.23) anwendet und mit der Wahrscheinlichkeit ε eine zufällige Steuerung $\boldsymbol{u}_k \in \mathbb{U}(\boldsymbol{x}_k)$ zurückgibt.

2.1.4 Exakte Lösung eines Markow-Entscheidungsproblems (MEP)

Für die exakte Lösung eines MEP muss V^* bzw. Q^* bestimmt werden. Dazu muss im Gegensatz zur approximativen Lösung, die bspw. mit den Methoden des bestärkenden Lernens erfolgen kann, die Systemdynamik (2.1) bekannt sein.

Werte-Iteration (engl. value iteration)

Unter Werte-Iteration versteht man die Berechnung von V^* durch die in Satz 2 beschriebene DP. Die Iteration wird für den gesamten Zustandsraum ausgehend von beliebig initialisierten Restkosten V_0 durchgeführt [66, S. 83]. Ist der Zustandsraum kontinuierlich, so muss er entsprechend diskretisiert werden.

Die Iterationsvorschrift in Satz 2 stellt eine Fixpunktiteration für V dar und konvergiert gegen V^* [59]. Wenn also $V_{k+1} = V_k$ gilt, ist die optimale Restkostenfunktion V^* gefunden.

Regler-Iteration (engl. policy iteration)

Bei der Regler-Iteration wird zunächst die Restkostenfunktion $V^\pi(\boldsymbol{x}_k)$ einer beliebig initialisierten Rückführung π bestimmt [66, S. 80]:

$$V^\pi(\boldsymbol{x}_k) = c_k(\boldsymbol{x}_k, \mu(\boldsymbol{x}_k)) + \gamma \mathbb{E}_{\boldsymbol{x}_{k+1}\sim p(\cdot|\boldsymbol{x}_k,\mu(\boldsymbol{x}_k))}\{V^\pi(\boldsymbol{x}_{k+1})\}, \quad \forall \boldsymbol{x}_k \in \mathbb{X}_k. \tag{2.25a}$$

Anschießend kann mit (2.24) eine neue Rückführung π' bestimmt werden:

$$\mu'(\boldsymbol{x}_k) := \arg\min_{\boldsymbol{u}_k \in \mathbb{U}(\boldsymbol{x}_k)} \left(c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \mathbb{E}_{\boldsymbol{x}_{k+1}\sim p(\cdot|\boldsymbol{x}_k,\boldsymbol{u}_k)}\{V^\pi(\boldsymbol{x}_{k+1})\} \right), \quad \forall \boldsymbol{x}_k \in \mathbb{X}_k. \tag{2.25b}$$

[2] Im Englischen wird von *gierig* gesprochen, da diese Rückführung die Belohnung maximiert.

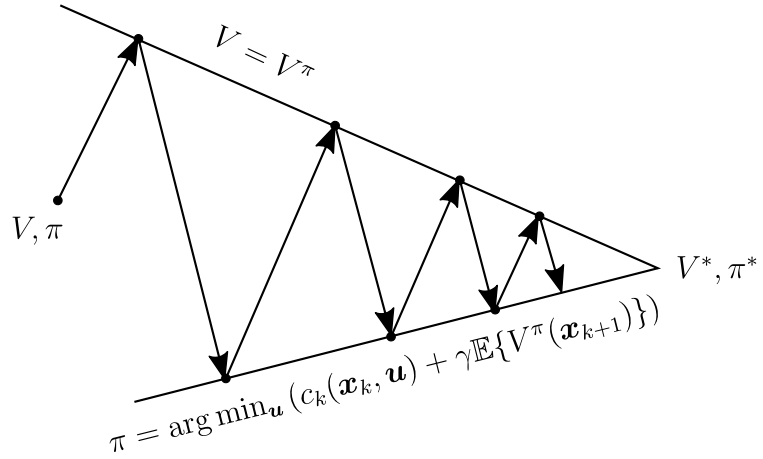

Abbildung 1 – Bei der Regler-Iteration [66, S. 87] wechselt der Algorithmus – ausgehend von initialen V und π – zwischen der Bestimmung von V^π und der Verbesserung von π.

Diese beiden Schritte werden wiederholt, bis $\pi = \pi'$ gilt. Der Algorithmus wechselt zwischen der Bestimmung von V^π und der Verbesserung von π. Dieser Vorgang ist in Abbildung 1 dargestellt. Auch die Regler-Iteration ist eine Fixpunktiteration und konvergiert gegen π^*. Ein Beweis ist bspw. in [52] aufgeführt.

Der Werte- und Regler-Iteration ist gemein, dass bei großen $\mathbb{X}$ die Auswertung von Satz 2 und (2.25) sehr rechenaufwendig sind. In Kapitel 3 werden deshalb Methoden zur approximativen Lösung aufgezeigt.

2.1.5 Zeitdiskreter stochastischer LQR mit endlichem Zeithorizont

Der LQR-Entwurf stellt eine bedeutende Anwendung der DP in der linearen Steuerungs- und Regelungstheorie dar. Der LQR eignet sich zur Stabilisierung von Ruhelagen nichtlinearer Systeme. Der Algorithmus wird an dieser Stelle in Vorbereitung auf Kapitel 5 detailliert hergeleitet.

Beim zeitdiskreten LQR-Entwurf wird die bezüglich eines quadratischen Kostenfunktionals optimale Rückführung π^* für ein lineares zeitinvariantes System mit diskreter Zeitentwicklung

$$\boldsymbol{x}_{k+1} = f_d(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k) = A\boldsymbol{x}_k + B\boldsymbol{u}_k + \boldsymbol{w}_k, \quad k = 0, 1, ..., N-1, \tag{2.26}$$

bestimmt. Das Paar (A, B) ist steuerbar [14, S. 616].

Die Störgröße $\boldsymbol{w}_k$ ist durch folgende Eigenschaften charakterisiert [7]:

$$\mathbb{E}\{\boldsymbol{w}_k\} = \boldsymbol{0}, \tag{2.27a}$$

$$\mathbb{E}\{\boldsymbol{w}_k \boldsymbol{w}_k^{\mathrm{T}}\} = W. \tag{2.27b}$$

Die Kosten sind gegeben durch

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}\left(\boldsymbol{x}_k^{\mathrm{T}} S \boldsymbol{x}_k + \boldsymbol{u}_k^{\mathrm{T}} R \boldsymbol{u}_k + 2\boldsymbol{x}_k^{\mathrm{T}} P \boldsymbol{u}_k\right), \tag{2.28a}$$

$$c_N(\boldsymbol{x}_N) = \frac{1}{2}\boldsymbol{x}_N^{\mathrm{T}} S_N \boldsymbol{x}_N. \tag{2.28b}$$

Mit $S \geq 0, S_N \geq 0$ positiv semi- und $R > 0$ positiv definiten, symmetrischen Gewichtsmatrizen entsprechender Dimension, die in der Regel Diagonalstruktur aufweisen. Die Gewichtsmatrix P erfüllt die Bedingung [14, S. 615]:

$$S - PR^{-1}P^T \geq 0. \tag{2.29}$$

Mit (2.26) und (2.10) stellt das LQR-Problem einen Spezialfall von (2.19) dar.

Um die optimale Rückführung π^* in der Form

$$\boldsymbol{u}_k^* = \mu^*(\boldsymbol{x}_k) = -K_k \boldsymbol{x}_k \tag{2.30}$$

zu bestimmen, können zwei Ansätze verfolgt werden:

1. Lösung von (2.19) mittels Satz 2 (DP) (s. Abschnitt 2.1.6),
2. Überführung von (2.19) auf ein lineares Optimierungsproblem [2] (hier nicht weiter beschrieben).

Der LQR-Entwurf stellt ein Verfahren zur Polplatzierung des geschlossenen Regelkreises bereit. Anstatt jedoch die Pole explizit vorzugeben, kann mittels der Gewichtsmatrizen das gewünschte Regelverhalten eingestellt werden.

2.1.6 Lösung des stochastischen LQR-Problems mittels dynamischer Programmierung (DP)

Ausgehend von den Endkosten $c_N(\boldsymbol{x}_N)$ wird eine Rückwärtsrechnung auf Basis von (2.21) bis zu $\boldsymbol{x}_0$ durchgeführt, bei der für jeden Zeitschritt k die optimale Rückführmatrix K_k durch Optimierung bestimmt wird:

$$V_N(\boldsymbol{x}_N) = \frac{1}{2}\boldsymbol{x}_N^{\mathrm{T}} S \boldsymbol{x}_N, \tag{2.31a}$$

$$V_{N-1}(\boldsymbol{x}_{N-1}) = \min_{\boldsymbol{u}_{N-1}} c_{N-1}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) + \mathbb{E}\{V_N(\boldsymbol{x}_N)\}. \tag{2.31b}$$

Das Einsetzen der Systemgleichung $\boldsymbol{x}_N = A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1} + \boldsymbol{w}_{N-1}$ liefert die Bewertungsfunktion:

$$Q_{N-1}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) = c_{N-1}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) + \mathbb{E}_{\boldsymbol{w}_{N-1}}\{V_N(A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1} + \boldsymbol{w}_{N-1})\}, \tag{2.32a}$$

$$= c_{N-1}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) \tag{2.32b}$$

$$+ \mathbb{E}_{\boldsymbol{w}_{N-1}}\left\{\frac{1}{2}(A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1} + \boldsymbol{w}_{N-1})^{\mathrm{T}} S_N (A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1} + \boldsymbol{w}_{N-1})\right\},$$

$$= c_{N-1}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) + \frac{1}{2}(A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1})^{\mathrm{T}} S_N (A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1}) \tag{2.32c}$$

$$+ \frac{1}{2}\mathbb{E}_{\boldsymbol{w}_{N-1}}\left\{\boldsymbol{w}_{N-1}^{\mathrm{T}} S_N \boldsymbol{w}_{N-1}\right\},$$

$$= \frac{1}{2}\left(\boldsymbol{x}_{N-1}^{\mathrm{T}} S \boldsymbol{x}_{N-1} + \boldsymbol{u}_{N-1}^{\mathrm{T}} R \boldsymbol{u}_{N-1} + 2\boldsymbol{x}_{N-1}^{\mathrm{T}} P \boldsymbol{u}_{N-1}\right) \tag{2.32d}$$

$$+ \frac{1}{2}(A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1})^{\mathrm{T}} S_N (A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1})$$

$$+ \frac{1}{2}\mathrm{Spur}(W S_N),$$

$$= \frac{1}{2}\Big(\boldsymbol{x}_{N-1}^{\mathrm{T}}(S + A^{\mathrm{T}} S_N A)\boldsymbol{x}_{N-1} + \boldsymbol{u}_{N-1}^{\mathrm{T}}(R + B^{\mathrm{T}} S_N B)\boldsymbol{u}_{N-1} \tag{2.32e}$$

$$+ 2\boldsymbol{x}_{N-1}^{\mathrm{T}}(A^{\mathrm{T}} S_N B + P)\boldsymbol{u}_{N-1} + \mathrm{Spur}(W S_N)\Big).$$

Aus $Q_{N-1}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1})$ kann mittels Minimierung der optimale Eingang bestimmt werden:

$$\boldsymbol{u}_{N-1}^{*} = \arg\min_{\boldsymbol{u}_{N-1}} Q_{N-1}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}). \tag{2.33}$$

Dazu wird die notwendige Bedingung für Optimalität verwendet [5, S. 13]:

$$\frac{\partial}{\partial \boldsymbol{u}_{N-1}} Q_{N-1}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}^{*}) \stackrel{!}{=} 0, \tag{2.34a}$$

$$\Leftrightarrow \quad \boldsymbol{u}_{N-1}^{*\mathrm{T}}(R + B^{\mathrm{T}} S_N B) + \boldsymbol{x}_{N-1}^{\mathrm{T}}(A^{\mathrm{T}} S_N B + P) = 0, \tag{2.34b}$$

$$\Leftrightarrow \quad (R + B^{\mathrm{T}} S_N B)^{\mathrm{T}} \boldsymbol{u}_{N-1} + (B^{\mathrm{T}} S_N A + P^{\mathrm{T}})\boldsymbol{x}_{N-1} = 0, \tag{2.34c}$$

$$\Leftrightarrow \quad \boldsymbol{u}_{N-1}^{*} = -K_{N-1}\boldsymbol{x}_{N-1}, \tag{2.34d}$$

mit

$$K_{N-1} := (R + B^{\mathrm{T}} S_N B)^{-1}(B^{\mathrm{T}} S_N A + P^{\mathrm{T}}). \tag{2.35}$$

Dass es sich dabei tatsächlich um den optimalen Eingang handelt, folgt aus der hinreichenden Bedingung für Optimalität [5, S. 13]:

$$\frac{\partial^2}{\partial^2 \boldsymbol{u}_{N-1}} Q_{N-1}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) > 0, \tag{2.36}$$

$$\Leftrightarrow \quad (R + B^{\mathrm{T}} S_N B) > 0. \tag{2.37}$$

Durch die Definitheitsanforderungen an die Gewichtsmatrizen R und S_N ist diese immer erfüllt. Somit ist das Problem konvex und bereits (2.34) hinreichend [5, S. 14].

Das Einsetzen der in (2.34) bestimmten Steuerung liefert die minimalen Restkosten für den Anfangszustand $\boldsymbol{x}_{N-1}$:

$$V_{N-1}(\boldsymbol{x}_{N-1}) = Q_{N-1}(\boldsymbol{x}_{N-1}, -K_{N-1}\boldsymbol{x}_{N-1}), \tag{2.38a}$$

$$= \frac{1}{2}\Big(\boldsymbol{x}_{N-1}^{\mathrm{T}}(S + A^{\mathrm{T}} S_N A)\boldsymbol{x}_{N-1} + \boldsymbol{x}_{N-1}^{\mathrm{T}} K_{N-1}^{\mathrm{T}}(R + B^{\mathrm{T}} S_N B) K_{N-1}\boldsymbol{x}_{N-1} \tag{2.38b}$$

$$- 2\boldsymbol{x}_{N-1}^{\mathrm{T}}(A^{\mathrm{T}} S_N B + P) K_{N-1}\boldsymbol{x}_{N-1} + \mathrm{Spur}(W S_N)\Big),$$

$$= \frac{1}{2}\boldsymbol{x}_{N-1}^{\mathrm{T}} \Big(S + A^{\mathrm{T}} S_N A + K_{N-1}^{\mathrm{T}}(R + B^{\mathrm{T}} S_N B) K_{N-1} - 2(A^{\mathrm{T}} S_N B + P) K_{N-1}\Big) \boldsymbol{x}_{N-1} \tag{2.38c}$$

$$+ \frac{1}{2}\mathrm{Spur}(W S_N),$$

$$= \frac{1}{2}\boldsymbol{x}_{N-1}^{\mathrm{T}} S_{N-1}\boldsymbol{x}_{N-1} + \frac{1}{2}\mathrm{Spur}(W S_N), \tag{2.38d}$$

mit

$$S_{N-1} := S + A^{\mathrm{T}} S_N A + K_{N-1}^{\mathrm{T}}(R + B^{\mathrm{T}} S_N B) K_{N-1} - 2(A^{\mathrm{T}} S_N B + P) K_{N-1}. \tag{2.39}$$

Diese quadratische Form von $V_{N-1}(\boldsymbol{x}_{N-1})$ ist äquivalent zu (2.31a). Setzt man den Algorithmus fort, kann man iterativ S_0 bestimmen und damit auch $V_0(\boldsymbol{x}_0)$. Für S_{N-1} aus (2.39) folgt durch Eliminierung von K_{N-1}:

$$S_{N-1} = S + A^{\mathrm{T}} S_N A + K_{N-1}^{\mathrm{T}}(R + B^{\mathrm{T}} S_N B) K_{N-1} - 2(A^{\mathrm{T}} S_N B + P) K_{N-1}, \tag{2.40a}$$

$$= S + A^{\mathrm{T}} S_N A + K_{N-1}^{\mathrm{T}} \underbrace{(R + B^{\mathrm{T}} S_N B) K_{N-1}}_{\overset{(2.35)}{=}(B^{\mathrm{T}} S_N A + P^{\mathrm{T}})} - 2(A^{\mathrm{T}} S_N B + P) K_{N-1}, \tag{2.40b}$$

$$= S + A^{\mathrm{T}} S_N A + K_{N-1}^{\mathrm{T}}(B^{\mathrm{T}} S_N A + P^{\mathrm{T}}) - 2(A^{\mathrm{T}} S_N B + P) K_{N-1}, \tag{2.40c}$$

$$\overset{(2.35)}{=} S + A^{\mathrm{T}} S_N A + \Big((R + B^{\mathrm{T}} S_N B)^{-1}(B^{\mathrm{T}} S_N A + P^{\mathrm{T}})\Big)^{\mathrm{T}} (B^{\mathrm{T}} S_N A + P^{\mathrm{T}}) \tag{2.40d}$$

$$- 2(A^{\mathrm{T}} S_N B + P)(R + B^{\mathrm{T}} S_N B)^{-1}(B^{\mathrm{T}} S_N A + P^{\mathrm{T}}),$$

$$= S + A^{\mathrm{T}} S_N A + (A^{\mathrm{T}} S_N B + P)(R + B^{\mathrm{T}} S_N B)^{-1}(B^{\mathrm{T}} S_N A + P^{\mathrm{T}}). \tag{2.40e}$$

Die Matrizen K_k, mit $k = N-1, ..., 0$ resultieren aus der folgenden Iterationsvorschrift:

$$K_k = (R + B^{\mathrm{T}} S_{k+1} B)^{-1} (B^{\mathrm{T}} S_{k+1} A + P^{\mathrm{T}}), \tag{2.41}$$

wobei S_{k+1} beginnend mit S_N, aus der Lösung der folgenden Riccati-Gleichung resultiert [14, S. 616]:

$$S_k = S + A^{\mathrm{T}} S_{k+1} A + (A^{\mathrm{T}} S_{k+1} B + P)(R + B^{\mathrm{T}} S_{k+1} B)^{-1} (B^{\mathrm{T}} S_{k+1} A + P^{\mathrm{T}}). \tag{2.42}$$

Die optimale Rückführung für das deterministische lineare System

$$\boldsymbol{x}_{k+1} = f_d(\boldsymbol{x}_k, \boldsymbol{u}_k) = A\boldsymbol{x}_k + B\boldsymbol{u}_k, \quad k = 0, 1, ..., N-1, \tag{2.43}$$

ist bemerkenswerter Weise identisch zu der des stochastischen Systems (2.26).

Um nun die optimale Steuerfolge $\boldsymbol{U}^*_{[0:N-1]} := \{\boldsymbol{u}^*_0, \boldsymbol{u}^*_1, ..., \boldsymbol{u}^*_{N-1}\}$ zu bestimmen, wird ausgehend von $\boldsymbol{x}_0$ eine Vorwärtsrechnung durch Nutzung der optimalen Rückführung durchgeführt.

$$\begin{aligned} \boldsymbol{u}^*_0 &= K_0 \boldsymbol{x}_0 && \rightarrow && \boldsymbol{x}_1 = A\boldsymbol{x}_0 + B\boldsymbol{u}^*_0, && \text{(2.44a)} \\ &\vdots && && \vdots && \\ \boldsymbol{u}^*_{N-1} &= K_{N-1} \boldsymbol{x}_{N-1} && \rightarrow && \boldsymbol{x}_N = A\boldsymbol{x}_{N-1} + B\boldsymbol{u}^*_{N-1}. && \text{(2.44b)} \end{aligned}$$

Unendlicher Zeithorizont

Für den Grenzwert $N \rightarrow \infty$ geht (2.42) in die diskrete algebraische Riccati-Gleichung

$$S^* = S + A^{\mathrm{T}} S^* A + (A^{\mathrm{T}} S^* B + P)(R + B^{\mathrm{T}} S^* B)^{-1} (B^{\mathrm{T}} S^* A + P^{\mathrm{T}}) \tag{2.45}$$

über, die eine positiv semidefinite Lösung S^* hat [14, S. 617]. Mit dieser ergibt sich die Rückführmatrix zu

$$K^* = (R + B^{\mathrm{T}} S^* B)^{-1} (B^{\mathrm{T}} S^* A + P^{\mathrm{T}}), \tag{2.46}$$

mit der die optimale Rückführung aufgestellt werden kann:

$$\mu^* := K^* \boldsymbol{x}. \tag{2.47}$$

Bei unendlichem Zeithorizont ist die optimale Rückführung π^* demnach zeitinvariant:

$$\pi^* = \{\mu^*, ..., \mu^*\}. \tag{2.48}$$

2.2 Maschinelles Lernen

Das maschinelle Lernen stellt automatisierte Methoden für die Datenanalyse bereit. Es ermöglicht komplexe Aufgaben zu bewältigen, die nicht explizit programmiert werden können.

Das Buch [42, S. 2] liefert eine sehr allgemeine Formulierung für die Begrifflichkeit des Lernens eines Computerprogramms:

Definition 2.5. *A computer program is said to* ***learn*** *from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

Ein Computerprogramm ***lernt*** *aus der Erfahrung E bezogen auf eine Klasse von Aufgaben T und Leistungskriterium P, wenn sich seine Leistung in der Aufgabe T, gemessen durch P, mit der Erfahrung E verbessert.*

Das maschinelle Lernen lässt sich in drei Gebiete unterteilen:

1. Überwachtes Lernen (engl. supervised learning),
2. Unüberwachtes Lernen (engl. unsupervised learning),
3. Bestärkendes Lernen (engl. reinforcement learning).

Unter unüberwachtem Lernen versteht man Methoden, bei denen eigenständig Musterklassen erkannt werden [30, S. 55] Dieses Gebiet des maschinellen Lernens findet in dieser Arbeit keine Anwendung und wird deshalb auch nicht weiter beschrieben. Das bestärkende Lernen wird in Abschnitt 3.1 näher erläutert, weshalb im Folgenden nur auf das überwachte Lernen eingegangen wird.

2.2.1 Überwachtes Lernen

Beim überwachten Lernen ist ein Datensatz $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=0}^{N}$[3] aus Tupeln gegeben, welcher die Erfahrung E repräsentiert. Mit $\boldsymbol{x} \in \mathbb{R}^n$ ist der Eingabe- oder Merkmalsvektor und mit $\boldsymbol{y}$ der Ausgabevektor bezeichnet. Gehören die Ausgaben $\boldsymbol{y} \in \{1, 2, ..., C\}$ einer Menge von Klassen an, die bspw. durch natürliche Zahlen kodiert sind, so spricht man von einer Klassifikationsaufgabe. Gilt hingegen $\boldsymbol{y} \in \mathbb{R}^m$, so handelt es sich um eine Regressionsaufgabe.

In beiden Fällen besteht die Aufgabe darin, die tatsächliche Abbildung $f : \boldsymbol{x} \mapsto \boldsymbol{y}$ zu approximieren, welche dem Datensatz $\mathcal{D}$ zugrunde liegt. Die Approximation von f wird mittels einem durch den Vektor $\boldsymbol{\theta}$ parametrierten Modell $\hat{f} : (\boldsymbol{x}; \boldsymbol{\theta}) \mapsto \hat{\boldsymbol{y}}$ realisiert.

[3] Mit dem hochgestellten Index $^{(\cdot)}$ werden Datenpunkte gekennzeichnet.

Das Leistungskriterium P zu wählen, ist nicht trivial. Es ist mitunter schwer, die Leistung eines Lernsystems in der Aufgabe T zu messen oder zu quantifizieren. Deshalb versucht man durch geeignete Wahl eines Kostenfunktionals f_P, das Leistungskriterium P zu approximieren. Man hofft durch eine Minimierung von f_P, die Leistung des Lernsystems zu verbessern und P positiv zu beeinflussen [17, S. 102].

Beim überwachten Lernen repräsentiert das Kostenfunktional $f_P(\boldsymbol{y}, \hat{\boldsymbol{y}})$ mit

$$\hat{\boldsymbol{y}} = (\hat{y}_0, \hat{y}_1, ..., \hat{y}_N)^{\mathrm{T}} \tag{2.49}$$

und

$$\boldsymbol{y} = (y_0, y_1, ..., y_N)^{\mathrm{T}} \tag{2.50}$$

die Güte der Approximation. Mittels einer Optimierung ist der optimale Parametervektor $\boldsymbol{\theta}^*$ zu bestimmen, welcher $f_P(\boldsymbol{y}, \hat{\boldsymbol{y}})$ minimiert:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} f_P(\boldsymbol{y}, \hat{\boldsymbol{y}}). \tag{2.51}$$

Unter bestimmten Voraussetzungen kann dieses Problem analytisch gelöst werden. In der Regel handelt es sich aber um ein hochdimensionales nichtlineares Optimierungsproblem. Die Optimierung (2.51) wird im Kontext des maschinellen Lernens als *Training* bezeichnet.

Da f_P nicht-konvex ist und viele lokale Minima aufweist, ist die gefundene Lösung $\boldsymbol{\theta}^*$ mit sehr hoher Wahrscheinlichkeit nur ein lokales Optimum. In [11] wird gezeigt, dass die lokalen Minima von f_P für KNN ähnliche Qualität haben, und zudem nicht zu stark vom globalen Minimum abweichen. Konvergiert die Optimierung (2.51) zu einem lokalen Minimum, wird eine Überanpassung (engl. overfitting) unterbunden.

2.3 Künstliche neuronale Netzwerke (KNN)

KNN sind nichtlineare mathematische Modelle, die beim maschinellen Lernen unter anderem verwendet werden, um Abbildungen zu approximieren. Es gibt verschiedene Typen von KNNs, bspw. mehrschichtige Perzeptren (engl. multilayer perceptrons) (MLP), rekurrente neuronale Netzwerke (RNN) und Faltungsnetzwerke (CNN), um nur einige bedeutende zu nennen. In dieser Arbeit wird sich auf die Anwendung von MLP beschränkt, weshalb weitere Typen nicht näher betrachtet werden.

2.3.1 Mehrschichtiges Perzeptron (MLP)

Ein mehrschichtiges Perzeptron (engl. multilayer perceptron) (MLP) ist aus kleinen Recheneinheiten, den sog. Perzeptren (engl. perceptrons) (s. Abbildung 2), zusammengesetzt.

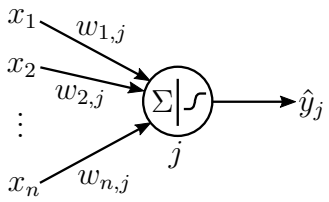

Abbildung 2 – Perzeptron.

Das Perzeptron realisiert die Abbildung:

$$\hat{y}_j = f_{\text{Akt}}(\boldsymbol{w}_j^{\mathrm{T}} \boldsymbol{x} + b_j), \tag{2.52}$$

mit dem Gewichtsvektor $\boldsymbol{w}_j \in \mathbb{R}^n$ und dem sog. Bias $b_j \in \mathbb{R}$, dem Eingabevektor $\boldsymbol{x} \in \mathbb{R}^n$ und der Ausgabe $\hat{y}_j \in \mathbb{R}$. Die Aktivierungsfunktion f_{Akt} ist im Allgemeinen nichtlinear (bspw. tanh). Aus mehreren Perzeptren in Reihen- und Parallelschaltung setzt sich das MLP, wie in Abbildung 3 dargestellt, zusammen.

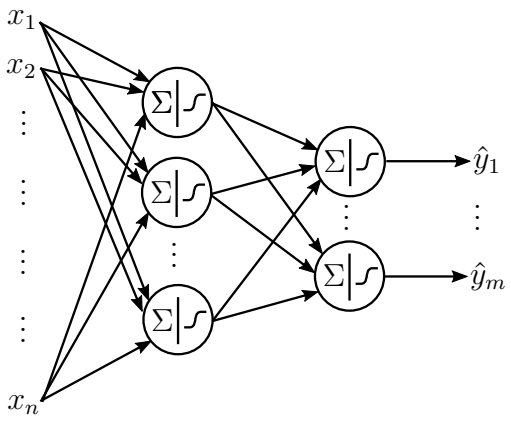

Abbildung 3 – MLP mit zwei Schichten.

Die Modellgleichung des MLP in Abbildung 3 lautet demnach[4]:

$$\hat{\boldsymbol{y}}^{[1]} = f_{\text{Akt}}^{[1]}(W^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]}), \tag{2.53a}$$

$$\hat{\boldsymbol{y}}^{[2]} = f_{\text{Akt}}^{[2]}(W^{[2]} \hat{\boldsymbol{y}}^{[1]} + \boldsymbol{b}^{[2]}), \tag{2.53b}$$

mit den Gewichtsmatrizen

$$W^{[i]} = (\boldsymbol{w}_{0,i}^{\mathrm{T}}, ..., \boldsymbol{w}_{j,i}^{\mathrm{T}})^{\mathrm{T}}, \tag{2.54}$$

sowie den Bias-Vektoren

$$\boldsymbol{b}^{[i]} = (b_{0,i}, ..., b_{j,i})^{\mathrm{T}}, \quad i = 1, 2. \tag{2.55}$$

[4]Mit dem hochgestellten Index $^{[\cdot]}$ wird die Schicht eines KNN gekennzeichnet.

Damit gilt für den Parametervektor

$$\boldsymbol{\theta} := \{W^{[1]}, \boldsymbol{b}^{[1]}, W^{[2]}, \boldsymbol{b}^{[2]}\}. \tag{2.56}$$

Aktivierungsfunktionen

Die gängigen Aktivierungsfunktionen [30, S. 39][17, S. 170]

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.57}$$

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1, \tag{2.58}$$

sowie

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.59}$$

sind in Abbildung 4 dargestellt. Die Rectified-Linear-Unit (ReLU) kommt besonders bei tiefen Netzen zum Einsatz, da der Gradient dieser Funktion stückweise konstant ist, was numerisch günstig für das Training ist.

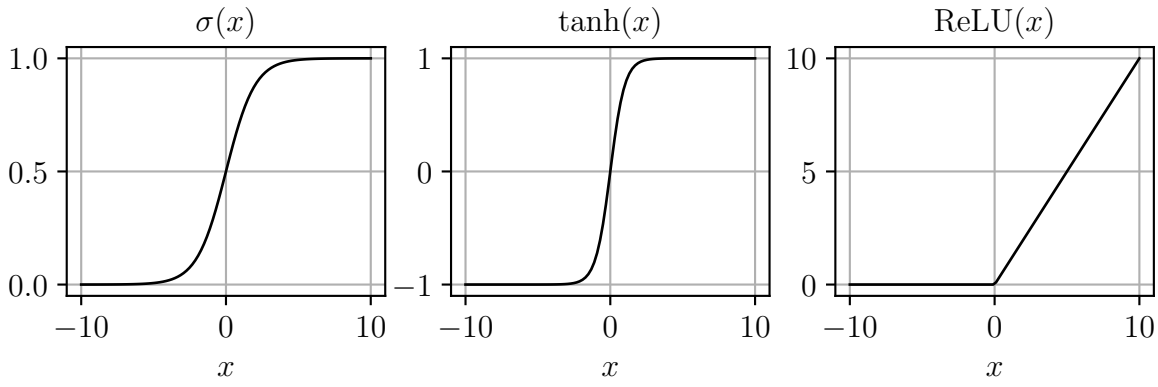

Abbildung 4 – Gängige Aktivierungsfunktionen f_{Akt} für die Verwendung in KNN.

Kostenfunktionale

Das Kostenfunktional (engl. loss function) $f_P(\boldsymbol{y}, \hat{\boldsymbol{y}}) : \boldsymbol{\theta} \mapsto \mathbb{R}$ ist ein Maß für die Güte der Approximation. Je nachdem, was unter Güte in diesem Kontext verstanden wird, lassen sich unterschiedliche Funktionale definieren. Neben den gewöhnlichen Funktionalen

– mittlerer quadratischer Fehler (engl. mean squared error) (MSE)

$$f_P(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{2}||\boldsymbol{y} - \hat{\boldsymbol{y}}||_2^2 \tag{2.60}$$

und

– mittlerer absoluter Fehler (engl. mean absolute error) (MAE)

$$f_P(\boldsymbol{y}, \hat{\boldsymbol{y}}) = ||\boldsymbol{y} - \hat{\boldsymbol{y}}||_1 \tag{2.61}$$

haben besonders Maße aus der Informationstheorie eine Bedeutung [44, S. 57].

Backpropagation (BP)

Backpropagation ist ein Verfahren zur effizienten Bestimmung des Gradienten der Kostenfunktion bezüglich des Parametervektors $\nabla_{\boldsymbol{\theta}} f_P$. Ist $\nabla_{\boldsymbol{\theta}} f_P$ bekannt, kann (2.51) , ausgehend von einem initialen Parametervektor, mittels Gradientenabstiegsverfahren gelöst werden:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \beta \nabla_{\boldsymbol{\theta}} f_P. \tag{2.62}$$

Für ein MLP mit L Schichten

$$\hat{\boldsymbol{y}}^{[1]} = f_{\text{Akt}}^{[1]}(W^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}), \tag{2.63a}$$
$$\hat{\boldsymbol{y}}^{[2]} = f_{\text{Akt}}^{[2]}(W^{[2]}\hat{\boldsymbol{y}}^{[1]} + \boldsymbol{b}^{[2]}), \tag{2.63b}$$
$$\vdots$$
$$\hat{\boldsymbol{y}}^{[L-1]} = f_{\text{Akt}}^{[L-1]}(W^{[L-1]}\hat{\boldsymbol{y}}^{[L-2]} + \boldsymbol{b}^{[L-1]}), \tag{2.63c}$$
$$\hat{\boldsymbol{y}}^{[L]} = f_{\text{Akt}}^{[L]}(W^{[L]}\hat{\boldsymbol{y}}^{[L-1]} + \boldsymbol{b}^{[L]}), \tag{2.63d}$$

ist $\nabla_{\boldsymbol{\theta}} f_P$ als Vektor aufzufassen, der sich folgendermaßen zusammensetzt:

$$\nabla_{\boldsymbol{\theta}} f_P := \left\{ \frac{\partial f_P}{\partial W^{[1]}}, \frac{\partial f_P}{\partial \boldsymbol{b}^{[1]}}, ..., \frac{\partial f_P}{\partial W^{[L]}}, \frac{\partial f_P}{\partial \boldsymbol{b}^{[L]}} \right\}. \tag{2.64}$$

Im folgenden soll die Berechnung am Beispiel von $f_P(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{2}||\boldsymbol{y} - \hat{\boldsymbol{y}}||_2^2$ durchgeführt werden. Die Argumente von $f_{\text{Akt}}^{[l]}$ können zu einem Vektor

$$\boldsymbol{z}^{[l]} := W^{[l]}\hat{\boldsymbol{y}}^{[l-1]} + \boldsymbol{b}^{[l]}, \quad l = 1, ..., L, \tag{2.65}$$

mit

$$\hat{\boldsymbol{y}}^{[0]} := \boldsymbol{x}, \quad \hat{\boldsymbol{y}}^{[L]} := \hat{\boldsymbol{y}}, \tag{2.66}$$

zusammengefasst werden.

Um die einzelnen Komponenten von (2.64) zu bestimmen wird die Kettenregel angewandt. Zunächst werden die Gradienten der Ausgabeschicht L bestimmt:

$$\boldsymbol{\delta}^L := \frac{\partial f_P}{\partial \boldsymbol{z}^{[L]}} = \frac{\partial \hat{\boldsymbol{y}}^{[L]}}{\partial \boldsymbol{z}^{[L]}} \frac{\partial f_P}{\partial \hat{\boldsymbol{y}}^{[L]}} = \operatorname{diag}\left(f'^{[L]}_{\text{Akt}}(\boldsymbol{z}^{[L]})\right)(\hat{\boldsymbol{y}}^{[L]} - \boldsymbol{y}). \tag{2.67}$$

Mit $\boldsymbol{\delta}^L$ lassen sich die Gradienten bezüglich der Gewichte von Schicht L kompakt darstellen:

$$\frac{\partial f_P}{\partial W^{[L]}} = \frac{\partial \boldsymbol{z}^{[L]}}{\partial W^{[L]}} \frac{\partial \hat{\boldsymbol{y}}^{[L]}}{\partial \boldsymbol{z}^{[L]}} \frac{\partial f_P}{\partial \hat{\boldsymbol{y}}^{[L]}} = \hat{\boldsymbol{y}}^{[L-1]} \otimes \boldsymbol{\delta}^L, \tag{2.68a}$$

$$\frac{\partial f_P}{\partial \boldsymbol{b}^{[L]}} = \frac{\partial \boldsymbol{z}^{[L]}}{\partial \boldsymbol{b}^{[L]}} \frac{\partial \hat{\boldsymbol{y}}^{[L]}}{\partial \boldsymbol{z}^{[L]}} \frac{\partial f_P}{\partial \hat{\boldsymbol{y}}^{[L]}} = \boldsymbol{\delta}^L. \tag{2.68b}$$

Das dyadische Produkt $\otimes$ tritt an dieser Stelle auf, da $\frac{\partial \boldsymbol{z}^{[L]}}{\partial W^{[L]}}$ ein Tensor dritter Stufe ist, der sich aus den folgenden Jacobi-Matrizen zusammensetzt:

$$\left(\frac{\partial z_i^{[L]}}{\partial W^{[L]}}\right)_{j,k} = \begin{cases} \hat{y}_k^{[L]}, & \text{wenn} \quad j = i, \\ 0 & \text{sonst.} \end{cases} \tag{2.69}$$

Für die weiteren Schichten $l = 1, ..., L-1$ lässt sich die folgende Iterationsvorschrift herleiten:

$$\boldsymbol{\delta}^l := \frac{\partial f_P}{\partial \boldsymbol{z}^{[l]}} = \frac{\partial \hat{\boldsymbol{y}}^{[l]}}{\partial \boldsymbol{z}^{[l]}} \frac{\partial \boldsymbol{z}^{[l+1]}}{\partial \hat{\boldsymbol{y}}^{[l]}} \boldsymbol{\delta}^{l+1} = \operatorname{diag}\left(f'^{[l]}_{\text{Akt}}(\boldsymbol{z}^{[l]})\right) W^{[l+1]^{\mathrm{T}}} \boldsymbol{\delta}^{l+1}. \tag{2.70}$$

Damit folgt für die weiteren Gradienten:

$$\frac{\partial f_P}{\partial W^{[l]}} = \hat{\boldsymbol{y}}^{[l-1]} \otimes \boldsymbol{\delta}^l, \tag{2.71a}$$

$$\frac{\partial f_P}{\partial \boldsymbol{b}^{[l]}} = \boldsymbol{\delta}^l. \tag{2.71b}$$

Um (2.67), (2.68), (2.70) und (2.71) zu berechnen müssen die Zwischenwerte aller Netzwerkschichtausgaben $\hat{\boldsymbol{y}}^{[l]}, \boldsymbol{z}^{[l]}$ mit $l = 1, ..., L$ bekannt sein. Es muss deshalb für ein gegebenes Paar $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) \in \mathcal{D}$ zunächst eine Vorwärtsrechnung der Modellgleichung (2.63) durchgeführt werden. In Algorithmus 2 wird das Verfahren kompakt zusammengefasst.

Moderne Software-Bibliotheken für maschinelles Lernen nutzen das automatische Differenzieren [57] zur effizienten Berechnung der in Algorithmus 2 benötigten Gradienten.

Algorithmus 2 Backpropagation [17, vgl. S. 208f]

Require: $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) \in \mathcal{D}$
  $\hat{\boldsymbol{y}}^{[l]}, \boldsymbol{z}^{[l]} \quad \forall l \in [1, L]$ ▷ Vorwärtsrechnung des KNN (2.63)
  $\boldsymbol{\delta}^{[L]}, \frac{\partial f_P}{\partial W^{[L]}}, \frac{\partial f_P}{\partial \boldsymbol{b}^{[L]}}$ ▷ Gradienten Schicht L (2.67),(2.68)
  for $l = [L-1, ..., 1]$ **do** ▷ Rückwärtsrechnung
    $\boldsymbol{\delta}^{[l]}, \frac{\partial f_P}{\partial W^{[l]}}, \frac{\partial f_P}{\partial \boldsymbol{b}^{[l]}}$ ▷ Gradienten Schicht l (2.70),(2.71)
  end for

Training von KNN

Mit der Backpropagation lässt sich der Gradient $\nabla_{\boldsymbol{\theta}} f_P$ für ein gegebenes Paar $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ berechnen. Um damit ein KNN zu trainieren, ist es jedoch nötig $\nabla_{\boldsymbol{\theta}} f_P$ für alle N Paare des Datensatzes $\mathcal{D}$ zu berechnen. Der Gradientenabstieg erfolgt dann mittels der folgenden Vorschrift, bei welcher der Mittelwert über alle berechneten Gradienten gebildet wird:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \beta \boldsymbol{g}, \tag{2.72}$$

mit

$$\boldsymbol{g} := \frac{1}{N} \sum_{i=0}^{N} \nabla_{\boldsymbol{\theta}} f_P(\boldsymbol{y}^{(i)}, \hat{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})). \tag{2.73}$$

Da die Berechnung von $\boldsymbol{g}$ für große Datensätze $\mathcal{D}$ sehr rechenintensiv ist, kann $\boldsymbol{g}$ auch geschätzt werden. Dazu wird $\boldsymbol{g}$ nur für eine in jedem Iterationsschritt des Gradientenabstiegsverfahren zufällig ausgewählte Teilmenge $\mathcal{B} \subset \mathcal{D}$ bestimmt:

$$\hat{\boldsymbol{g}} := \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} f_P(\boldsymbol{y}^{(i)}, \hat{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})). \tag{2.74}$$

Dieses Verfahren nennt man stochastisches Gradientenabstiegsverfahren (engl. stochastic gradient descent) (SGD). Die Teilmenge $\mathcal{B}$ wird im Kontext des maschinellen Lernens als *Minibatch* bezeichnet.

2.3.2 Maschinelles Lernen in *Python*

Die Entwicklung freier Software-Bibliotheken für maschinelles Lernen hat einen wesentlichen Beitrag zu dessen Popularität geleistet, da der Einstieg in die Entwicklung von Software, die das maschinelle Lernen nutzt, so bedeutend vereinfacht wurde. Es ist bspw. nicht mehr nötig, das BP-Verfahren oder einen Optimierungsalgorithmus selbst zu implementieren.

Im Rahmen der Arbeit wurde die Software-Bibliothek *Pytorch* [47] genutzt.

Kapitel 3

Bestärkendes Lernen

Das bestärkende Lernen (engl. reinforcement learning) stellt datengetriebene Methoden bereit, die Probleme der sequenziellen Entscheidungsfindung lösen. Als computergestütztes Verfahren weißt es Analogien zu den Lernprozessen von Mensch und Tier auf. Bei der instrumentellen Konditionierung in der Lernpsychologie wird das gewünschte Verhalten eines Individuums durch externe Reize in Form von Belohnungen und Bestrafungen in eine gewünschte Richtung gelenkt [63]. Auch beim bestärkenden Lernen ist eine solche Reizrückkopplung in Form eines Kosten- bzw. Belohnungssignals vorhanden, welches Informationen über die Erfüllung einer Aufgabenstellung widerspiegelt.

Im Kontext der Regelungs- und Steuerungstheorie kann das bestärkende Lernen als adaptive Optimalsteuerung interpretiert werden [67]. Es wird genutzt um selbstlernende Regler zu entwerfen und findet besonders in der Robotik, wo Bildverarbeitung und komplexe Steuerungsaufgaben zusammenkommen, eine breite Anwendung [34, 1, 53, 33, 22, 73].

In diesem Kapitel werden die wesentlichen Konzepte des bestärkenden Lernens erläutert, um eine Basis für die weiterführenden Methoden in Kapitel 4 zu schaffen.

3.1 Agent-Umgebung-Interaktionsmodell

Die Umgebung befindet sich zum Zeitpunkt k im Zustand $\boldsymbol{x}_k \in \mathbb{X}_k$. Der Agent nimmt in jedem Zeitschritt durch die Ausgabe der Steuerung $\boldsymbol{u}_k$ Einfluss auf die Umgebung, wodurch eine Zustandstransition von $\boldsymbol{x}_k$ auf $\boldsymbol{x}_{k+1}$ erfolgt. Der Agent erhält daraufhin den Folgezustand $\boldsymbol{x}_{k+1}$ und die inkrementellen Kosten c_k. Das Agent-Umgebung-Interaktionsmodell ist ein MEP. Zur Veranschaulichung ist dieses Agent-Umgebung-Interaktionsmodell in Abbildung 5 dargestellt.

Die aus der Interaktion mit der Umgebung resultierenden Tupel $d_i = (\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}, c_k^{(i)}, \boldsymbol{x}_{k+1}^{(i)})$ werden in einem stetig wachsenden Datensatz $\mathcal{D} = \{d_0, d_1, ...\}$ abgespeichert, der die Erfahrung E des Agenten repräsentiert.

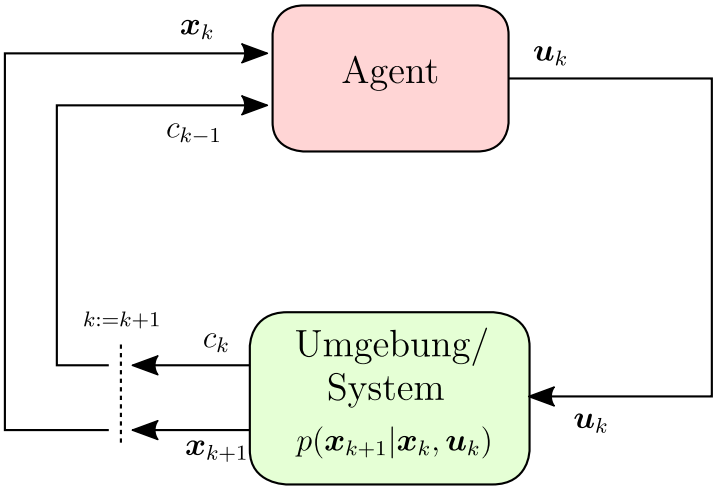

Abbildung 5 – Agent-Umgebung-Interaktionsmodell [66, Abb. 3.1].

In der Literatur des bestärkenden Lernens wird die Steuerung $\boldsymbol{u}_k$ als Aktion (engl. action) $\boldsymbol{a}_k$ bezeichnet, der Zustand (engl. state) $\boldsymbol{x}_k$ mit $\boldsymbol{s}_k$. Zudem wird nicht von Kosten, sondern von Belohnung (engl. reward) r_k gesprochen, wobei der Zusammenhang $c_k = -r_k$ gilt. Die Umgebung entspricht im regelungstechnischen Kontext der Regelstrecke bzw. dem System. Aus Konsistenzgründen zu den vorherigen Betrachtungen werden in dieser Arbeit nach Möglichkeit die regelungstechnischen Symbole und Bezeichnungen genutzt.

3.1.1 Aufgabe des Agenten

Die Aufgabe des Agenten ist es die optimale Rückführung π^* zu bestimmen, welche ein Kostenfunktional J minimiert. Dabei sind neben (2.11) noch die diskontierten Restkosten

$$J = \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau})} \left\{ \sum_{k=0}^{N-1} \gamma^k c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) \right\}, \quad \gamma \in [0, 1), \tag{3.1}$$

sowie die mittleren Restkosten

$$J = \lim_{N\to\infty} \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau})} \left\{ \frac{1}{N} \sum_{k=0}^{N-1} c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) \right\}, \tag{3.2}$$

die den Grenzfall für $\gamma = 1$ von (3.1) bilden, geläufig [29, S. 8]. Im Weiteren wird jedoch ausschließlich (3.1) betrachtet.

Mit γ wird der Diskontierungsfaktor bezeichnet, der angibt, inwieweit zukünftige Kosten gewichtet werden.

3.1.2 Die Umgebung im Kontext des bestärkenden Lernens

Die Dynamik der Umgebung kann als bedingte Verteilung $p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)$ äquivalent zu (2.1) angeben werden. Eine Besonderheit des bestärkenden Lernen ist es, dass der Agent die optimale Rückführung π bestimmen soll, ohne die Systemdynamik (2.1) zu kennen. Die einzige Information, die dem Agenten zur Lösung seiner Aufgabe zur Verfügung steht, ist der Datensatz $\mathcal{D}$. Auf diese Besonderheit wird in Abschnitt 3.4 näher eingegangen.

3.1.3 Der Lernvorgang des Agenten

Der Lernvorgang des Agenten setzt sich aus sog. Episoden zusammen. Zu Beginn einer Episode wird das System wieder in einen Anfangszustand $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$ versetzt[1]. Das Gedächtnis des Agenten geht dabei nicht verloren. Der Agent kann nun für eine festgelegte Anzahl von Zeitschritten mit dem System interagieren und versuchen die Restkosten J zu minimieren. Tritt dabei ein unzulässiger Zustand $\boldsymbol{x}_{k+1} \in \mathbb{X}^-$ ein, wird die Episode abgebrochen. So können bspw. Zustandsbeschränkungen in den Kontext des bestärkenden Lernen einbezogen werden.

3.2 Erkundung und Verwertung

Mit Erkundung (engl. exploration) und Verwertung (engl. exploitation) wird im Kontext des bestärkenden Lernen eine Kosten-Nutzen-Abwägung bezeichnet. Der Agent wird beim Lernvorgang auf seine Erfahrung zurückgreifen, sie *verwerten* und Steuerungen wählen, die zu geringen Kosten geführt haben. Weiterhin sollte er aber auch den Eingangsraum $\mathbb{U}$ *erkunden*, um neue Erfahrung zu sammeln und sich auf deren Basis verbessern zu können. [66, vgl. S. 3]

Dieser Sachverhalt lässt sich an einem Pfadfindungsproblem veranschaulichen. Ein Agent soll eine möglichst kurze Wegstrecke von einem Punkt A zu einem Punkt B finden. Hat er einen Weg von A nach B gefunden, kann er dieses Wissen *verwerten*. Um sich zu verbessern und einen kürzeren Weg zu finden, muss er jedoch von der gefundenen Lösung abweichen, in dem er noch nicht besuchte Wegstücke *erkundet* . Dabei geht er das Risiko eines Umwegs ein. Im schlimmsten Fall ist es nicht mehr möglich, bei B anzukommen.

[1] Bei realen Systemen kann sich dies schwierig gestalten, weshalb dieser Schritt keine Notwendigkeit darstellt.

3.3 Monte-Carlo-Methoden

Eine Möglichkeit zur approximativen Lösung des beschriebenen MEP stellen Monte-Carlo-Methoden dar [66, S. 91]. Damit werden Schätzverfahren bezeichnet, die auf Stichproben basieren. Bspw. kann der Erwartungswert von V^π in (2.25) durch den Mittelwert über Stichproben approximiert werden:

$$V^\pi(\boldsymbol{x}_k) \approx \frac{1}{n} \sum_{i=0}^{n} \sum_{j=k+1}^{N-1} \gamma^{j-k-1} c_j^{(i)}. \tag{3.3}$$

Diese Stichproben werden aus der Interaktion mit dem System erzeugt. Ist die Dynamik bekannt, können diese Stichproben auch mit einer Simulation generiert werden – man spricht dann von Monte-Carlo Baumsuchverfahren [66, S. 198].

3.4 Datengetriebene approximative Lösungsmethoden

Neben dem hohen Rechenaufwand der in Abschnitt 2.1.4 beschriebenen Methoden, kommt hinzu, dass die Systemdynamik (2.1) bekannt sein muss. Dies ist beim bestärkenden Lernen jedoch nicht der Fall. Es müssen deshalb Lösungsmethoden formuliert werden, die ohne diese Kenntnis auskommen.

3.4.1 Temporal-Difference(TD)-Lernen

Beim TD-Lernen werden die beschriebenen Monte-Carlo-Methoden mit der DP kombiniert. Ziel ist es, Restkosten V^π für eine gegebene Rückführung π approximativ zu bestimmen. Die Restkosten V^π werden dabei durch eine Tabelle $\hat{V}^\pi$ approximiert. In jedem Zeitschritt k einer Episode wird die folgende Berechnungsvorschrift für die resultierenden Transitionsdaten $d_i = (\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}, c_k^{(i)}, \boldsymbol{x}_{k+1}^{(i)})$ durchgeführt [66, S. 120]:

$$\hat{V}^\pi(\boldsymbol{x}_k^{(i)}) \leftarrow \hat{V}^\pi(\boldsymbol{x}_k^{(i)}) + \alpha[c_k^{(i)} + \gamma \hat{V}^\pi(\boldsymbol{x}_{k+1}^{(i)}) - \hat{V}^\pi(\boldsymbol{x}_k^{(i)})], \quad \alpha \in (0, 1]. \tag{3.4}$$

Mit α wird die Lernrate bezeichnet. Für $\alpha = 0$ ändert sich der Wert $\hat{V}^\pi(\boldsymbol{x}_k)$ nicht. Für $\alpha = 1$ wird der alte Wert $\hat{V}^\pi(\boldsymbol{x}_k^{(i)})$ verworfen und durch $c_k^{(i)} + \gamma \hat{V}^\pi(\boldsymbol{x}_{k+1}^{(i)})$ ersetzt.

$\hat{V}^\pi$ wird in (3.4) auf sich selbst abgebildet. Diese auf einer Schätzung basierende Schätzung wird als *Bootstrapping* bezeichnet [66, S. 89].

Algorithmus 3 TD-Lernen: Approximation von V^π [66, S. 121]

Lernrate $\alpha \in (0, 1]$, kleines $\varepsilon > 0$
Initialisiere $\hat{V}^\pi(\boldsymbol{x}_k)$ für alle $\boldsymbol{x}_k \in \mathbb{X}_k$
for n Episoden: **do**
  $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$
  for $k = 0, ..., N-1$ **do**
    Agent bestimmt Steuerung $\boldsymbol{u}_k \sim \pi(\boldsymbol{u}_k|\boldsymbol{x}_k)$
    Zustandstransition $p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)$ liefert $c_k, \boldsymbol{x}_{k+1}$
    TD-Update (3.4)
  end for
end for

3.4.2 Q-learning

Das Q-learning stellt eine Erweiterung des TD-Lernen auf Q^π dar. Dadurch ist es möglich, nicht nur die Restkosten V^π zu prädizieren, sondern auch die Rückführung π zu bestimmen.

$$\hat{Q}^\pi(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}) \leftarrow \hat{Q}^\pi(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}) + \alpha[c_k^{(i)} + \gamma \arg\min_{\boldsymbol{u}} \hat{Q}^\pi(\boldsymbol{x}_{k+1}^{(i)}, \boldsymbol{u}) - \hat{Q}^\pi(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)})]. \quad (3.5)$$

Durch die Approximation von Q^* durch die Tabelle $\hat{Q}^\pi$ kann die optimale Rückführung π^* durch π approximiert werden. Q-learning konvergiert für endliche Zustands- und Eingangsräume gegen die optimale Lösung, wenn alle Zustands-Eingangs-Paare unendlich oft besucht werden [41].

Algorithmus 4 Q-learning ($\pi \approx \pi^*$) [74],[66, S. 131]

Lernrate $\alpha \in (0, 1]$, kleines $\varepsilon > 0$
Initialisiere $\hat{Q}^\pi(\boldsymbol{x}_k, \boldsymbol{u}_k)$ für alle $\boldsymbol{x}_k \in \mathbb{X}_k, \boldsymbol{u}_k \in \mathbb{U}_k$
for n Episoden: **do**
  $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$
  for $k = 0, ..., N-1$ **do**
    Agent bestimmt Steuerung $\boldsymbol{u}_k \sim \pi_\varepsilon(\cdot|\boldsymbol{x}_k)$ aus $\hat{Q}^\pi(\boldsymbol{x}_k, \boldsymbol{u}_k)$
    Zustandstransition $p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)$ liefert $c_k, \boldsymbol{x}_{k+1}$
    Q-Update (3.5)
  end for
end for

3.5 Approximative Lösung eines MEP mittels Funktionsapproximation

Für große Zustands- und Eingangsräume sind Methoden, bei denen die Restkosten V^π oder die Bewertungsfunktion Q^π mittels einer Tabelle repräsentiert sind, nicht mehr händelbar. Dies liegt am hohen Speicherbedarf der Tabelle, da für jeden Zustand bzw. jedes Zustands-Eingangs-Paar ein Wert abgespeichert werden muss. Zudem müssten bei einem kontinuierlichen Zustandsraum unendlich viele Werte abgespeichert werden und der Rechenaufwand ist erheblich, wenn bspw. (3.5) oder (3.4) auf jeden Zustand bzw. jedes Zustands-Eingangs-Paar angewendet werden müssen.

Anstatt einer Tabelle kann ein mit dem Vektor $\boldsymbol{\phi}$ parametrierter Funktionsapproximator genutzt werden, um V^π bzw. Q^π zu repräsentieren [59]:

$$V^\pi(\boldsymbol{x}_k) \approx \hat{V}^\pi(\boldsymbol{x}_k;\boldsymbol{\phi}), \tag{3.6}$$

$$Q^\pi(\boldsymbol{x}_k,\boldsymbol{u}_k) \approx \hat{Q}^\pi(\boldsymbol{x}_k,\boldsymbol{u}_k;\boldsymbol{\phi}). \tag{3.7}$$

Bemerkung 3.1. *Von nun an wird für Funktionsapproximatoren die Kurzschreibweise*

$$\hat{V}^\pi_{\boldsymbol{\phi}}(\boldsymbol{x}_k) \equiv \hat{V}^\pi(\boldsymbol{x}_k;\boldsymbol{\phi}), \tag{3.8}$$

$$\hat{Q}^\pi_{\boldsymbol{\phi}}(\boldsymbol{x}_k,\boldsymbol{u}_k) \equiv \hat{Q}^\pi(\boldsymbol{x}_k,\boldsymbol{u}_k;\boldsymbol{\phi}), \tag{3.9}$$

verwendet.

Der Speicheraufwand reduziert sich somit auf den Parametervektor $\boldsymbol{\phi}$. Weiterhin generalisiert ein Funktionsapproximator von bereits besuchten Zuständen bzw. Zustands-Eingangs-Paaren auf nicht besuchte. Wären $V^\pi(\boldsymbol{x}_k)$ und $Q^\pi(\boldsymbol{x}_k,\boldsymbol{u}_k)$ bekannt, könnte man die Methoden des überwachten Lernens aus Abschnitt 2.2.1 anwenden, um diese durch $\hat{V}^\pi$ und $\hat{Q}^\pi$ zu approximieren:

$$\mathcal{D} = \{(\boldsymbol{x}_k^{(i)}, V^\pi(\boldsymbol{x}_k^{(i)})\}_{i=0}^N \tag{3.10}$$

$$\mathcal{D} = \{((\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}), Q^\pi((\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}))\}_{i=0}^N. \tag{3.11}$$

Mit $\mathcal{D}$ kann die Regressionsaufgabe bspw. wie folgt formuliert werden:

$$\min_{\boldsymbol{\phi}} \frac{1}{|\mathcal{D}|} \sum_{i\in\mathcal{D}} \left(\hat{V}^\pi_{\boldsymbol{\phi}}(\boldsymbol{x}_k^{(i)}) - V^\pi(\boldsymbol{x}_k^{(i)})\right)^2 \tag{3.12}$$

bzw.

$$\min_{\boldsymbol{\phi}} \frac{1}{|\mathcal{D}|} \sum_{i\in\mathcal{D}} \left(\hat{Q}^\pi_{\boldsymbol{\phi}}(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}) - Q^\pi(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)})\right)^2. \tag{3.13}$$

Da V^π und Q^π nicht bekannt sind, können (3.5) oder (3.4) genutzt werden, um Trainingsdaten für den Funktionsapproximator zu generieren:

$$\mathcal{D} = \{(\boldsymbol{x}_k^{(i)}, c_k^{(i)} + \gamma \hat{V}_{\boldsymbol{\phi}}^\pi(\boldsymbol{x}_{k+1}^{(i)}\}_{i=0}^N, \tag{3.14}$$

$$\mathcal{D} = \{((\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}), c_k^{(i)} + \gamma \min_{\boldsymbol{u}} \hat{Q}_{\boldsymbol{\phi}}^\pi(\boldsymbol{x}_{k+1}^{(i)}, \boldsymbol{u})\}_{i=0}^N. \tag{3.15}$$

Diese Annahme beruht darauf, dass sowohl V^π als auch Q^π die Bellman-Gleichung, also die Gleichung der DP erfüllen:

$$V^\pi(\boldsymbol{x}_k) = c_k + \gamma V^\pi(\boldsymbol{x}_{k+1}) \tag{3.16a}$$

bzw.

$$Q^\pi(\boldsymbol{x}_k, \boldsymbol{x}_k) = c_k + \gamma \min_{\boldsymbol{u}} Q^\pi(\boldsymbol{x}_{k+1}, \boldsymbol{u}). \tag{3.16b}$$

Definition 3.1 (Mean-Squared-Bellman-Error (MSBE))**.** *Die MSBE-Kostenfunktion ist ein Maß dafür, in wieweit ein Funktionsapproximator $\hat{V}^\pi$ bzw. $\hat{Q}^\pi$ – für einen gegebenen Datensatz $\mathcal{D}$ – die Bellman-Gleichung (3.16) erfüllt:*

$$\text{MSBE}(\boldsymbol{\phi}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \left(\hat{V}_{\boldsymbol{\phi}}^\pi(\boldsymbol{x}_k^{(i)}) - c_k^{(i)} + \gamma \hat{V}_{\boldsymbol{\phi}}^\pi(\boldsymbol{x}_{k+1}^{(i)}) \right)^2 \tag{3.17a}$$

bzw.

$$\text{MSBE}(\boldsymbol{\phi}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \left(\hat{Q}_{\boldsymbol{\phi}}^\pi(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}) - c_k^{(i)} + \gamma \min_{\boldsymbol{u}} \hat{Q}_{\boldsymbol{\phi}}^\pi(\boldsymbol{x}_{k+1}^{(i)}, \boldsymbol{u}) \right)^2. \tag{3.17b}$$

3.5.1 Stabilisierung der approximativen Werte-Iteration

Die Trainingsdaten für den Funktionsapproximator stammen aus der Auswertung des Funktionsapproximators selbst (Bootstrapping). Daraus resultieren schlechte Konvergenzeigenschaften. Weiterhin stellt dieses Vorgehen einen Widerspruch zu Satz 2 dar. Dort werden die Restkosten des Folgezustands V_{k+1} nicht auf sich selbst, sondern auf die Restkosten V_k abgebildet.

Um die Konvergenzeigenschaften zu verbessern werden zwei Funktionsapproximatoren, die durch $\boldsymbol{\phi}$ und $\boldsymbol{\phi}'$ parametriert sind für die Werte-Iteration genutzt. Diese repräsentieren jeweils V_{k+1} und V_k in Satz 2.

Für die Regression von $\hat{V}^\pi$ bzw. $\hat{Q}^\pi$ aus den Trainingsdaten $\mathcal{D}$ wird $\boldsymbol{\phi}$ genutzt. Für die Erzeugung der Trainingsdaten, also der Auswertung der rechten Seite von Satz 2, wird hingegen der Parametervektor $\boldsymbol{\phi}'$ verwendet. Ist die Regression abgeschlossen und soll zum nächsten Schritt der Werte-Iteration übergegangen werden, so wird $\boldsymbol{\phi}' = \boldsymbol{\phi}$ gesetzt. Dieses Verfahren kommt bspw. in [36, 23] zur Anwendung.

3.6 Policy Gradients

Unter dem Begriff Policy-Gradient (PG) werden Methoden zusammengefasst, bei denen eine durch den Parametervektor $\boldsymbol{\theta}$ parametrierte Rückführung

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k|\boldsymbol{x}_k) \tag{3.18}$$

angelernt wird.

Mit dieser parametrierten Rückführung kann eine multivariate Verteilung

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = p(\boldsymbol{x}_0)\prod_{k=0}^{N-1}\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k,\boldsymbol{u}_k) \tag{3.19}$$

eingeführt werden, die die Wahrscheinlichkeit für das Auftreten einer Trajektorie $\boldsymbol{\tau}$ unter dem Einfluss von $\pi_{\boldsymbol{\theta}}$ angibt [49].

Damit ergeben sich analog zu (2.17) die Bewertungsfunktion

$$Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_k,\boldsymbol{u}_k) = \mathbb{E}_{\boldsymbol{\tau}\sim p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left\{\sum_{i=k}^{N-1}\gamma^{i-k}c_i(\boldsymbol{x}_i,\boldsymbol{u}_i)\middle|\boldsymbol{x}_k,\boldsymbol{u}_k\right\},\quad \gamma\in[0,1), \tag{3.20}$$

und analog zu (2.18) die Restkosten zu

$$V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_k) = \mathbb{E}_{\boldsymbol{u}_k\sim\pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}_k)}\{Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_k,\boldsymbol{u}_k)\}. \tag{3.21}$$

Damit kann das Problem der optimalen Steuerung (2.20) als Optimierung über den Parametervektor $\boldsymbol{\theta}$ formuliert werden [32][65, S. 96]:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{3.22}$$

mit

$$J(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{x}_0\sim p(\boldsymbol{x}_0)}\{V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_0)\}. \tag{3.23}$$

Die Optimierung kann bspw. mit einem Gradientenabstiegsverfahren erfolgen [75]:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \beta\nabla_{\boldsymbol{\theta}_k}J(\boldsymbol{\theta}_k), \tag{3.24}$$

wobei der PG $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$ bekannt sein muss und β die Schrittweite bezeichnet.

3.6.1 Stochastic-Policy-Gradient (SPG)

Um den PG $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$ für die Rückführung (3.18) zu bestimmen wird wie folgt vorgegangen.

Aus (3.23) folgt mit $c(\boldsymbol{\tau}) := \sum_{k=0}^{N-1} \gamma^k c_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$ [32]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) c(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau}, \tag{3.25a}$$

$$= \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) c(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \tag{3.25b}$$

mit der Identität $p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \frac{\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} = \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ folgt:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) c(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau}, \tag{3.25c}$$

$$= \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left\{ \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) c(\boldsymbol{\tau}) \right\}. \tag{3.25d}$$

Der Term $\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ lässt sich mit (3.19) in drei Summanden aufteilen:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \log p(\boldsymbol{x}_0) + \sum_{k=0}^{N-1} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k | \boldsymbol{x}_k) + \sum_{k=0}^{N-1} \log p(\boldsymbol{x}_{k+1} | \boldsymbol{x}_k, \boldsymbol{u}_k). \tag{3.26}$$

Bildet man den Gradienten bezüglich des Parametervektors, entfallen die Summanden, die von der Systemdynamik abhängen:

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \nabla_{\boldsymbol{\theta}} \sum_{k=0}^{N-1} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k | \boldsymbol{x}_k), \tag{3.27a}$$

$$= \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k | \boldsymbol{x}_k). \tag{3.27b}$$

Setzt man (3.27b) in (3.25d) ein resultiert daraus der PG [29]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left\{ \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k | \boldsymbol{x}_k) \sum_{i=k}^{N-1} \gamma^{i-k} c_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \right\}, \tag{3.28}$$

$$\overset{(3.20)}{=} \mathbb{E}_{\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left\{ \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k | \boldsymbol{x}_k) Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}_k, \boldsymbol{u}_k) \right\}. \tag{3.29}$$

Mit der diskontierten Zustandsverteilung[2]

$$\rho_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}) := \lim_{N \to \infty} \mathbb{E}_{\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)} \left\{ \sum_{i=1}^{N-1} \gamma^{i-1} \prod_{k=0}^{i} \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k | \boldsymbol{x}_k) p(\boldsymbol{x}_{k+1} | \boldsymbol{x}_k, \boldsymbol{u}_k) \right\} \tag{3.30}$$

ergibt sich der Stochastic-Policy-Gradient (SPG) bei unendlichem Zeithorizont zu [68]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x} \sim \rho_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}), \boldsymbol{u} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})} \left\{ \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}, \boldsymbol{u}) \right\}. \tag{3.31}$$

[2]Durch Einführung der diskontierten Zustandsverteilung kann die Summe in (3.29) aus dem Ausdruck innerhalb des Erwartungswerts genommen werden.

Satz 3 (Stochastic-Policy-Gradient)**.** *Für jede differenzierbare stochastische Rückführung* $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k|\boldsymbol{x}_k)$*, für das Kostenfunktional* J *ist der SPG gegeben durch* (3.29) *bzw.* (3.31) *[68].*

Bemerkung 3.2 (Deterministische Rückführung)**.** *Man nehme an, dass* $\pi := \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)$ *gelte,* π *also deterministisch sei. Damit ergibt sich* (3.26) *zu:*

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \log p(\boldsymbol{x}_0) + \sum_{k=0}^{N-1} \log p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)|_{\boldsymbol{u}_k=\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)}. \tag{3.32}$$

Bildet man nun wie in (3.27) *den Gradienten bezüglich* $\boldsymbol{\theta}$*:*

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \nabla_{\boldsymbol{\theta}} \sum_{k=0}^{N-1} \log p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)|_{\boldsymbol{u}_k=\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)}, \tag{3.33}$$

$$= \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} \log p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)|_{\boldsymbol{u}_k=\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)}, \tag{3.34}$$

fällt der Term, der die Systemdynamik enthält, im Gegensatz zu (3.27) *nicht mehr weg und der resultierende PG ist von dieser abhängig [48]. Die Systemdynamik müsste dann bekannt sein, um den PG berechnen zu können.*

Stellt man die deterministische Rückführung als Verteilung

$$\pi(\boldsymbol{u}_k|\boldsymbol{x}_k) = \begin{cases} 1, & \text{wenn} \quad \boldsymbol{u}_k = \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k), \\ 0 & \text{sonst.} \end{cases} \tag{3.35}$$

dar und setzt diese in (3.29) *ein, führt dies zu Problemen mit dem Term* $\log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k|\boldsymbol{x}_k)$*. Gilt* $\pi(\boldsymbol{u}_k|\boldsymbol{x}_k) = 1$ *und damit* $\log 1 = 0$*, so verschwindet der PG* $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$*. Eine Optimierung mittels Gradientenabstieg ist nicht mehr möglich. Für den Fall* $\pi(\boldsymbol{u}_k|\boldsymbol{x}_k) = 0$ *resultiert daraus* $\log 0$*.* $\log 0$ *ist jedoch nicht definiert.*

Algorithmus 5 REINFORCE, Monte Carlo PG [75]

$\boldsymbol{\tau} \sim p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ ▷ Stichprobe von j Trajektorien

$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \sum_j \left(\sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_k^{(j)}|\boldsymbol{x}_k^{(j)}) \sum_{i=k}^{N-1} \gamma^{i-k} c_i(\boldsymbol{x}_i^{(j)}, \boldsymbol{u}_i^{(j)}) \right) =: \widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$

$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \beta \widehat{\nabla_{\boldsymbol{\theta}_k} J(\boldsymbol{\theta}_k)}$ ▷ Stochastischer Gradientenabstieg SGD

Algorithmus 5 realisiert ein stochastisches Gradientenabstiegsverfahren. Der PG wird durch Monte Carlo Methoden approximiert. Dazu wird $Q^{\pi_{\boldsymbol{\theta}}}$ durch die Summe der diskontierten, inkrementellen Kosten $\sum_{i=k}^{N-1} \gamma^{i-k} c_i(\boldsymbol{x}_i)$ abgeschätzt. Der geschätzte PG $\widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$ unterliegt dabei einer hohen Varianz, woraus schlechte Konvergenzeigenschaften resultieren [32]. Um die Varianz zu reduzieren kann eine sog. *Baseline* $b \in \mathbb{R}$ von Q^{π} in (3.31) subtrahiert werden, ohne den Erwartungswert zu beeinflussen [49]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x} \sim \rho_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}), \boldsymbol{u} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})} \left\{ \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) (Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}, \boldsymbol{u}) - b) \right\}. \tag{3.36}$$

Eine weit verbreitete Baseline sind die Restkosten V^{π}:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim\rho_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}),\boldsymbol{u}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})} \left\{\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x},\boldsymbol{u}) - V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x})\right\} \tag{3.37}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim\rho_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x}),\boldsymbol{u}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})} \left\{\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{x},\boldsymbol{u})\right\}. \tag{3.38}$$

Definition 3.2 (Vorteilsfunktion A^{π})**.** *Die Vorteilsfunktion (engl. advantage function)*

$$A^{\pi}(\boldsymbol{x}_k,\boldsymbol{u}_k) := Q^{\pi}(\boldsymbol{x}_k,\boldsymbol{u}_k) - V^{\pi}(\boldsymbol{x}_k) \tag{3.39}$$

für eine gegebene Rückführung π gibt an, wie viel besser als der Durchschnitt die Steuerung $\boldsymbol{u}_k$ im Zustand $\boldsymbol{x}_k$ ist [59].

3.6.2 Deterministic-Policy-Gradient (DPG)

In [61] werden die vorgestellten Ergebnisse für deterministische Rückführungen der Form $\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)$ entwickelt.[3] Davor wurde nicht davon ausgegangen, dass $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ für diese Form von Rückführungen existiert [48]. Die Autoren konnten zeigen, dass der Deterministic-Policy-Gradient (DPG) einen Grenzfall des SPG in Satz 3 darstellt.

Zunächst wird für eine parametrierte deterministische Rückführung $\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)$ in Analogie zu (3.19) die multivariate Verteilung

$$p_{\mu_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) = p(\boldsymbol{x}_0) \prod_{k=0}^{N-1} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k,\boldsymbol{u}_k)|_{\boldsymbol{u}_k=\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)} \tag{3.40}$$

eingeführt.

Die zu minimierende Kostenfunktion

$$J(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{x}_0\sim p(\boldsymbol{x}_0)}\{V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_0)\} \tag{3.41}$$

wird nach $\boldsymbol{\theta}$ differenziert:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}_0\sim p(\boldsymbol{x}_0)}\{\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_0)\}. \tag{3.42}$$

Um den DPG zu bestimmen, muss demnach $\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_0)$ bekannt sein.

[3] Es wird vorausgesetzt, dass der PG $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{x}_k)$ existiert und ebenso wie $\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)$ stetig in $\boldsymbol{x}_k$ und $\boldsymbol{\theta}$ ist.

Die Berechnung erfolgt im Folgenden für den allgemeinen Fall $\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_k)$ in Anlehnung an den Beweis in [61]:

$$\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_k) = \nabla_{\boldsymbol{\theta}} Q^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)), \tag{3.43a}$$

$$\overset{(2.25a)}{=} \nabla_{\boldsymbol{\theta}} \left(c_k(\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) + \gamma \mathbb{E}_{\boldsymbol{x}_{k+1} \sim p(\cdot|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k))} \{ V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1}) \} \right), \tag{3.43b}$$

$$= \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} c_k(\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) \tag{3.43c}$$

$$+ \gamma \nabla_{\boldsymbol{\theta}} \int p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1}) \mathrm{d}\boldsymbol{x}_{k+1},$$

$$= \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} c_k(\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) \tag{3.43d}$$

$$+ \gamma \int \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1})$$

$$+ p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1}) \mathrm{d}\boldsymbol{x}_{k+1},$$

$$= \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} c_k(\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) \tag{3.43e}$$

$$+ \gamma \int \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1})$$

$$+ p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1}) \mathrm{d}\boldsymbol{x}_{k+1},$$

$$= \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} \left(c_k(\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) + \gamma \int p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1}) \mathrm{d}\boldsymbol{x}_{k+1} \right) \tag{3.43f}$$

$$+ \gamma \int p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1}) \mathrm{d}\boldsymbol{x}_{k+1},$$

$$= \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} \left(c_k(\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) + \gamma \mathbb{E}_{\boldsymbol{x}_{k+1} \sim p(\cdot|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k))} \{ V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1}) \} \right) \tag{3.43g}$$

$$+ \gamma \mathbb{E}_{\boldsymbol{x}_{k+1} \sim p(\cdot|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k))} \{ \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1}) \},$$

$$\nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_k) \overset{(2.25a)}{=} \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} Q^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_k, \boldsymbol{u}_k)|_{\boldsymbol{u}_k = \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)} \tag{3.43h}$$

$$+ \gamma \mathbb{E}_{\boldsymbol{x}_{k+1} \sim p(\cdot|\boldsymbol{x}_k, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k))} \{ \nabla_{\boldsymbol{\theta}} V^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_{k+1}) \}.$$

Führt man die Rekursion in (3.43h) fort, setzt diese in (3.42) ein, und fasst den Erwartungswert über $\boldsymbol{\tau}$ zusammen, so erhält man einen Ausdruck für den DPG:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim p_{\mu_{\boldsymbol{\theta}}}(\boldsymbol{\tau})} \left\{ \sum_{k=0}^{N-1} \gamma^k \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k) \nabla_{\boldsymbol{u}_k} Q^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}_k, \boldsymbol{u}_k)|_{\boldsymbol{u}_k = \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)} \right\}. \tag{3.44}$$

Mit der diskontierten Zustandsverteilung[4]

$$\rho_{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}) := \lim_{N \to \infty} \mathbb{E}_{\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)} \left\{ \sum_{i=1}^{N-1} \gamma^{i-1} \prod_{k=0}^{i} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)|_{\boldsymbol{u}_k = \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)} \right\} \tag{3.45}$$

[4]Durch Einführung der diskontierten Zustandsverteilung kann die Summe im (3.44) aus dem Ausdruck innerhalb des Erwartungswerts genommen werden.

ergibt sich der DPG bei unendlichem Zeithorizont zu [61, Gl. 9]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim\rho_{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x})} \left\{ \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}) \nabla_{\boldsymbol{u}} Q^{\mu_{\boldsymbol{\theta}}}(\boldsymbol{x}, \boldsymbol{u})|_{\boldsymbol{u}=\mu_{\boldsymbol{\theta}}(\boldsymbol{x})} \right\}. \tag{3.46}$$

Der DPG ist demnach der Erwartungswert des Gradienten der Bewertungsfunktion.

Satz 4 (Deterministic-Policy-Gradient)**.** *Für eine deterministische Rückführung $\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)$, für das Kostenfunktional J ist der PG gegeben durch (3.44) bzw. (3.46) [61].*

3.7 Aktor-Kritiker Algorithmen

Bei Aktor-Kritiker (engl. actor-critic) Algorithmen wird sowohl die Rückführung π (Aktor) als auch die Restkosten V^{π}, bzw. die Bewertungsfunktion Q^{π} (Kritiker) durch einen Funktionsapproximator repräsentiert (s. Abbildung 6).

Der Schätzer $\hat{V}^{\pi}$ bzw. $\hat{Q}^{\pi}$ wird dabei mit den Methoden aus Abschnitt 3.5 erlernt. Dieser wird dann genutzt um mittels Satz 3 oder Satz 4 den PG $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ zu schätzen und iterativ die Rückführung $\pi_{\boldsymbol{\theta}}$ zu optimieren.

Die Aktor-Kritiker Architektur findet bspw. bei dem in Abschnitt 4.3 vorgestellten Algorithmus DDPG Verwendung.

3.8 Imitationslernen

Das Imitationslernen stellt die einfachste Form des bestärkenden Lernens dar. Dabei wird davon ausgegangen, dass zunächst nicht der Agent selbst, sondern ein Experte mit dem System interagiert. Mit den Methoden des überwachten Lernens wird anschließend die Rückführung $\pi_{\boldsymbol{\theta}}$ bzw. $\mu_{\boldsymbol{\theta}}$ bestimmt, die das Verhalten des Experten imitiert [25].

In [6] wird das Imitationslernen auf das autonome Fahren angewandt. Den Autoren ist es gelungen einen kamerabasierten Lenkregler zur Spurverfolgung zu trainieren, der das Verhalten eines menschlichen Fahrers (Experte) imitiert. Der angelernte Regler wurde an einem realen Fahrzeug in verschieden Verkehrssituationen erprobt.

In [51] wird das Imitationslernen genutzt, um auf Basis der Trajektorienplanung (Experte) eine Überführungsaufgabe zu lösen. Dazu wurden zunächst optimale Eingangstrajektorien aus einer Menge von Anfangswerten $\mathbb{X}_0$ generiert. Der resultierenden Datensatz aus Tupeln $(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)})$ wurde anschließend genutzt um die optimale Rückführung π^* mittels eines KNN

$$\mu_{\boldsymbol{\theta}} : \mathbb{X}_k \to \mathbb{U}_k, \boldsymbol{x}_k \mapsto \boldsymbol{u}_k \tag{3.47}$$

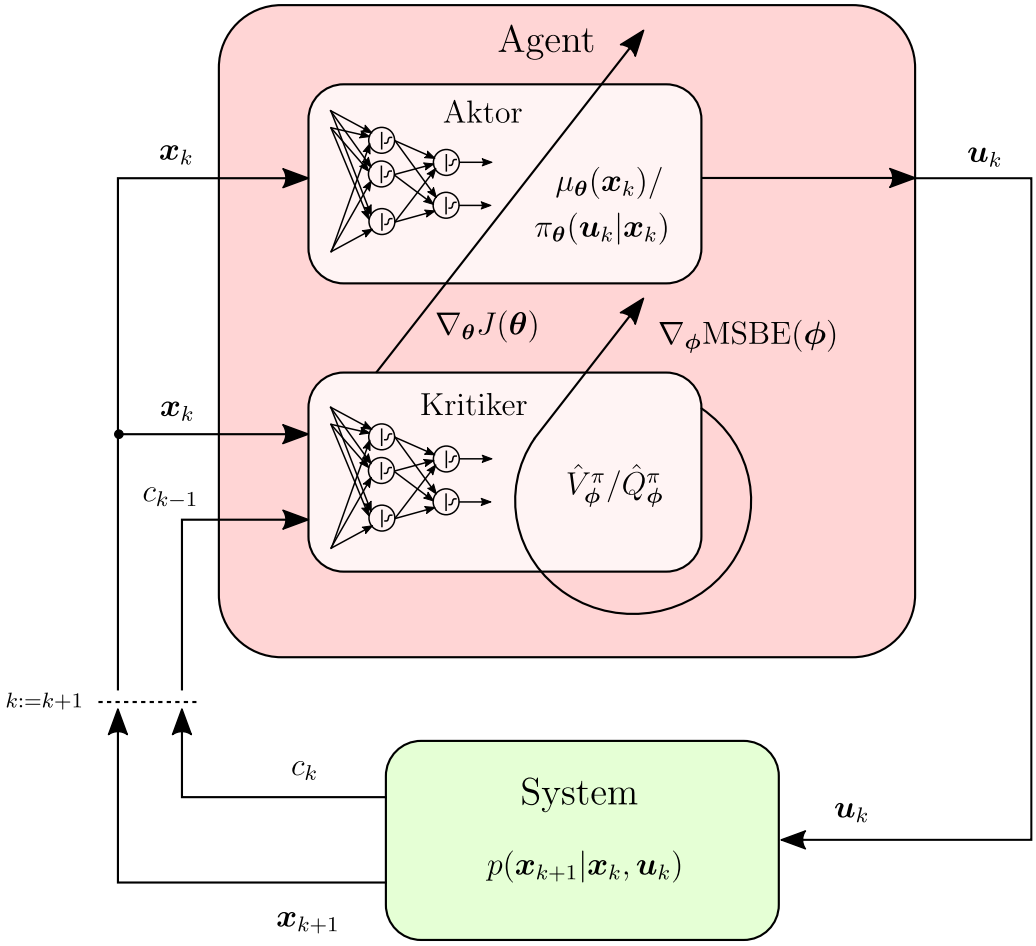

Abbildung 6 – Architektur von Aktor-Kritiker Algorithmen. Der Kritiker wird mit TD-Lernen und Minimierung des MSBE trainiert ($\nabla_{\boldsymbol{\phi}}$MSBE$(\boldsymbol{\phi})$). Mit dem Kritiker kann der PG $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ geschätzt werden, um mit diesem den Aktor derart zu optimieren, dass er J minimiert [66, vgl. Abb. 15.5].

zu approximieren. Mit dieser Rückführung ist es möglich näherungsweise optimale Trajektorien zu generieren, ohne eine erneute Trajektorienplanung durchzuführen.

Ein Nachteil des Imitationslernen ist, dass der Agent nicht besser als der Experte werden kann. Zudem ist es mitunter sehr zeitaufwendig und teuer, einen Experten Trainingsdaten erzeugen zu lassen.

3.9 Modellbasiertes bestärkendes Lernen

Beim modellbasierten bestärkenden Lernen wird im Gegensatz zu den bereits vorgestellten Methoden ein Systemmodell erlernt. Dem Ansatz liegt die Annahme zu Grunde, dass

die Aufgabe des Agenten mit den Methoden der optimalen Steuerung (s. Abschnitt 2.1) gelöst werden kann, wenn die Systemdynamik (2.1) bzw. (2.3) bekannt ist.

Es wird ein durch $\boldsymbol{\phi}$ parametrierten Schätzer $\hat{f}_d : (\boldsymbol{x}_k, \boldsymbol{u}_k; \boldsymbol{\phi}) \mapsto \boldsymbol{x}_{k+1}$ von (2.1) bzw. (2.3) aus den Interaktionsdaten $\mathcal{D}$ angelernt. Im Gegensatz zur Systemidentifikation, wo Expertenwissen mit in den Entwurf des Schätzers fließt, wird im Kontext des bestärkenden Lernens ein möglichst allgemein gehaltener Ansatz verfolgt. So soll gewährleistet werden, dass sich ein Algorithmus auf möglichst viele verschiedene Systeme anwenden lässt.

Bei dem Algorithmus Probabilistic-Inference-for-Learning-Control (PILCO) [13] wird ein Gaußscher Prozess genutzt, um die Systemdynamik (2.3) zu erlernen. Dieser nicht-parametrische stochastische Ansatz ist sehr dateneffizient. Die erlernte Systemdynamik wird genutzt um für eine parametrierte Rückführung $\pi_{\boldsymbol{\theta}}$ den PG $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ analytisch zu bestimmen. So kann bspw. das Aufschwingen des Doppelpendels (s. Abschnitt A.4) in 20 bis 30 Episoden, bzw. 60 bis 90 Sekunden Interaktion mit dem System erlernt werden. Der Rechenaufwand von PILCO ist jedoch erheblich und der Algorithmus kann durch die Nutzung von Gaußschen Prozessen, nur auf solche niedrig-dimensionalen (bezogen auf die Zustandsdimension) Probleme angewandt werden [36, 73].

Kapitel 4

Tiefes bestärkendes Lernen

Unter dem Begriff *tiefes bestärkendes Lernen (engl. deep reinforcement learning)* werden Methoden zusammengefasst, die in ihrer Architektur tiefe KNNs verwenden.

Durch die Nutzung von tiefen KNNs konnten beeindruckende Fortschritte auf dem Gebiet des bestärkenden Lernens erreicht werden, die internationales Aufsehen erregten [43, 60]. Dadurch ist es möglich das bestärkende Lernen auf Probleme mit sehr hoher Zustands- und Eingangsdimension anzuwenden. Ist der Zustand $\boldsymbol{x}_k$ durch Pixel eines Bildes repräsentiert, bspw. bei einem Computerspiel, oder bei einer Kameraaufnahme eines Roboters, so gilt $\dim(\boldsymbol{x}_k) >> 10^4$. Für solche Anwendungsfälle werden Faltungsnetzwerke (CNN) eingesetzt. In dieser Arbeit kommen jedoch nur MLPs zur Anwendung, da keine Bilddaten oder ähnlich hoch-dimensionale Objekte zum Lernen verwendet werden.[1]

In diesem Kapitel werden drei Methoden des tiefen bestärkenden Lernens vorgestellt. Diese bilden die historische Entwicklung des Wissenschaftsgebietes ab und ermöglichen es Heuristiken und Konzepte motiviert einzuführen.

4.1 Neural-Fitted-Q-Iteration (NFQ)

Bei NFQ [55] wird die Werteiteration des Q-learning mit der Funktionsapproximation durch ein KNN kombiniert. Der Algorithmus ist dateneffizient und wurde von den Autoren auf reale Systeme, wie bspw. das Wagen-Pendel (s. Abschnitt A.3) angewandt.[2] In [51] wurde der Algorithmus in Simulationsstudien zur Stabilisierung eines aufrecht fahrenden Balancierroboters eingesetzt.

[1] Eine entsprechende Erweiterung der entwickelten Software, wäre mit überschaubarem Aufwand zu bewältigen.

[2] Ein Aufschwingen aus der unteren Ruhelage ist mit diesem Algorithmus nicht möglich.

Die erlernte Bewertungsfunktion $\hat{Q}_\phi$ wird genutzt, um den Eingang zu bestimmen:

$$\boldsymbol{u}_k := \arg\min_{\boldsymbol{u}\in\mathbb{U}} \hat{Q}_\phi(\boldsymbol{x}_k, \boldsymbol{u}). \tag{4.1}$$

Da diese Optimierung für kontinuierliche Eingänge sehr rechenaufwendig sein kann ist NFQ nur für diskrete Eingänge geeignet. Sollte der Eingang dennoch kontinuierlich sein, muss er entsprechend diskretisiert werden. Für den diskreten Eingang kann (4.1) für alle $\boldsymbol{u} \in \mathbb{U}$ ausgewertet werden. Der Rechenaufwand steigt mit der Mächtigkeit von $\mathbb{U}$, weshalb in [55] für jede Dimension von $\boldsymbol{u}_k$ nur zwei bis drei Werte vorgegeben werden. Für ein System mit zwei Eingängen $u_1 \in \{u_{1,\min}, 0, u_{1,\max}\}, u_2 \in \{u_{2,\min}, u_{2,\max}\}$ gilt $|\mathbb{U}| = 6$. Für die Auswertung von (4.1) muss $\hat{Q}_\phi$ demnach 6-mal ausgewertet werden. Der Algorithmus ist deshalb nur für kleine $|\mathbb{U}|$ geeignet.

4.1.1 KNN und Training

Bei NFQ wird ein relativ kleines KNN genutzt, um Q^π zu repräsentieren. Die Autoren schlagen als Architektur ein MLP vor. Dieses hat zwei versteckte Schichten mit je 20 Perzeptren und tanh-Aktivierungsfunktion. In der Ausgabeschicht wird die σ-Aktivierungsfunktion verwendet, um die Werte von $\hat{Q}_\phi$ auf das Intervall $[0, 1]$ zu beschränken. Nach Beendigung einer Episode wird $\hat{Q}_\phi$ mit der MSBE-Kostenfunktion für den gesamten Datensatz $\mathcal{D}$ trainiert, bis die Regression konvergiert ist. Da der Datensatz $\mathcal{D}$ stetig wächst, steigt die Trainingsdauer mit jeder Episode an. Um dem entgegenzuwirken ist die Größe von $\mathcal{D}$ auf 50000 Datenpunkte d_i beschränkt. Der Agent nutzt die *ε-gierige* Rückführung.

4.1.2 Kosten

Die Kostenfunktion c_k ist bei NFQ nicht vom Eingang $\boldsymbol{u}_k$ abhängig. Es werden zunächst eine Menge von Zielzuständen $\mathbb{X}^+$ definiert, in die das System möglichst schnell überführt werden soll. Weiterhin wird eine Menge von unzulässigen Zuständen $\mathbb{X}^-$ eingeführt, womit bspw. Zustandsbeschränkungen modelliert werden können. Die Kostenfunktion ergibt sich damit zu:

$$c_k(\boldsymbol{x}_k) = \begin{cases} 0, & \text{wenn} \quad \boldsymbol{x}_k \in \mathbb{X}^+, \\ 1, & \text{wenn} \quad \boldsymbol{x}_k \in \mathbb{X}^-, \\ 0.01 & \text{sonst.} \end{cases} \tag{4.2}$$

Die korrespondierende optimale Rückführung π^* überführt das System aus einem Anfangszustand $\boldsymbol{x}_0$ in möglichst wenig Zeitschritten in die Menge $\mathbb{X}^+$ und ist damit zeitoptimal.

Algorithmus 6 Neural-Fitted-Q-Iteration (NFQ) [55]

Initialisierung des Datensatzes $\mathcal{D}$
Zufällige Initialisierung des Q-Netzwerks $\hat{Q}_{\phi}(\boldsymbol{x}_k, \boldsymbol{u}_k)$
for n Episoden **do** ▷ Lernvorgang
  $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$ ▷ Anfangszustand bestimmen
  for $k \in [0, N-1]$ **do**
    $\boldsymbol{u}_k \sim \pi_{\varepsilon}(\boldsymbol{u}_k|\boldsymbol{x}_k)$ ▷ ε-gierige Rückführung
    $\boldsymbol{x}_{k+1} \sim p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)$ ▷ Simulation des Systems
    $c_k(\boldsymbol{x}_k)$ ▷ Berechnung der inkrementellen Kosten
    $\mathcal{D} \leftarrow d_i = (\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}, c_k^{(i)}, \boldsymbol{x}_{k+1}^{(i)})$ ▷ Transitionsdaten speichern
    Setze $y^{(i)} = \begin{cases} 1, & \text{für } \boldsymbol{x}_{k+1}^{(i)} \in \mathbb{X}^{-}, \\ c_k^{(i)} + \gamma \min\limits_{\boldsymbol{u}} \hat{Q}_{\boldsymbol{\phi}}(\boldsymbol{x}_{k+1}^{(i)}, \boldsymbol{u}) & \text{sonst.} \end{cases}$ ▷ Trainingsdaten
  end for
  $\min_{\phi} \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} (y^{(i)} - \hat{Q}_{\phi}(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}))^2$ ▷ Training des MLP
end for

4.2 Deep-Q-Network (DQN)

Mit DQN [43] wird ein wichtiger Meilenstein des bestärkenden Lernens markiert. Durch die Kombination von bestärkendem und tiefem maschinellen Lernen war es erstmals möglich, Steuerungen aus hochdimensionalen Objekten zu erlernen. Den Autoren ist es gelungen einen Algorithmus zu entwerfen, der aus den Pixeldaten des Spielbildschirms lernt Videospiele zu spielen. Dem Agenten steht dabei keine domänenspezifische Information über das jeweilige Videospiel zur Verfügung und alle Hyperparameter des Algorithmus wurden konstant gehalten. Die Autoren haben DQN auf 49 Spiele der Videospielkonsole *Atari 2600* angewandt. Bei 29 Spielen erzielte der Agent nach dem Lernvorgang höhere Punktzahlen, als ein überdurchschnittlicher menschlicher Spieler.

4.2.1 Besonderheiten von DQN gegenüber NFQ

Der Algorithmus DQN kombiniert NFQ mit den Methoden des tiefen maschinellen Lernen. Um die Bilddaten zu verarbeiten wird für die Repräsentation des Q-Netzwerks $\hat{Q}_{\phi}$ ein tiefes KNN genutzt. Zudem werden bei DQN noch heuristische Verfahren angewandt, die den Lernvorgang beschleunigen und stabilisieren.

Q-Netzwerk

Der Zustand $\boldsymbol{x}_k \in \mathbb{R}^{210\times160\times3\times4}$ ist durch die Pixeldaten der letzten vier Spielbildschirme repräsentiert, der diskrete Eingang $\boldsymbol{u}_k \in \mathbb{R}^{18}$ durch Tasten und mögliche Tastenkombinationen des Spielcontrollers. Mittels einer Abbildung

$$h : \mathbb{R}^{210\times160\times3\times4} \to \mathbb{R}^{84\times84\times4}, \boldsymbol{x}_k \mapsto \boldsymbol{o}_k \tag{4.3}$$

werden die Bilddaten vorverarbeitet, bevor sie dem Q-Netzwerk übergeben werden.

Um bei der Berechnung von $\boldsymbol{u}_k$ mittels (4.1) nicht für jede mögliche Steuerung das Q-Netzwerk auswerten zu müssen, wird ein Trick angewandt. Im Gegensatz zu NFQ hat $\hat{Q}_\phi$ nur $\boldsymbol{o}_k$ als Eingabe und gibt dafür einen Vektor $(\hat{Q}_\phi(\boldsymbol{o}_k, u_1), ..., \hat{Q}_\phi(\boldsymbol{o}_k, u_{18}))^\mathrm{T}$ für alle möglichen Steuerungen aus. Für die Berechnung von (4.1) muss nur noch über diesen Vektor minimiert werden. Zur besseren Veranschaulichung ist $\hat{Q}_\phi$ in Abbildung 7 dargestellt.

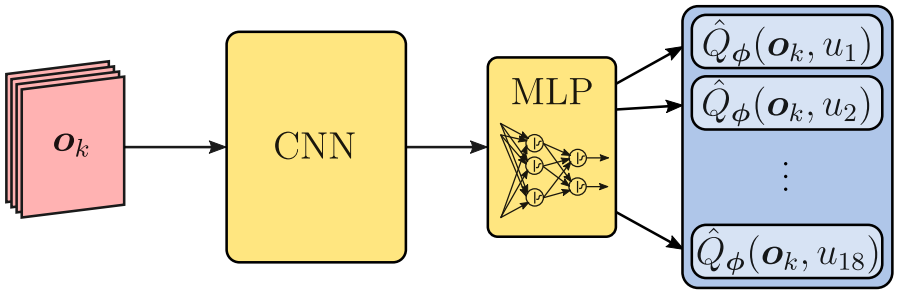

Abbildung 7 – Schematische Struktur des bei DQN verwendeten Q-Netzwerks.

Experience Replay

Anstatt, wie bei NFQ, das Q-Netzwerk $\hat{Q}_\phi$ nach jeder Episode mit dem gesamten Datensatz $\mathcal{D}$ zu trainieren, wird nach jedem Zeitschritt k ein Minibatch $\mathcal{B} \subset \mathcal{D}$ ausgewählt. Auf dessen Basis wird der Gradient des MSBE ∇_ϕ geschätzt und ein Iterationsschritt des gewählten Gradientenabstiegsverfahrens ausgeführt. Durch dieses SGD-Verfahren steigt die benötigte Rechenzeit für die Optimierung von $\hat{Q}_\phi$ bei wachsendem Datensatz $\mathcal{D}$ nicht wesentlich an.

Target-Netzwerk

Wie in Abschnitt 3.5.1 beschrieben, weist die Werte-Iteration bei der Nutzung von Funktionsapproximatoren schlechte Konvergenzeigenschaften auf, die unter anderem

durch das Bootstrapping verursacht werden. Bei DQN wird deshalb ein sog. Target-Netzwerk $\hat{Q}_{\boldsymbol{\phi}'}$ eingesetzt, dass die Werte-Iteration stabilisiert. Mit diesem KNN werden die Sollwerte (engl. target values) $y^{(i)}$ in Algorithmus 6 erzeugt. Nach einer bestimmten Anzahl Episoden wird $\boldsymbol{\phi}' \leftarrow \boldsymbol{\phi}$ gesetzt.

4.3 Deep-Deterministic-Policy-Gradient (DDPG)

DDPG [36] kombiniert die Ergebnisse aus Abschnitt 3.6.2 mit dem tiefen maschinellen Lernen. So kann das bestärkende Lernen erstmals auf hochdimensionale Systeme mit kontinuierlichem Eingang $\boldsymbol{u}_k$ angewandt werden. DDPG verwendet die Aktor-Kritiker-Architektur (s. Abschnitt 3.7) und nutzt KNN zur Repräsentation von Aktor $\mu_{\boldsymbol{\theta}}$ und Kritiker $\hat{Q}^{\mu}_{\boldsymbol{\phi}}$ (s. Abbildung 8).

Der Kritiker wird wie bei NFQ und DDPG durch Minimierung des MSBE trainiert (4.9). Mit dem Kritiker kann der DPG bestimmt werden, mit dem der Aktor trainiert wird (4.10). Anstatt wie bei NFQ und DQN den Kritiker zu minimieren, um die Steuerung zu bestimmen, werden mit dem DPG die Parameter des Aktors derart angepasst, dass dieser ein Minimierer des Kritikers ist. Durch den Einsatz des Aktors kann der Eingang $\boldsymbol{u}_k$ kontinuierliche Werte annehmen.

4.3.1 Target-Netzwerke

Ebenso wie bei DQN werden Target-Netzwerke $\mu_{\boldsymbol{\theta}'}$ und $\hat{Q}^{\mu}_{\boldsymbol{\phi}'}$ genutzt, um die Regler-Iteration zu stabilisieren. Statt jedoch eine harte Aktualisierung

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}, \tag{4.4}$$
$$\boldsymbol{\phi}' \leftarrow \boldsymbol{\phi} \tag{4.5}$$

der Parametervektoren durchzuführen, wird in jedem Zeitschritt eine weiche Aktualisierung durchgeführt, bei der die Parameter der Target-Netzwerke $\boldsymbol{\theta}'$ und $\boldsymbol{\phi}'$ in Richtung der Parameter von Aktor ($\boldsymbol{\theta}$) und Kritiker ($\boldsymbol{\phi}$) bewegt:

$$\boldsymbol{\phi}' \leftarrow \tau\boldsymbol{\phi} + (1-\tau)\boldsymbol{\phi}', \tag{4.6a}$$
$$\boldsymbol{\theta}' \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta}', \quad \tau \ll 1. \tag{4.6b}$$

So wird gewährleistet, dass sich die Parameter der Target-Netzwerke nur langsam verändern und den Lernvorgang stabilisieren.

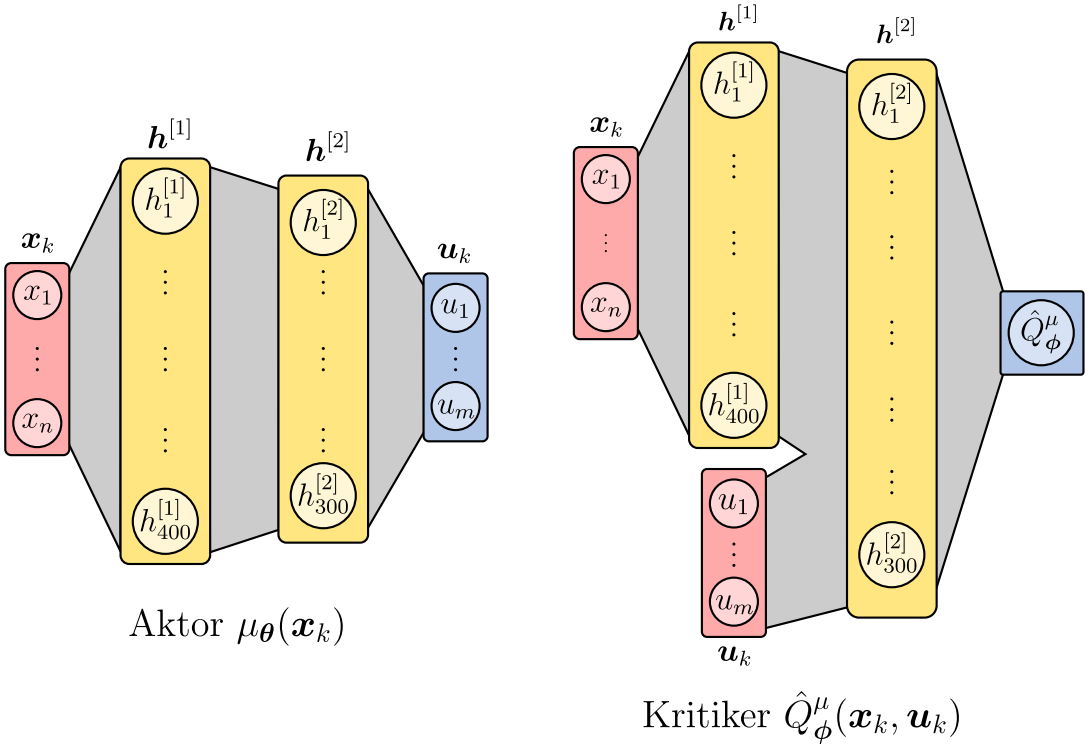

Abbildung 8 – Architektur der verwendeten KNN bei DDPG. Beide KNN haben zwei versteckte Schichten mit ReLU-Aktivierungsfunktion und respektive 300 bzw. 400 Perzeptren. Die Ausgabeschicht des Aktors hat eine tanh-Aktivierungsfunktion, die die Ausgabe auf das Intervall $[-1, 1]$ beschränkt. Um die Netzwerkausgabe $\boldsymbol{u}_k$ auf ein definiertes Intervall $[-\boldsymbol{u}_{max}, \boldsymbol{u}_{max}]$ zu skalieren, wird die Ausgabe der Aktivierungsfunktion mit $\boldsymbol{u}_{max}$ multipliziert. Beim Kritiker wird der Eingang $\boldsymbol{u}_k$ erst in der zweiten versteckten Schicht eingebunden. Die Ausgabeschicht hat im Gegensatz zu [36] eine ReLU-Aktivierungsfunktion, um die Ausgabe auf $\mathbb{R}^{\geq 0}$ zu beschränken. Da für die inkrementellen Kosten $c_k \geq 0$ angesetzt wird, kann $\hat{Q}^{\mu}$ nicht negativ sein.

4.3.2 Erkundung

Auch bei DDPG hat die Erkundung einen nennenswerten Einfluss auf die Konvergenzeigenschaften des Algorithmus. Bei DDPG wird dazu ein Rauschprozess $\mathcal{R}$ auf die Ausgabe des Aktor-Netzwerks addiert:

$$\boldsymbol{u}_k = \mu_{\boldsymbol{\theta}_1}(\boldsymbol{x}_k) + \mathcal{R}_k. \tag{4.7}$$

Die Autoren nutzen für den Rauschprozess $\mathcal{R}$ einen Ornstein-Uhlenbeck-Prozess, der korreliertes Rauschen erzeugt. In [50] wird als Alternative zu (4.7) vorgeschlagen, den Parametervektor $\boldsymbol{\theta}$ zu verrauschen.

Im Rahmen dieser Arbeit wurde normalverteiltes Rauschen $\mathcal{N}(\mu = 0.0, \sigma = 0.1)$ auf die Aktorausgabe addiert und entsprechend der maximalen Ausgabe $\boldsymbol{u}_{max}$ skaliert. Diese einfache Methode stabilisierte den Lernvorgang und wird deshalb – auch aufgrund der Einfachheit – bevorzugt.

Algorithmus 7 Deep-Deterministic-Policy-Gradient (DDPG) [36]

Initialisierung des Datensatzes $\mathcal{D}$
Aufwärmphase – Steuerung wird für bestimmte Anzahl Zeitschritte zufällig bestimmt
Zufällige Initialisierung des Kritiker- und Aktor-Netzwerks $\hat{Q}^{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}_k, \boldsymbol{u}_k), \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k)$
Initialisierung der Target-Netzwerke $\hat{Q}^{\mu}_{\boldsymbol{\phi}'}$ und $\mu_{\boldsymbol{\theta}'}$ mit den Gewichten $\boldsymbol{\phi}' \leftarrow \boldsymbol{\phi}, \boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
for n Episoden **do** ▷ Lernvorgang
  Initialisierung des Zufallsprozesses $\mathcal{R}$ für die Erkundung
  $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$ ▷ Anfangszustand bestimmen
  for $k \in [0, N-1]$ **do**
    $\boldsymbol{u}_k = \mu_{\boldsymbol{\theta}_1}(\boldsymbol{x}_k) + \mathcal{R}_k$ ▷ Bestimmung der verrauschten Steuerung
    $\boldsymbol{x}_{k+1} \sim p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)$ ▷ Simulation des Systems
    $c_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$ ▷ Berechnung der inkrementellen Kosten
    $\mathcal{D} \leftarrow d_i = (\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}, c_k^{(i)}, \boldsymbol{x}_{k+1}^{(i)})$ ▷ Transitionsdaten speichern
    Stichprobe eines Minibatches $\mathcal{B} \subset \mathcal{D}$
    Setze $\forall d_i \in \mathcal{B}$:

$$y^{(i)} = \begin{cases} c_k^{(i)}, & \text{für } \boldsymbol{x}_{k+1}^{(i)} \in \mathbb{X}^-, \\ c_k^{(i)} + \gamma \hat{Q}^{\mu}_{\boldsymbol{\phi}'}(\boldsymbol{x}_{k+1}^{(i)}, \mu_{\boldsymbol{\theta}'}(\boldsymbol{x}_{k+1}^{(i)})) & \text{sonst.} \end{cases} \tag{4.8}$$

    Training des Kritikers $\hat{Q}^{\mu}_{\boldsymbol{\phi}}$ durch Minimierung des MSBE:

$$\nabla_{\boldsymbol{\phi}} \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(y^{(i)} - \hat{Q}^{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}) \right)^2 \right) \tag{4.9}$$

    Training des Aktors $\mu_{\boldsymbol{\theta}}$ durch Schätzung des DPG:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k^{(i)}) \nabla_{\boldsymbol{u}_k} \hat{Q}^{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k) \Big|_{\boldsymbol{u}_k = \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_k^{(i)})} \tag{4.10}$$

    Aktualisierung der Target-Netzwerke:

$$\boldsymbol{\phi}' \leftarrow \tau \boldsymbol{\phi} + (1 - \tau) \boldsymbol{\phi}' \tag{4.11a}$$
$$\boldsymbol{\theta}' \leftarrow \tau \boldsymbol{\theta} + (1 - \tau) \boldsymbol{\theta}' \tag{4.11b}$$

  end for
end for

Tabelle 1 – Hyperparameter von DDPG.

Hyperparameter	Wert	Wert in [36]	Anmerkung
Größe eines Minibatchs $\mathcal{B}$	100	64	$\lvert\mathcal{B}\rvert$
Größe des Datensatzes $\mathcal{D}$	1×10^6	1×10^6	$\lvert\mathcal{D}\rvert$
Aktor-Lernrate	1×10^{-4}	1×10^{-4}	Schrittweite für Optimierung
Kritiker-Lernrate	1×10^{-3}	1×10^{-3}	Schrittweite für Optimierung
Regularisierung Kritiker	1×10^{-2}	1×10^{-2}	L2-Regularisierung von $\boldsymbol{\phi}$
Target-Update-Rate τ	5×10^{-3}	1×10^{-3}	
Diskontierungsfaktor γ	0.99	0.99	
Aufwärmphase	1×10^4	64	Zeitschritte bis Training beginnt.

Kapitel 5

Differenzielle dynamische Programmierung (DDP) und iterativer linear-quadratischer Regler (iLQR)

Die differenzielle dynamische Programmierung (DDP) [40] und der iterative linear-quadratische Regler (iLQR) [35] sind Methoden zur Lösung des Optimalsteuerungsproblems (2.19) für nichtlineare dynamische Systeme durch sukzessive Approximation, welche auf der DP basieren. Das nichtlineare Optimierungsproblem (2.19) wird mittels sequentieller linear-quadratischer Optimierung gelöst.

Die Lösung von (2.19) mittels iterativer linear-quadratischer Regler (iLQR) ist sehr recheneffizient, weshalb das Verfahren in hochdynamischen NMPC-Anwendungen eingesetzt wird [46].

Basis für beide Verfahren sind deterministische nichtlineare dynamische Systeme der Form

$$\boldsymbol{x}_{k+1} = f_d(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad k = 0, 1, ..., N-1. \tag{5.1}$$

Die Kosten sind wie beim zeitdiskreten LQR in (2.28) gegeben durch:

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}\left(\boldsymbol{x}_k^{\mathrm{T}} S \boldsymbol{x}_k + \boldsymbol{u}_k^{\mathrm{T}} R \boldsymbol{u}_k + 2\boldsymbol{x}_k^{\mathrm{T}} P \boldsymbol{u}_k\right), \tag{5.2a}$$

$$c_N(\boldsymbol{x}_N) = \frac{1}{2}\boldsymbol{x}_N^{\mathrm{T}} S_N \boldsymbol{x}_N. \tag{5.2b}$$

Bei beiden Methoden wird grundsätzlich wie folgt vorgegangen:

1. Start mit $i := 0$, Wahl einer initialen Steuerfolge $\bar{\boldsymbol{U}}_i = \{\bar{\boldsymbol{u}}_0, , \bar{\boldsymbol{u}}_1, ..., \bar{\boldsymbol{u}}_{N-1}\}$

2. Bestimmung der zu $\bar{\boldsymbol{U}}_i$ korrespondierenden Zustandstrajekorie $\bar{\boldsymbol{X}}_i = \{\bar{\boldsymbol{x}}_0, ..., \bar{\boldsymbol{x}}_N\}$ ausgehend von $\boldsymbol{x}_0$ durch Vorwärtsrechnung der nichtlinearen Systemdynamik

3. Approximation von (2.19) mittels Taylor-Entwicklung von (2.10) und (5.1) um $\bar{\boldsymbol{U}}_i$ und $\bar{\boldsymbol{X}}_i$

4. Bestimmung der optimalen Rückführung für die Approximation mittels LQR-Entwurf

5. Vorwärtsrechnung der nichtlinearen Systemdynamik mit der bestimmten optimalen Rückführung. Daraus resultieren eine neue Steuerfolge $\bar{\boldsymbol{U}}_{i+1}$ und eine zugehörige Zustandstrajektorie $\bar{\boldsymbol{X}}_{i+1}$. Falls bestimmte Abbruchkriterien nicht erfüllt sind $\rightarrow$ $i := i + 1$ und weiter mit Schritt 3.

5.1 Detaillierte Beschreibung des iterativen linear-quadratischen Reglers (iLQR)

Im Folgenden sollen die Schritte 3 bis 5 des beschriebenen Algorithmus näher erläutert werden.

Schritt 3: Approximation von (2.19) mittels Taylor-Entwicklung

Der wesentliche Unterschied zwischen DDP und iLQR liegt in der Approximationsordnung der Systemdynamik (5.1). Beim iLQR werden die Terme zweiter Ordnung vernachlässigt, da dies eine wesentliche Effizienzsteigerung mit sich bringt, die den Genauigkeitsverlust rechtfertigt [35, S. 7]. Im Weiteren wird deshalb nur der iLQR betrachtet.

Ausgangspunkt für die Approximation ist (2.21) mit (5.1):

$$V_N(\boldsymbol{x}_N) = c_N(\boldsymbol{x}_N), \tag{5.3a}$$

$$V_k(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathbb{U}(\boldsymbol{x}_k)} c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) + V_{k+1}\Big(f_d(\boldsymbol{x}_k, \boldsymbol{u}_k)\Big), k = 0, 1, ..., N-1. \tag{5.3b}$$

Da (5.1) deterministisch ist entfällt gegenüber (2.21) die Bildung des Erwartungswertes.

Eine Taylor-Entwicklung von (2.28) und (5.1) um $\bar{\boldsymbol{U}}_i$ und $\bar{\boldsymbol{X}}_i$ resultiert in:

$$\begin{aligned} c_N(\boldsymbol{x}_N) \approx\ & c_N(\bar{\boldsymbol{x}}_N) + \frac{\partial c_N}{\partial \boldsymbol{x}_N}(\bar{\boldsymbol{x}}_N)(\boldsymbol{x}_N - \bar{\boldsymbol{x}}_N) \\ & + \frac{1}{2}(\boldsymbol{x}_N - \bar{\boldsymbol{x}}_N)^{\mathrm{T}} \frac{\partial^2 c_N}{\partial \boldsymbol{x}_N^2}(\bar{\boldsymbol{x}}_N)(\boldsymbol{x}_N - \bar{\boldsymbol{x}}_N) + \text{T.h.O.}, \end{aligned} \tag{5.4a}$$

$$\begin{aligned} c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) \approx\ & c_k(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \frac{\partial c_k}{\partial \boldsymbol{x}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \frac{\partial c_k}{\partial \boldsymbol{u}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)(\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k) \\ & + \frac{1}{2}(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)^{\mathrm{T}} \frac{\partial^2 c_k}{\partial \boldsymbol{x}_k^2}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \frac{1}{2}(\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k)^{\mathrm{T}} \frac{\partial^2 c_k}{\partial \boldsymbol{u}_k^2}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)(\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k) \\ & + \frac{1}{2}(\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k)^{\mathrm{T}} \frac{\partial^2 c_k}{\partial \boldsymbol{x}_k \partial \boldsymbol{u}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \text{T.h.O.}, \end{aligned} \tag{5.4b}$$

sowie

$$f_d(\boldsymbol{x}_k, \boldsymbol{u}_k) \approx f_d(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \frac{\partial f_d}{\partial \boldsymbol{x}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \frac{\partial f_d}{\partial \boldsymbol{u}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)(\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k). \tag{5.5}$$

Aus (2.28) folgt für die Ableitungen in (5.4):

$$\frac{\partial c_N}{\partial \boldsymbol{x}_N}(\bar{\boldsymbol{x}}_N) = \bar{\boldsymbol{x}}_N^{\mathrm{T}} S_N =: \boldsymbol{c}_{x,N}^{\mathrm{T}}, \qquad \frac{\partial^2 c_N}{\partial \boldsymbol{x}_N^2}(\bar{\boldsymbol{x}}_N) = S_N =: C_{xx,N}, \tag{5.6a}$$

$$\frac{\partial c_k}{\partial \boldsymbol{x}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) = \bar{\boldsymbol{x}}_k^{\mathrm{T}} S + \bar{\boldsymbol{u}}_k^{\mathrm{T}} P^{\mathrm{T}} =: \boldsymbol{c}_{x,k}^{\mathrm{T}}, \qquad \frac{\partial c_k}{\partial \boldsymbol{u}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) = \bar{\boldsymbol{u}}_k^{\mathrm{T}} R + \bar{\boldsymbol{x}}_k^{\mathrm{T}} P =: \boldsymbol{c}_{u,k}^{\mathrm{T}} \tag{5.6b}$$

$$\frac{\partial^2 c_k}{\partial \boldsymbol{x}_k^2}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) = S =: C_{xx}, \qquad \frac{\partial^2 c_k}{\partial \boldsymbol{u}_k^2}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) = R =: C_{uu}, \tag{5.6c}$$

$$\frac{\partial^2 c_k}{\partial \boldsymbol{x}_k \partial \boldsymbol{u}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) = P^{\mathrm{T}} =: C_{xu}, \qquad \frac{\partial^2 c_k}{\partial \boldsymbol{u}_k \partial \boldsymbol{x}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) = P =: C_{ux} = C_{xu}^{\mathrm{T}}. \tag{5.6d}$$

Mit den Definitionen

$$\tilde{\boldsymbol{x}}_k := \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k \tag{5.7}$$

und

$$\tilde{\boldsymbol{u}}_k := \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k \tag{5.8}$$

sowie (5.6) folgt für die Reihenentwicklung in (5.4) somit:

$$\tilde{c}_N(\boldsymbol{x}_N) := c_N(\boldsymbol{x}_N) - \underbrace{c_N(\bar{\boldsymbol{x}}_N)}_{=:\bar{c}_N} \approx \frac{1}{2}\tilde{\boldsymbol{x}}_N^{\mathrm{T}} C_{xx,N} \tilde{\boldsymbol{x}}_N + \boldsymbol{c}_{x,N}^{\mathrm{T}} \tilde{\boldsymbol{x}}_N, \tag{5.9a}$$

$$\begin{aligned} \tilde{c}_k(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k) :=\ & c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) - \underbrace{c_k(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}_{=:\bar{c}_k}, \\ \approx\ & \frac{1}{2}\left(\tilde{\boldsymbol{x}}_k^{\mathrm{T}} C_{xx} \tilde{\boldsymbol{x}}_k + \tilde{\boldsymbol{u}}_k^{\mathrm{T}} C_{uu} \tilde{\boldsymbol{u}}_k + \tilde{\boldsymbol{x}}_k^{\mathrm{T}} C_{xu} \tilde{\boldsymbol{u}}_k + \tilde{\boldsymbol{u}}_k^{\mathrm{T}} C_{ux} \tilde{\boldsymbol{x}}_k\right) + \boldsymbol{c}_{x,k}^{\mathrm{T}} \tilde{\boldsymbol{x}}_k + \boldsymbol{c}_{u,k}^{\mathrm{T}} \tilde{\boldsymbol{u}}_k. \end{aligned} \tag{5.9b}$$

Im Gegensatz zu (2.28) weist (5.9) je einen zusätzlichen linearen Term in $\tilde{\boldsymbol{x}}_k$ und $\tilde{\boldsymbol{u}}_k$ auf.

Mit

$$A_k := \frac{\partial f_d}{\partial \boldsymbol{x}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{5.10}$$

und

$$B_k := \frac{\partial f_d}{\partial \boldsymbol{u}_k}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{5.11}$$

ergibt sich die Approximation der Systemdynamik als zeitvariantes lineares System

$$\tilde{f}_d(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k) = \tilde{\boldsymbol{x}}_{k+1} := f_d(\boldsymbol{x}_k, \boldsymbol{u}_k) - f_d(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \approx A_k \tilde{\boldsymbol{x}}_k + B_k \tilde{\boldsymbol{u}}_k. \tag{5.12}$$

Mit (5.12) und (5.9) folgt für (5.3) in den Koordinaten $\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k$:

$$\tilde{V}_N(\tilde{\boldsymbol{x}}_N) = \tilde{c}_N(\tilde{\boldsymbol{x}}_N), \tag{5.13a}$$

$$\tilde{V}_k(\tilde{\boldsymbol{x}}_k) = \min_{\tilde{\boldsymbol{u}}_k \in \mathbb{U}(\tilde{\boldsymbol{x}}_k)} \tilde{c}_k(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k) + \tilde{V}_{k+1}\Big(\tilde{f}_d(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k)\Big), \quad k = 0, 1, ..., N-1. \tag{5.13b}$$

Ergebnis der Approximation ist ein Optimalsteuerungsproblem mit quadratischer Kostenfunktion (5.9) und linearem zeitvariantem System (5.12), welches mittels LQR-Entwurf gelöst werden kann.

Schritt 4: Rückwärtsrechnung - Bestimmung der optimalen Rückführung für die Approximation mittels LQR-Entwurf

Um nun die optimale Rückführung für die Approximation (5.13) zu bestimmen, wird wie in Abschnitt 2.1.5 verfahren. Ausgehend von den Endkosten $\tilde{V}_N(\tilde{\boldsymbol{x}}_N)$ wird eine Rückwärtsrechnung auf Basis der DP durchgeführt, welche eine Rückführung der Form

$$\tilde{\boldsymbol{u}}_k^* = \tilde{K}_k \tilde{\boldsymbol{x}}_k + \tilde{\boldsymbol{k}}_k, \quad k = 0, ..., N-1 \tag{5.14}$$

zum Ergebnis hat.

$$\tilde{V}_N(\tilde{\boldsymbol{x}}_N) \overset{(5.9)}{=} \boldsymbol{c}_{x,N}^{\mathrm{T}} \tilde{\boldsymbol{x}}_N + \frac{1}{2} \tilde{\boldsymbol{x}}_N^{\mathrm{T}} C_{xx,N} \tilde{\boldsymbol{x}}_N, \tag{5.15a}$$

$$\tilde{V}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}) = \min_{\tilde{\boldsymbol{u}}_{N-1} \in \mathbb{U}(\tilde{\boldsymbol{x}}_{N-1})} \tilde{c}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}, \tilde{\boldsymbol{u}}_{N-1}) + \tilde{V}_N\Big(\tilde{f}_d(\tilde{\boldsymbol{x}}_{N-1}, \tilde{\boldsymbol{u}}_{N-1})\Big). \tag{5.15b}$$

Einsetzen der Systemgleichung (5.12) liefert die Bewertungsfunktion:

$$\begin{aligned}
\tilde{Q}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}, \tilde{\boldsymbol{u}}_{N-1}) &= \tilde{c}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}, \tilde{\boldsymbol{u}}_{N-1}) + \tilde{V}_N(A_{N-1}\tilde{\boldsymbol{x}}_{N-1} + B_{N-1}\tilde{\boldsymbol{u}}_{N-1}), && \text{(5.16a)}\\
&= \boldsymbol{c}_{x,N-1}^{\mathrm{T}}\tilde{\boldsymbol{x}}_{N-1} + \boldsymbol{c}_{u,N-1}^{\mathrm{T}}\tilde{\boldsymbol{u}}_{N-1} && \text{(5.16b)}\\
&\quad + \frac{1}{2}\Big(\tilde{\boldsymbol{x}}_{N-1}^{\mathrm{T}} C_{xx}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{u}}_{N-1}^{\mathrm{T}} C_{uu}\tilde{\boldsymbol{u}}_{N-1} && \text{(5.16c)}\\
&\quad + \tilde{\boldsymbol{x}}_{N-1}^{\mathrm{T}} C_{xu}\tilde{\boldsymbol{u}}_{N-1} + \tilde{\boldsymbol{u}}_{N-1}^{\mathrm{T}} C_{ux}\tilde{\boldsymbol{x}}_{N-1}\Big)\\
&\quad + \boldsymbol{c}_{x,N}^{\mathrm{T}}(A_{N-1}\tilde{\boldsymbol{x}}_{N-1} + B_{N-1}\tilde{\boldsymbol{u}}_{N-1})\\
&\quad + \frac{1}{2}(A_{N-1}\tilde{\boldsymbol{x}}_{N-1} + B_{N-1}\tilde{\boldsymbol{u}}_{N-1})^{\mathrm{T}} C_{xx,N}(A_{N-1}\tilde{\boldsymbol{x}}_{N-1} + B_{N-1}\tilde{\boldsymbol{u}}_{N-1}),\\
&= \frac{1}{2}\Big(\tilde{\boldsymbol{x}}_{N-1}^{\mathrm{T}} Q_{xx,N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{x}}_{N-1}^{\mathrm{T}} Q_{xu,N-1}\tilde{\boldsymbol{u}}_{N-1} && \text{(5.16d)}\\
&\quad + \tilde{\boldsymbol{u}}_{N-1}^{\mathrm{T}} Q_{ux,N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{u}}_{N-1}^{\mathrm{T}} Q_{uu,N-1}\tilde{\boldsymbol{u}}_{N-1}\Big) && \text{(5.16e)}\\
&\quad + \boldsymbol{q}_{x,N-1}^{\mathrm{T}}\tilde{\boldsymbol{x}}_{N-1} + \boldsymbol{q}_{u,N-1}^{\mathrm{T}}\tilde{\boldsymbol{u}}_{N-1},
\end{aligned}$$

mit

$$\begin{aligned}
Q_{xx,N-1} &:= C_{xx} + A_{N-1}^{\mathrm{T}} C_{xx,N} A_{N-1}, && \text{(5.17a)}\\
Q_{xu,N-1} &:= C_{xu} + A_{N-1}^{\mathrm{T}} C_{xx,N} B_{N-1}, && \text{(5.17b)}\\
Q_{ux,N-1} &:= C_{ux} + B_{N-1}^{\mathrm{T}} C_{xx,N} A_{N-1} = Q_{xu,N-1}^{\mathrm{T}}, && \text{(5.17c)}\\
Q_{uu,N-1} &:= C_{uu} + B_{N-1}^{\mathrm{T}} C_{xx,N} B_{N-1}, && \text{(5.17d)}\\
\boldsymbol{q}_{x,N-1}^{\mathrm{T}} &:= \boldsymbol{c}_{x,N-1}^{\mathrm{T}} + \boldsymbol{c}_{x,N}^{\mathrm{T}} A_{N-1}, && \text{(5.17e)}\\
\boldsymbol{q}_{u,N-1}^{\mathrm{T}} &:= \boldsymbol{c}_{u,N-1}^{\mathrm{T}} + \boldsymbol{c}_{x,N}^{\mathrm{T}} B_{N-1}. && \text{(5.17f)}
\end{aligned}$$

Mittels Minimierung von (5.16) kann der optimale Eingang

$$\tilde{\boldsymbol{u}}_{N-1}^{*} = \arg\min_{\tilde{\boldsymbol{u}}_{N-1}} \tilde{Q}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}\tilde{\boldsymbol{u}}_{N-1}) \tag{5.18}$$

bestimmt werden.

Dazu wird die notwendige[1] Bedingung für Optimalität verwendet [5, S. 13]:

$$\begin{aligned}
0 &\overset{!}{=} \frac{\partial}{\partial \tilde{\boldsymbol{u}}_{N-1}} \tilde{Q}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}, \tilde{\boldsymbol{u}}_{N-1}^{*}), && \text{(5.19a)}\\
\Leftrightarrow \quad 0 &= \boldsymbol{q}_{u,N-1} + Q_{uu,N-1}\tilde{\boldsymbol{u}}_{N-1}^{*} + \quad Q_{xu,N-1}^{\mathrm{T}}\tilde{\boldsymbol{x}}_{N-1}, && \text{(5.19b)}\\
\Leftrightarrow \quad \tilde{\boldsymbol{u}}_{N-1}^{*} &= -Q_{uu,N-1}^{-1}(Q_{xu,N-1}^{\mathrm{T}}\tilde{\boldsymbol{x}}_{N-1} + \boldsymbol{q}_{u,N-1}). && \text{(5.19c)}
\end{aligned}$$

Die optimale Rückführung π^* ergibt sich im Gegensatz zu Abschnitt 2.1.5, aufgrund des linearen Terms in (5.9) zu:

$$\tilde{\boldsymbol{u}}_{N-1}^{*} = \tilde{K}_{N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{k}}_{N-1}, \tag{5.20}$$

[1] Für konvexe Funktionen ist diese Bedingung hinreichend [5, S. 14].

mit

$$\tilde{K}_{N-1} := -Q_{uu,N-1}^{-1} Q_{xu,N-1}^{\mathrm{T}} \tag{5.21a}$$

und

$$\tilde{\boldsymbol{k}}_{N-1} := -Q_{uu,N-1}^{-1} \boldsymbol{q}_{u,N-1}. \tag{5.21b}$$

Dass die Rückführung (5.20) tatsächlich optimal ist folgt aus der hinreichenden Bedingung für Optimalität [5, S. 13]:

$$\frac{\partial^2}{\partial \tilde{\boldsymbol{u}}_{N-1}^2} \tilde{Q}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}, \tilde{\boldsymbol{u}}_{N-1}^*) \overset{!}{=} 0, \tag{5.22a}$$

$$\Leftrightarrow \qquad Q_{uu,N-1} \succ 0. \tag{5.22b}$$

Die optimale Steuerung kann nun in die Bewertungsfunktion eingesetzt werden:

$$\begin{aligned}
\tilde{V}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}) &= \tilde{Q}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}, \tilde{\boldsymbol{u}}_{N-1}^*), && (5.23a)\\
&\overset{(5.20)}{=} \tilde{Q}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}, \tilde{K}_{N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{k}}_{N-1}), && (5.23b)\\
&\overset{(5.16)}{=} \frac{1}{2}\Big(\tilde{\boldsymbol{x}}_{N-1}^{\mathrm{T}} Q_{xx,N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{x}}_{N-1}^{\mathrm{T}} Q_{xu,N-1}(\tilde{K}_{N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{k}}_{N-1}) && (5.23c)\\
&\quad + (\tilde{K}_{N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{k}}_{N-1})^{\mathrm{T}} Q_{ux,N-1}\tilde{\boldsymbol{x}}_{N-1} && (5.23d)\\
&\quad + (\tilde{K}_{N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{k}}_{N-1})^{\mathrm{T}} Q_{uu,N-1}(\tilde{K}_{N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{k}}_{N-1})\Big)\\
&\quad + \boldsymbol{q}_{x,N-1}^{\mathrm{T}}\tilde{\boldsymbol{x}}_{N-1} + \boldsymbol{q}_{u,N-1}^{\mathrm{T}}(\tilde{K}_{N-1}\tilde{\boldsymbol{x}}_{N-1} + \tilde{\boldsymbol{k}}_{N-1}).
\end{aligned}$$

Somit ergibt sich $\tilde{V}_{N-1}(\tilde{\boldsymbol{x}}_{N-1})$ zu:

$$\tilde{V}_{N-1}(\tilde{\boldsymbol{x}}_{N-1}) = \frac{1}{2}\tilde{\boldsymbol{x}}_{N-1}^{\mathrm{T}} V_{xx,N-1}\tilde{\boldsymbol{x}}_{N-1} + \boldsymbol{v}_{x,N-1}^{\mathrm{T}}\tilde{\boldsymbol{x}}_{N-1} + v_{\text{konst.},N-1}, \tag{5.24}$$

mit

$$V_{xx,N-1} := Q_{xx,N-1} + \tilde{K}_{N-1}^{\mathrm{T}} Q_{ux,N-1} + Q_{xu,N-1}\tilde{K}_{N-1} + \tilde{K}_{N-1}^{\mathrm{T}} Q_{uu,N-1}\tilde{K}_{N-1}, \tag{5.25a}$$

$$\boldsymbol{v}_{x,N-1} := \boldsymbol{q}_{x,N-1} + \tilde{K}_{N-1}^{\mathrm{T}}\boldsymbol{q}_{u,N-1} + Q_{xu,N-1}\tilde{\boldsymbol{k}}_{N-1}^{\mathrm{T}} + \tilde{K}_{N-1}^{\mathrm{T}} Q_{uu,N-1}\tilde{\boldsymbol{k}}_{N-1}, \tag{5.25b}$$

$$v_{\text{konst.},N-1} := \frac{1}{2}\tilde{\boldsymbol{k}}_{N-1}^{\mathrm{T}} Q_{uu,N-1}\tilde{\boldsymbol{k}}_{N-1} + \boldsymbol{q}_{u,N-1}^{\mathrm{T}}\tilde{\boldsymbol{k}}_{N-1}. \tag{5.25c}$$

Nun lassen sich in Analogie zu Abschnitt 2.1.5 Riccati-Gleichungen in $\boldsymbol{v}_{x,k}$ und $V_{xx,k}$ mit $k = N, ..., 0$ und den Anfangswerten $\boldsymbol{v}_{x,N} := \boldsymbol{c}_{x,N}, V_{xx,N} := C_{xx,N}$ aufstellen. Dazu

werden zunächst die Matrizen und Vektoren von $\tilde{Q}_k(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k)$ bestimmt [70, Gl. 4]:

$$Q_{xx,k} := C_{xx} + A_k^{\mathrm{T}} V_{xx,k+1} A_k, \tag{5.26a}$$
$$Q_{xu,k} := C_{xu} + A_k^{\mathrm{T}} V_{xx,k+1} B_k, \tag{5.26b}$$
$$Q_{ux,k} := C_{ux} + B_k^{\mathrm{T}} V_{xx,k+1} A_k = Q_{xu,k}^{\mathrm{T}}, \tag{5.26c}$$
$$Q_{uu,k} := C_{uu} + B_k^{\mathrm{T}} V_{xx,k+1} B_k, \tag{5.26d}$$
$$\boldsymbol{q}_{x,k}^{\mathrm{T}} := \boldsymbol{c}_{x,k}^{\mathrm{T}} + \boldsymbol{v}_{x,k+1}^{\mathrm{T}} A_k, \tag{5.26e}$$
$$\boldsymbol{q}_{u,k}^{\mathrm{T}} := \boldsymbol{c}_{u,k}^{\mathrm{T}} + \boldsymbol{v}_{x,k+1}^{\mathrm{T}} B_k. \tag{5.26f}$$

Daraus kann im nächsten Schritt die zeitvariante Rückführung bestimmt werden:

$$\tilde{K}_k := -Q_{uu,k}^{-1} Q_{ux,k}, \tag{5.27a}$$
$$\tilde{\boldsymbol{k}}_k := -Q_{uu,k}^{-1} \boldsymbol{q}_{u,k}. \tag{5.27b}$$

Die resultierenden Folgen $\tilde{K}_{[0:N-1]} = \{\tilde{K}_0, ..., \tilde{K}_{N-1}\}$ sowie $\tilde{\boldsymbol{k}}_{[0:N-1]} = \{\tilde{\boldsymbol{k}}_0, ..., \tilde{\boldsymbol{k}}_{N-1}\}$ bilden die optimale Rückführung für (5.12) [70, Gl. 5b]:

$$\tilde{\boldsymbol{u}}_k^* = \tilde{K}_k \tilde{\boldsymbol{x}}_k + \tilde{\boldsymbol{k}}_k, \quad k = 0, 1, ..., N-1, \tag{5.28}$$

sofern (5.19) und (5.22) für alle $k \in [0, N]$ erfüllt sind.

Mit (5.26) und (5.27) können nun die Riccati-Gleichungen aufgestellt werden [69, Gl. 11]:

$$V_{xx,k} := Q_{xx,k} + \tilde{K}_k^{\mathrm{T}} Q_{ux,k} + Q_{xu,k} \tilde{K}_k + \tilde{K}_k^{\mathrm{T}} Q_{uu,k} \tilde{K}_k, \tag{5.29a}$$
$$\boldsymbol{v}_{x,k} := \boldsymbol{q}_{x,k} + \tilde{K}_k^{\mathrm{T}} \boldsymbol{q}_{u,k} + Q_{xu,k} \tilde{\boldsymbol{k}}_k + \tilde{K}_k^{\mathrm{T}} Q_{uu,k} \tilde{\boldsymbol{k}}_k, \tag{5.29b}$$
$$v_{\text{konst.},k} := \frac{1}{2} \tilde{\boldsymbol{k}}_k^{\mathrm{T}} Q_{uu,k} \tilde{\boldsymbol{k}}_k + \boldsymbol{q}_{u,k}^{\mathrm{T}} \tilde{\boldsymbol{k}}_k. \tag{5.29c}$$

Die Restkosten $\tilde{V}_k(\tilde{\boldsymbol{x}}_k)$ resultieren demnach aus der folgenden Summe:

$$\tilde{V}_k(\tilde{\boldsymbol{x}}_k) = \frac{1}{2} \tilde{\boldsymbol{x}}_k^{\mathrm{T}} V_{xx,k} \tilde{\boldsymbol{x}}_k + \boldsymbol{v}_{x,k}^{\mathrm{T}} \tilde{\boldsymbol{x}}_k + v_{\text{konst.},k}. \tag{5.30}$$

Schritt 5: Vorwärtsrechnung der nichtlinearen Systemdynamik mit der bestimmten optimalen Rückführung.

Um die neue Steuerfolge $\bar{\boldsymbol{U}}_{i+1}$ und zugehörige Zustandstrajektorie $\bar{\boldsymbol{X}}_{i+1}$ von (5.1) zu bestimmen wird eine Vorwärtsrechnung unter Nutzung von (5.28) durchgeführt. Damit (5.28) auf das System (5.1) angewandt werden kann, muss (5.28) nach $\boldsymbol{u}_k$ umgestellt werden:

$$\boldsymbol{u}_k = \tilde{K}_k (\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \tilde{\boldsymbol{k}}_k + \bar{\boldsymbol{u}}_k, \quad k = 0, 1, ..., N-1. \tag{5.31}$$

Ausgehend von $\boldsymbol{x}_0$ können $\bar{\boldsymbol{U}}_{i+1}$ und $\bar{\boldsymbol{X}}_{i+1}$ nach dem folgenden Schema berechnet werden:

$$\boldsymbol{u}_0 = \tilde{K}_0(\boldsymbol{x}_0 - \bar{\boldsymbol{x}}_0) + \tilde{\boldsymbol{k}}_0 + \bar{\boldsymbol{u}}_0 \quad \rightarrow \quad \boldsymbol{x}_1 = f_d(\boldsymbol{x}_0, \boldsymbol{u}_0), \tag{5.32a}$$

$$\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots$$

$$\boldsymbol{u}_{N-1} = \tilde{K}_{N-1}(\boldsymbol{x}_{N-1} - \bar{\boldsymbol{x}}_{N-1}) + \tilde{\boldsymbol{k}}_{N-1} + \bar{\boldsymbol{u}}_{N-1} \quad \rightarrow \quad \boldsymbol{x}_N = f_d(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}). \tag{5.32b}$$

Somit gilt

$$\bar{\boldsymbol{U}}_{i+1} = \{\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_{N-1}\}, \tag{5.33a}$$

$$\bar{\boldsymbol{X}}_{i+1} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_{N-1}, \boldsymbol{x}_N\}. \tag{5.33b}$$

Unter der Voraussetzung, dass der Algorithmus noch nicht konvergiert ist, werden

$$\bar{\boldsymbol{U}}_i := \bar{\boldsymbol{U}}_{i+1}, \tag{5.34a}$$

$$\bar{\boldsymbol{X}}_i := \bar{\boldsymbol{X}}_{i+1}, \tag{5.34b}$$

gesetzt und zu Schritt 3 übergegangen.

5.2 Abbruchkriterien

Der Algorithmus ist konvergiert, wenn eines der folgenden Kriterien erfüllt ist:

1. Die Restkosten J haben sich nach einer Iteration nicht weiter verringert. Der Algorithmus hat ein lokales Minimum der Kostenfunktion bestimmt:

$$J_{i+1} - J_i < \varepsilon_J, \quad \varepsilon_J > 0. \tag{5.35}$$

2. Die Betragsnorm des Gradienten der Kostenfunktion ist klein:

$$\max_{k \in [0, N-1]} \left| \frac{|\boldsymbol{k}_k|}{|\bar{\boldsymbol{u}}_k| + 1} \right| < \varepsilon_{\text{grad.}} \tag{5.36}$$

Bemerkung 5.1. *Wenn die bestimmten $\boldsymbol{k}_k$ die beschriebene Bedingung erfüllen (vernachlässigbar klein sind), wird eine erneute Vorwärtsrechnung (5.32) ausgehend von $\boldsymbol{x}_0$ auf die selbe Trajektorie des vorherigen Iterationsschrittes führen.*

5.3 Numerische Probleme

Durch die Approximation ergeben sich numerische Probleme, die beim Anwenden des beschriebenen Algorithmus im schlimmsten Fall zu Divergenz führen können. Es können die folgenden Probleme auftreten:

1. Q_{uu} ist nicht positiv definit,
2. Die Restkosten erhöhten sich. $J_{i+1} > J_i$.

Im folgenden sollen die Probleme, sowie die in [70] vorgeschlagenen Lösungsansätze erläutert werden.

Problem 1 - Rückwärtsrechnung Problem 1 hat zur Folge, dass die hinreichende Bedingung für Optimalität nicht mehr erfüllt ist. Die quadratische Approximation der Kostenfunktion ist somit nicht mehr konvex und die bestimmte Steuerung $\tilde{\boldsymbol{u}}_k^*$ nicht minimierend.

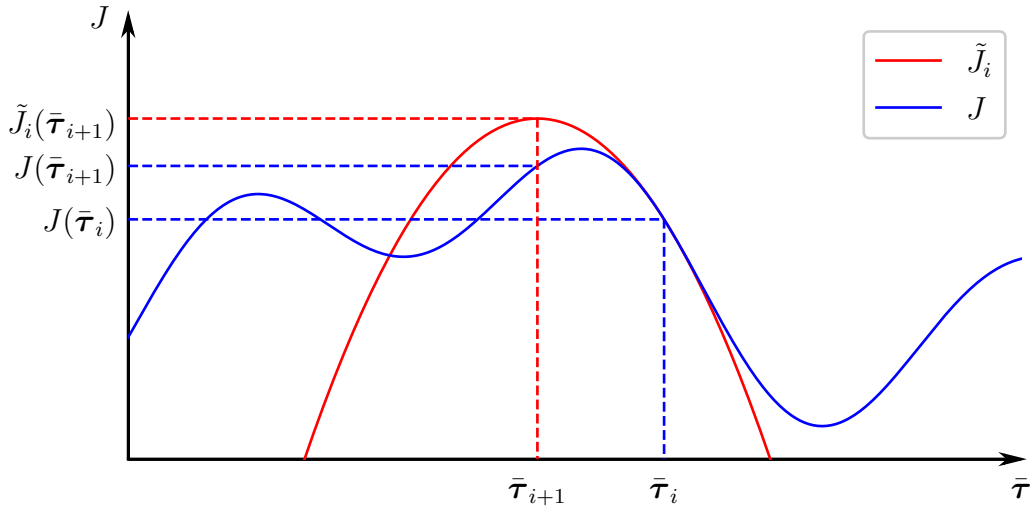

Abbildung 9 – Problem 1: Die Kostenfunktion J wurde im *Punkt* $\bar{\boldsymbol{\tau}}_i := \{\bar{X}_i, \bar{U}_i\}$ durch $\tilde{J}_i$ quadratisch approximiert. Durch ein oder mehrere auftretende $Q_{uu,k} \prec 0$ ist $\tilde{J}_i$ jedoch nicht mehr konvex. Die Rückführung, die mittels Rückwertrechnung (Schritt 4) bestimmt wird ist nicht mehr optimal und führt bei Anwendung von (5.32) auf einen *Punkt* $\approx \bar{\boldsymbol{\tau}}_{i+1}$. Für diesen haben sich die Kosten gegenüber dem vorigen Iterationsschritt erhöht: $J(\bar{\boldsymbol{\tau}}_{i+1}) > J(\bar{\boldsymbol{\tau}}_i)$. Durch die Regularisierung wird erzwungen, dass $\tilde{J}_i$ konvex ist.

Um diesem Problem zu begegnen, muss zur Laufzeit überprüft werden, ob $Q_{uu,k}$ positiv definit ist, also nur positive Eigenwerte aufweist.[2]

[2] Anstatt die Eigenwerte zu bestimmen ist es rechentechnisch günstiger zu überprüfen, ob eine Cholesky-Zerlegung von $Q_{uu,k}$ möglich ist.

Ist dies nicht der Fall kann eine positive Definitheit erzwungen werden, in dem ein Regularisierungsterm addiert wird [69, Gl. 9][3]:

$$Q_{uu,k,\text{reg.}} := Q_{uu,k} + \eta I_{m\times m}. \tag{5.37}$$

$I_{m\times m}$ bezeichnet eine Einheitsmatrix der Dimension $m \times m$. Der Regularisierungsfaktor η wird zu Beginn gleich 1 gesetzt. Tritt bei der Rückwärtsrechnung eine nicht positiv definite Matrix $Q_{uu,k}$ auf, wird η erhöht und die Rechnung erneut durchgeführt. Dieser Vorgang wird wiederholt bis $Q_{uu,k,\text{reg.}} \succ 0$ für alle $k = 0, 1..., N-1$ gilt. Tritt der Fall nicht ein, kann η wieder verkleinert werden.

Problem 2 - Vorwärtsrechnung Bei der Anwendung von (5.31) kann es vorkommen, dass sich die Kosten J gegenüber dem vorigen Iterationsschritt erhöhen. Dies resultiert daraus, das (5.31) nur optimal für die linear-quadratische Approximation ist.

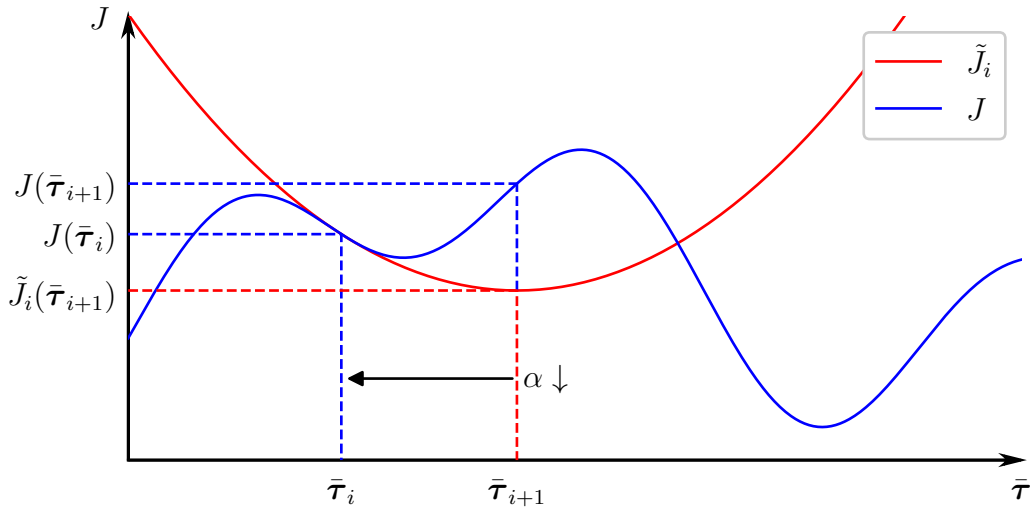

Abbildung 10 – Problem 2: Die Kostenfunktion J wurde im *Punkt* $\bar{\boldsymbol{\tau}}_i := \{\bar{X}_i, \bar{U}_i\}$ durch $\tilde{J}_i$ quadratisch approximiert. Die optimale Rückführung für die linear-quadratische Approximation führt bei Anwendung von (5.32) auf einen *Punkt* $\approx \bar{\boldsymbol{\tau}}_{i+1}$. Für $\tilde{J}_i$ stellt $\bar{\boldsymbol{\tau}}_{i+1}$ das Minimum dar. Es gilt demnach $\bar{\boldsymbol{\tau}}_{i+1} = \arg\min_{\bar{\boldsymbol{\tau}}} \tilde{J}_i(\bar{\boldsymbol{\tau}})$. Die tatsächliche Kostenfunktion J hat sich jedoch erhöht: $J(\bar{\boldsymbol{\tau}}_{i+1}) > J(\bar{\boldsymbol{\tau}}_i)$. Der Faktor α wird nun sukzessive verringert, bis $J_{i+1} < J_i$ gilt. Für $\alpha = 0$ gilt $\bar{\boldsymbol{\tau}}_i \equiv \bar{\boldsymbol{\tau}}_{i+1}$ und somit $J_i \equiv J_{i+1}$.

Als heuristisches Verfahren wird deshalb eine Liniensuche durchgeführt, um eine Verringerung der Kosten zu gewährleisten. Dazu wird in (5.31) ein Parameter $\alpha \in [0, 1]$

[3] In [70] wird noch eine weitere Möglichkeit zur Regularisierung vorgeschlagen.

eingeführt:

$$\boldsymbol{u}_k = \tilde{K}_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \alpha\tilde{\boldsymbol{k}}_k + \bar{\boldsymbol{u}}_k, \quad k = 0, 1, ..., N-1. \tag{5.38}$$

Ausgehend von $\alpha = 1$ wird dieser solange verringert und die Vorwärtsrechnung erneut durchgeführt, bis die Bedingung $J_{i+1} < J_i$ erfüllt ist. Dass es immer ein $\alpha \in [0, 1)$ gibt, für das $J_{i+1} < J_i$ gilt, folgt aus der erzwungenen Konvexität der Kostenfunktion. Für $\alpha = 1$ ist (5.38) identisch (5.31). Für $\alpha = 0$ vereinfacht sich (5.38) zu

$$\boldsymbol{u}_k = \tilde{K}_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \bar{\boldsymbol{u}}_k, \quad k = 0, 1, ..., N-1. \tag{5.39}$$

Unter der Bedingung $\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0$ vereinfacht sich diese zu:

$$\boldsymbol{u}_k = \bar{\boldsymbol{u}}_k, \quad k = 0, 1, ..., N-1. \tag{5.40}$$

Damit entspricht $\bar{\boldsymbol{U}}_{i+1}$ gleich $\bar{\boldsymbol{U}}_i$. Eine Vorwärtsrechnung führt somit auf die Zustandstrajektorie des vorigen Iterationsschrittes.

Bemerkung 5.2. *In [70] wird (5.29c) mit α als Abschätzung für die Kostenverringerung innerhalb einer Iteration des Algorithmus genutzt:*

$$\Delta\hat{J} := -\sum_{k=0}^{N-1} \alpha(\boldsymbol{q}_{u,k}^T\boldsymbol{k}_k + \alpha\boldsymbol{k}_k^T Q_{uu,k}\boldsymbol{k}_k). \tag{5.41}$$

Die Autoren verweisen auf [40].

5.4 Steuerbeschränkungen

Bei realen Systemen unterliegen die Steuergrößen Beschränkungen. Eine Heizquelle kann bspw. nur eine bestimmte Leistung liefern. Beim LQR-Entwurf kann es aber durch eine ungünstige Wahl der Gewichtsmatrizen dazu kommen, dass die Rückführung Werte außerhalb dieser physikalischen Grenzen für die Steuerung ausgibt. Den Eingang $\tilde{\boldsymbol{u}}_k$ einfach entsprechend der Grenzen zu beschneiden verschlechtert die Konvergenzeigenschaften des Algorithmus bedeutend. Deshalb wird in [70] ein Verfahren vorgestellt, das Steuerbeschränkungen $\boldsymbol{u} \in [\boldsymbol{u}_{\min}, \boldsymbol{u}_{\max}]$ explizit berücksichtigt. Dazu wird analog zum Pontryaginschen Maximumprinzip [19, S. 110] das beschränkte Optimierungsproblem

$$\tilde{\boldsymbol{k}}_k = \arg\min_{\tilde{\boldsymbol{u}}_k} \quad \tilde{Q}_k(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k), \tag{5.42a}$$

$$\text{u.B.v.} \quad \underline{\boldsymbol{b}} \leq \tilde{\boldsymbol{u}}_k \leq \overline{\boldsymbol{b}}, \tag{5.42b}$$

numerisch gelöst, anstatt (5.18) analytisch zu lösen. Das Problem (5.42) in expliziter Form lautet demnach:

$$\tilde{\boldsymbol{k}}_k = \arg\min_{\tilde{\boldsymbol{u}}_k} \quad \tilde{\boldsymbol{u}}_k^{\mathrm{T}} Q_{uu,k,\mathrm{reg.}} \tilde{\boldsymbol{u}}_k + \boldsymbol{q}_{u,k}^{\mathrm{T}} \tilde{\boldsymbol{u}}_k, \tag{5.43a}$$

$$\text{u.B.v.} \quad H\tilde{\boldsymbol{u}}_k \leq \boldsymbol{b}, \tag{5.43b}$$

mit

$$H := \begin{pmatrix} I_{m\times m} \\ -I_{m\times m} \end{pmatrix}, \qquad \boldsymbol{b} := \begin{pmatrix} \boldsymbol{u}_{\max} - \bar{\boldsymbol{u}}_k \\ \boldsymbol{u}_{\min} - \bar{\boldsymbol{u}}_k \end{pmatrix}. \tag{5.44}$$

Das Resultat der Optimierung ist die optimale Steuerung

$$\tilde{\boldsymbol{u}}_k^* := \tilde{\boldsymbol{k}}_k, \tag{5.45}$$

die im Gegensatz zur analytischen Lösung (5.28) keine Rückführung mehr enthält. Um diese Einschränkung nur dann durchzuführen, wenn sie auch nötig ist, wird (5.45) nur auf die Dimensionen von $\boldsymbol{u}_k$ angewandt, bei denen die Beschränkung aktiv ist. Die entsprechende Zeile in K_k aus (5.28) ist demnach eine Nullzeile.

Algorithmus 8 iLQR

$\bar{\boldsymbol{U}}_0$ ▷ Wahl der initialen Steuertrajektorie
$\bar{\boldsymbol{X}}_0$ aus (5.32) ▷ Vorwärtsrechnung
$i := 1$
while 1 **do** ▷ Optimierung
  while 1 **do** ▷ Rückwärtsrechnung
    $\boldsymbol{K}^i_{[0:N-1]}, \boldsymbol{k}^i_{[0:N-1]}$ aus (5.27) bzw. (5.42) ▷ Bestimmung der Rückführung.
    if any$(Q_{uu,k,\text{reg.}}) \prec 0$ **then** ▷ Rückwärtsrechnung nicht erfolgreich?
      $\eta \uparrow$ ▷ Regularisierungsfaktor erhöhen,
    else
      break ▷ Rückwärtsrechnung erfolgreich.
    end if
  end while
  for $\alpha = [1, ..., 0)$ **do** ▷ Liniensuche
    $\bar{\boldsymbol{U}}_{i+1}$, $\bar{\boldsymbol{X}}_{i+1}$, $J(\boldsymbol{x}_0, \bar{\boldsymbol{U}}_{i+1})$ aus (5.32) ▷ Vorwärtsrechnung
    $\Delta J := J(\boldsymbol{x}_0, \bar{\boldsymbol{U}}_{i+1}) - J(\boldsymbol{x}_0, \bar{\boldsymbol{U}}_i)$ ▷ Verringerung der Restkosten J
    $\Delta \hat{J} := -\sum_{k=0}^{N-1} \alpha(\boldsymbol{q}_{u,k}^T \boldsymbol{k}_k + \alpha \boldsymbol{k}_k^T Q_{uu,k} \boldsymbol{k}_k)$ ▷ Erwartete Verringerung
    if $\frac{\Delta J}{\Delta \hat{J}} > 0$ **then** ▷ Restkosten verringert?
      $\eta \downarrow$ ▷ Regularisierungsfaktor reduzieren
      if $\Delta J < \varepsilon_J$ **then** ▷ Geringe Kostenverringerung?
        break ▷ Konvergiert
      end if
      if $\nabla J < \varepsilon_{\text{grad.}}$ **then** ▷ Kleiner Gradient?
        break ▷ Konvergiert
      end if
    else ▷ Restkosten erhöht
      $\eta \uparrow$ ▷ Regularisierungsfaktor erhöhen.
      if $\eta > \eta_{\max}$ **then**
        break ▷ Divergiert
      end if
    end if
  end for
  $i := i + 1$
end while

Kapitel 6

Ergebnisse

Im folgenden Kapitel werden DDPG – als moderne Methode des bestärkenden Lernens – und iLQR – als modellbasierter Referenzansatz der Regelungs- und Steuerungstheorie – auf verschiedene regelungstechnische Problemstellungen angewandt und miteinander verglichen. DDPG wurde an dieser Stelle ausgewählt, da es im Gegensatz zu NFQ und DQN einen kontinuierlichen Eingang $\boldsymbol{u}_k$ ermöglicht.

Als Basis für die in Kapitel 2 erarbeiteten Grundlagen diente (2.1). Die untersuchten Methoden in Kapitel 4 und Kapitel 5 wurden im Rahmen dieser Arbeit zunächst jedoch nur auf deterministische Abtastsysteme der Form

$$\boldsymbol{x}_{k+1} = f_d(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{6.1}$$

angewandt, welche einen Spezialfall von (2.1) darstellen.

Abtastsystem Das Ergebnis der Modellbildung sind kontinuierliche nichtlineare Systeme der Form

$$\dot{\boldsymbol{x}} = f_c(\boldsymbol{x}, \boldsymbol{u}). \tag{6.2}$$

Um (6.2) auf (6.1) überführen zu können, wird vorausgesetzt, dass (6.2) für einen Anfangswert $\boldsymbol{x}_0$ und eine Anfangszeit t_0 mit einer lokal Lebesque-integrierbaren Funktion[1] $\boldsymbol{u}(\cdot)$ eine eindeutige Lösung $\boldsymbol{x}(t)$ hat [20, S. 16].

Das zeitdiskrete Abtastsystem lässt sich dann durch den Fluss φ von (6.2) über das Zeitintervall Δt ausdrücken [56, S. 26]:

$$\boldsymbol{x}_{k+1} = f_d(\boldsymbol{x}_k, \boldsymbol{u}_k) := \varphi_{\Delta t}(\boldsymbol{x}_k, \boldsymbol{u}_k). \tag{6.3}$$

Das Abtastsystem (6.3) lässt sich bspw. mit einem Runge-Kutta-Verfahren oder – wie hier dargestellt – mit einem Euler-Verfahren approximieren:

$$\boldsymbol{x}_{k+1} = f_d(\boldsymbol{x}_k, \boldsymbol{u}_k) \approx \boldsymbol{x}_k + \Delta t f_c(\boldsymbol{x}_k, \boldsymbol{u}_k). \tag{6.4}$$

[1] Dies ist gleichbedeutend mit: $\int_{\mathbb{R}_{\geq 0}} |u(t)| \mathrm{d}t < \infty$.

Für alle mechanischen nicht-flachen Systeme wurde eine partielle Linearisierung der nichtlinearen Systemdynamik durchgeführt [56, S. 123][28, S. 11], wodurch der Eingang $\boldsymbol{u}$ durch die Beschleunigungen der aktuierten generalisierten Koordinaten gegeben ist.

Leider war es Zeitgründen nicht mehr möglich, die untersuchten Methoden auf thermodynamische Gebäudemodelle anzuwenden. Die Schwierigkeit lag hier vor allem darin, dass diese Systeme zeitvarianten Störgrößen, wie bspw. der Außentemperatur unterliegen. Zudem werden die Kosten c_k in solchen Szenarien als zeitvariable Größen angenommen. Für realitätsnahe Untersuchungen bedarf es deshalb einer umfangreichen Erweiterung der entwickelten Software.

Überführung von stabilen in instabile Ruhelagen Die zu untersuchende Steuerungsaufgabe besteht darin, ein nichtlineares dynamisches System (6.1) aus einer stabilen in eine instabile Ruhelage zu überführen und dort zu stabilisieren.

Definition 6.1 (Ruhelage)**.** *Eine Ruhelage $\boldsymbol{x}_*$ des Systems (6.1) erfüllt die folgende Bedingung:*

$$\boldsymbol{x}_* = f_d(\boldsymbol{x}_*, \boldsymbol{u}_*), \tag{6.5}$$

wobei eine Steuerung $\boldsymbol{u}_* \in \mathbb{U}(\boldsymbol{x}_*)$ existieren muss, damit die Gleichung erfüllt wird.

Damit die Rückführung $\pi = \{\mu, ..., \mu\}$ das System in einer Ruhelage stabilisiert, muss demnach gelten [20, S. 45]:

$$\boldsymbol{x}_* = f_d(\boldsymbol{x}_*, \mu(\boldsymbol{x}_*)). \tag{6.6}$$

6.1 Untersuchung von DDPG an einem einfachen Beispielsystem

Als Beispielsystem für die Untersuchung dient das inverse Pendel.[2] Das inverse Pendel in Abbildung 11 besteht aus einem an einer Achse aufgehängten Pendelarm der Masse $m = 1\,\mathrm{kg}$ und Länge $l = 1\,\mathrm{m}$. Mit dem Drehmoment τ kann das Pendel um die Achse gedreht werden. Die Steuerungsaufgabe besteht darin, durch geeignete Wahl des Eingangs $\boldsymbol{u}$ das Pendel aus der unteren Ruhelage $\boldsymbol{x}_* = (\pi(1 + 2n), 0)^{\mathrm{T}}$ in die oberere Ruhelage $\boldsymbol{x}_* = (2n\pi, 0)^{\mathrm{T}}$ mit $n \in \mathbb{Z}$ zu überführen. Aufgrund der Stellgrößenbeschrenkung reicht das Moment $\tau \in [-3.5\,\mathrm{N\,m}, 3.5\,\mathrm{N\,m}]$ nicht aus, um das unten hängende Pendel ohne Schwingen aufzurichten.

[2] Das inverse Pendel wurde für diese Untersuchung ausgewählt, da sich aufgrund der niedrigen Zustands- und Eingangsdimension die Restkosten und Rückführung als Fläche im $\mathbb{R}^3$ darstellen lassen.

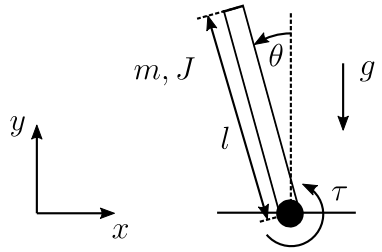

Abbildung 11 – Inverses Pendel. Zustand $\boldsymbol{x} = (\theta, \dot{\theta})^{\mathrm{T}}$, Eingang $\boldsymbol{u} = (\tau)$.

Die Bewegungsgleichung des Pendels lautet

$$J\ddot{\theta} - mgl\sin(\theta) + b\dot{\theta} = \tau \tag{6.7}$$

mit dem Trägheitsmoment $J = ml^2 = 1\,\mathrm{kgm}^2$ und der Dämpfungskonstante $b = 0.02\,\mathrm{N\,m\,s}$.

Diese lässt sich in Zustandsraumdarstellung überführen.

$$\dot{x}_1 = x_2 \tag{6.8a}$$

$$\dot{x}_2 = \frac{g}{l}\sin(x_1) - \frac{b}{J}bx_2 + \frac{1}{J}u_1 \tag{6.8b}$$

Bei Diskretisierung mit dem Euler-Verfahren folgt demnach als Näherung für das Abtastsystem:

$$x_{1,k+1} = x_{1,k} + \Delta t x_{2,k}, \tag{6.9a}$$

$$x_{2,k+1} = x_{2,k} + \Delta t\left(\frac{g}{l}\sin(x_{1,k}) - \frac{b}{J}x_{2,k} + \frac{1}{J}u_{1,k}\right). \tag{6.9b}$$

Erweiterter Zustand Für Systeme mit Zustandskomponenten, die Winkel repräsentieren, wird in der Literatur des bestärkenden Lernens oft angenommen, dass diese auf das Intervall $[-\pi, \pi)$ beschränkt sind. Durch das Ausnutzen der Periodizität des Winkels kann der Zustandsraum bedeutend verkleinert werden. Bildet man die Zustandstrajektorie einer Winkelgröße auf das Intervall $[-\pi, \pi)$ ab, so kommt es bei überschreiten des Intervalls zu Unstetigkeiten. Um dem entgegenzuwirken, kann ein Winkel auch durch seine Cosinus- und Sinuskomponente dargestellt werden. In dieser Repräsentation treten die beschriebenen Unstetigkeiten nicht auf.

Am Beispiel des Pendels würde dies demnach bedeuten, dass der Zustand $\boldsymbol{x} = (x_1, x_2)^{\mathrm{T}}$ durch den erweiterten Zustand $\boldsymbol{o} = (\cos(x_1), \sin(x_1), x_2)^{\mathrm{T}}$ repräsentiert wird. $\boldsymbol{o}$ ist dann anstelle von $\boldsymbol{x}$ der Eingabevektor für den Aktor $\mu_{\boldsymbol{\theta}}$ und den Kritiker $\hat{Q}^{\mu}_{\boldsymbol{\phi}}$.

6.1.1 Vergleich von DDPG und dem iLQR

Die inkrementellen Kosten werden mit

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = x_{1,k}^2 + 0.1x_{2,k}^2 + 0.05u_{1,k}^2 \tag{6.10}$$

angesetzt.

Bei iLQR gilt zusätzlich für die Endkosten:

$$c_N(\boldsymbol{x}_N) = 100x_{1,N}^2 + 100x_{2,N}^2. \tag{6.11}$$

Bemerkung 6.1. *Für das Lernen mit DDPG werden Winkel in ihre Cosinus- und Sinuskomponente aufgeteilt und in der Kostenfunktion auf das Intervall $[-\pi, \pi)$ abgebildet, da dies das Lernen enorm beschleunigt. Bei iLQR führen die daraus resultierenden Unstetigkeiten in den Ableitungen der Kostenfunktion hingegen zu schlechten Konvergenzeigenschaften.*

Für das Beispielsystem und die Kostenfunktion (6.10) hat iLQR zunächst keine Lösung gefunden. Es werden deshalb zusätzlich Endkosten c_N angesetzt, damit der Algorithmus konvergiert. Beide Algorithmen sind aufgrund der Periodizität der Kostenfunktion bei DDPG hinsichtlich der resultierenden Gesamtkosten nur schwer vergleichbar.

In der Literatur des bestärkenden Lernens werden Kosten für Winkel oft in deren Sinus- oder Cosinuskomponenten angesetzt. Für das inverse Pendel könnte dann bspw.

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = (1 - \cos(x_{1,k})) + 0.1x_{2,k}^2 + 0.05u_{1,k}^2. \tag{6.12}$$

verwendet werden. Auch bei iLQR ist es möglich, die Kostenfunktion derart anzusetzen.

Verteilung des Anfangszustands

Wie in Abschnitt 3.1 beschrieben, wird das System bei DDPG zu Beginn einer Episode in einen Anfangszustand $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$ versetzt. Für realitätsnahe Untersuchungen sollte $\boldsymbol{x}_0$ in einer kleinen Umgebung um eine stabile Ruhelage liegen, im Fall des inversen Pendels demnach nahe $\boldsymbol{x}_0 = (\pi, 0)^{\mathrm{T}}$. In der Simulation kann aber prinzipiell jede beliebige Verteilung $p(\boldsymbol{x}_0)$ angesetzt werden. So kann das System in Zustände versetzt werden, die sonst erst später im Laufe des Lernvorgangs besucht werden würden. Diese künstliche Erkundung beschleunigt den Lernvorgang, lässt sich aber an realen System nicht durchführen. Bei iLQR wird hingegen ein fester Wert für den Anfangszustand angesetzt, in dieser Untersuchung $\boldsymbol{x}_0 = (\pi, 0)^{\mathrm{T}}$.

Regelverhalten

In Abbildung 12 sind die Trajektorien des inversen Pendels bei Anwenden der mit DDPG erlernten Rückführung und iLQR dargestellt.

Die Rechenzeit von iLQR liegt auf einem gewöhnlichen[3] Desktop-PC im unteren einstelligen Minutenbereich. Das Training von DDPG für die Lernvorgänge in Abbildung 16 dauerte hingegen jeweils ca. 3 h, wobei der Algorithmus aber bereits nach ca. 15 bis 20 min Rechenzeit konvergiert.

Prinzipiell ist festzuhalten, dass es mit beiden Methoden möglich ist, das inverse Pendel aus der unteren Ruhelage $\boldsymbol{x}_{*,u} = ((1+2n)\pi, 0)^{\mathrm{T}}$ in die obere Ruhelage $\boldsymbol{x}_{*,o} = (n\pi, 0)^{\mathrm{T}}$ mit $n \in \mathbb{Z}$ zu überführen und dort zu stabilisieren. Besonders um die oberere Ruhelage weist DDPG jedoch einen relativ stark *zitternden* Eingang auf, der an eine Sliding-Mode-Regelung erinnert. Dies hängt mit dem steilen Anstieg der Rückführung in der Umgebung der Ruhelage zusammen, wie in Abbildung 17 unten rechts zu sehen. Dem könnte eventuell entgegengewirkt werden, indem man die Abtastzeit verringert.

Die Gesamtkosten der Trajektorien in Abbildung 12 sind bei iLQR 30.76 (21.70 mit Abbildung des Winkels x_1 auf $[-\pi, \pi)$) und bei DDPG 23.75.

Vergleicht man das Verhalten beider Rückführungen für verschiedene Anfangswerte, so zeigt sich, dass es mit der mit iLQR für $\boldsymbol{x}_0 = (\pi, 0)$ bestimmten Rückführung nicht mehr möglich ist, das System in die obere Ruhelage zu überführen und dort zu stabilisieren. Die mit iLQR bestimmte zeitvariante Rückführung ist nur lokal um die Nominaltrajektorie gültig. Mit iLQR muss demnach für einen neuen Anfangswert eine erneute Trajektorienplanung durchgeführt werden. Mit DDPG hingegen wurde eine *globale* Rückführung erlernt, die das System aus beliebigen Anfangswerten in die obere Ruhelage überführt und dort stabilisiert.

Bemerkung 6.2. *Die Verwendung des Begriffes* ***global*** *bezieht sich auf die Teilmenge des Zustandsraumes $\mathbb{X}$, die während des Lernvorgangs besucht wurde. Eine auf Simulationsexperimenten basierende Abschätzung für den Bereich, in dem $\mu_{\boldsymbol{\theta}}$ das System in die Ruhelage überführt wird, ist die Menge $\{\boldsymbol{x} \in \mathbb{X} | x_1 \in [-\pi, \pi), |x_2| < 8\}$.*

[3] Prozessor: Intel i5-6500, RAM: 8GB, Grafikkarte: Intel HD Graphics 530

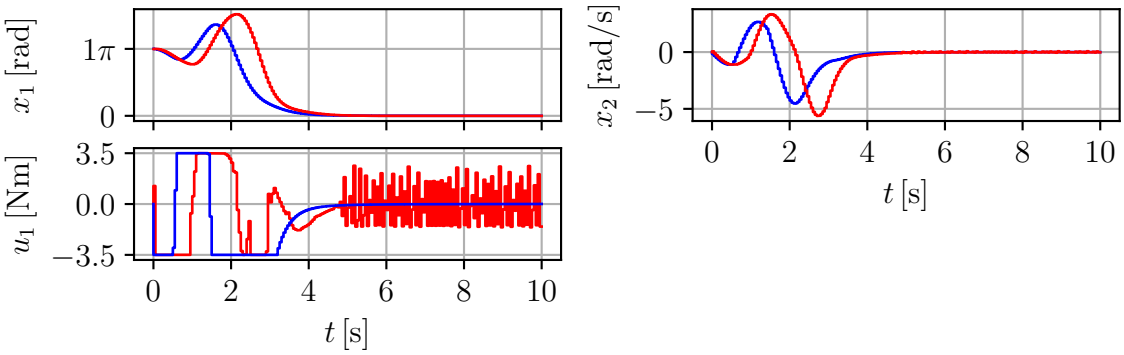

Abbildung 12 – Trajektorien des inversen Pendels bei iLQR (blau) und DDPG (rot) für den Anfangszustand $\boldsymbol{x}_0 = (\pi, 0)^{\mathrm{T}}$.

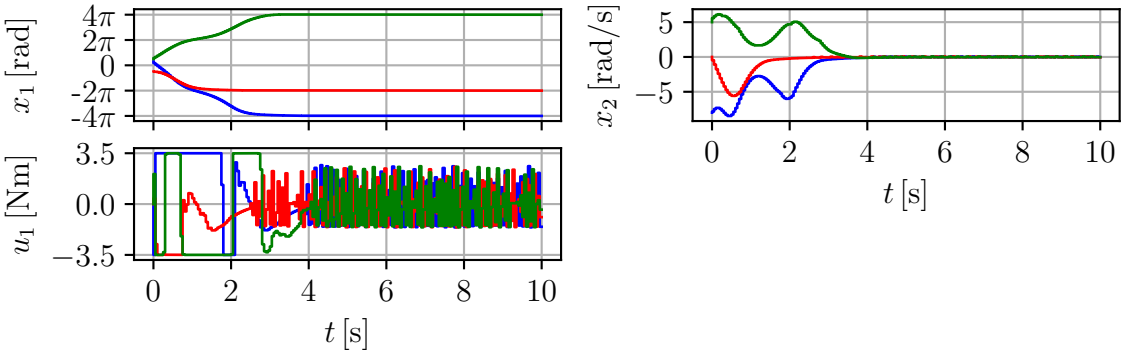

Abbildung 13 – Trajektorien des inversen Pendels bei Anwendung der mit DDPG erlernten Rückführung, für die Anfangszustände $\boldsymbol{X}_0 = \{(0.3\pi, -8)^{\mathrm{T}}, (-0.5\pi, 0)^{\mathrm{T}}, (0.5\pi, 5)^{\mathrm{T}}\}$.

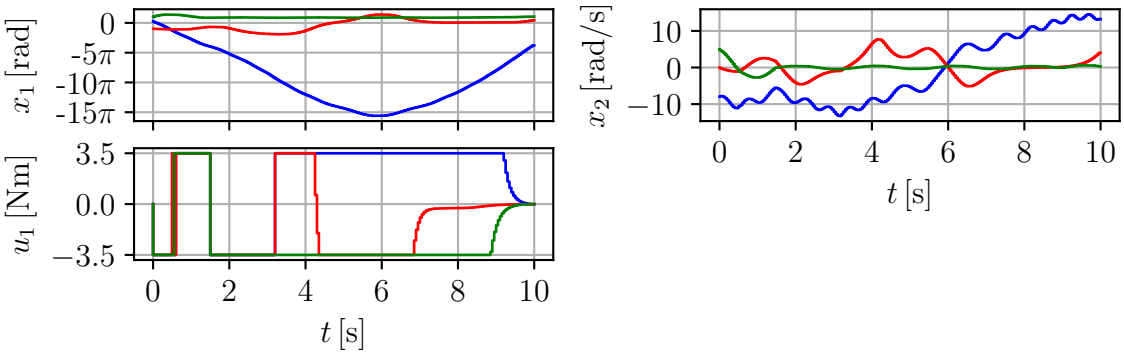

Abbildung 14 – Trajektorien des inversen Pendels bei Anwendung der mit iLQR für $\boldsymbol{x}_0 = (\pi, 0)^{\mathrm{T}}$ bestimmten Rückführung $\{\boldsymbol{K}_{[0:N-1]}, \boldsymbol{k}_{[0:N-1]}\}$ für die Anfangszustände $\boldsymbol{X}_0 = \{(0.3\pi, -8)^{\mathrm{T}}, (-0.5\pi, 0)^{\mathrm{T}}, (0.5\pi, 5)^{\mathrm{T}}\}$.

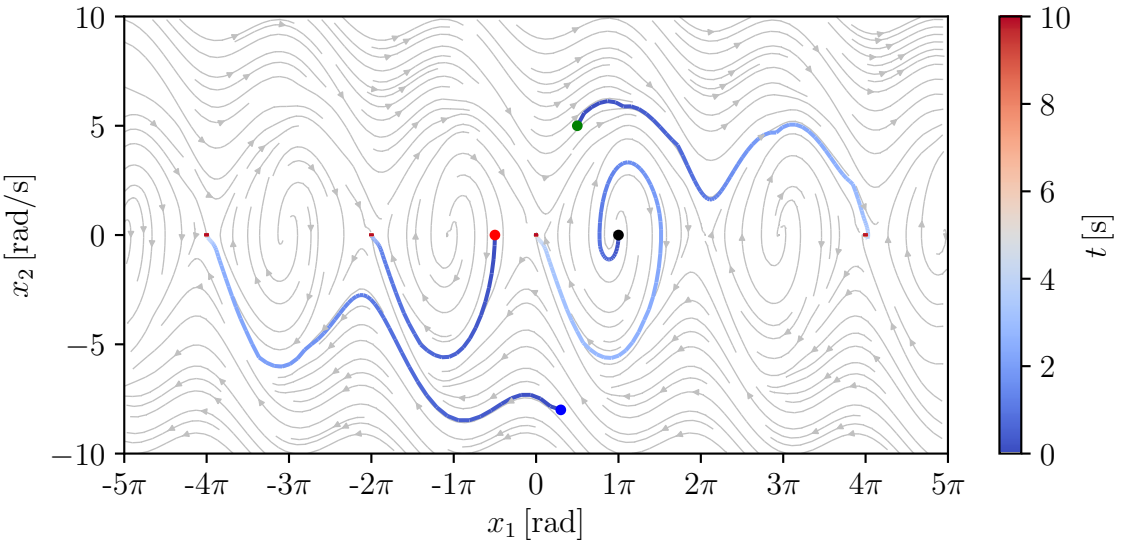

Abbildung 15 – Trajektorien des inversen Pendels bei Anwendung der mit DDPG erlernten Rückführung aus Abbildung 12 und Abbildung 13 in der Phasenebene für die Anfangszustände $\boldsymbol{X}_0 = \{(\pi, 0)^\mathrm{T}, (0.3\pi, -8)^\mathrm{T}, (-0.5\pi, 0)^\mathrm{T}, (0.5\pi, 5)^\mathrm{T}\}$. Im Hintergrund ist das Phasenportrait des autonomen Systems $f_c(\boldsymbol{x}, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}))$ dargestellt.

6.1.2 Approximation von V^μ mit DDPG

Bei DDPG werden Kritiker $\hat{Q}^\mu_{\boldsymbol{\phi}}$ und Aktor $\mu_{\boldsymbol{\theta}}$ verwendet, um eine approximative Regler-Iteration durchzuführen. Aus diesen lässt sich eine Approximation für die Restkosten $\hat{V}^\mu_{\boldsymbol{\phi}}(\boldsymbol{x}) = \hat{Q}^\mu_{\boldsymbol{\phi}}(\boldsymbol{x}, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}))$ bilden.

Für die Untersuchung in diesem Abschnitt wurden fünf Lernvorgänge am inversen Pendel mit der Kostenfunktion (6.10) durchgeführt. Die diskontierten Gesamtkosten sind in der Lernkurve in Abbildung 16 dargestellt.

In Abbildung 17 ist die aktuelle Schätzung $\hat{V}^\mu$, die korrespondierende Rückführung $\mu_{\boldsymbol{\theta}}$ und das resultierende Phasenportrait zu verschiedenen Zeitpunkten des Lernvorgangs dargestellt. Nach 0 Trainingsschritten sind sowohl die Restkosten $\hat{V}^\mu_{\boldsymbol{\phi}}$ als auch die Rückführung $\mu_{\boldsymbol{\theta}} \approx 0$, da die Netzwerkparameter mit sehr kleinen Werten initialisiert werden.

Nach 15000 Trainingsschritten (Reihe 2) hat der Agent gelernt, bei positiver Geschwindigkeit ein positives Drehmoment auszugeben und bei negativer Geschwindigkeit ein negatives. Im Laufe des Trainings lernt der Agent das Pendel wieder abzubremsen, wenn die Winkelgeschwindigkeit x_2 zu groß wird, um ein Überschwingen zu verhindern.

Die Schätzung der Restkosten $\hat{V}^\mu_{\boldsymbol{\phi}}$ übersteigt zuächst die tatsächlichen Gesamtkosten, da der Agent zu Beginn des Trainings hohe Gesamtkosten erfährt. Die Rückführung nähert sich im Laufe des Trainings der optimalen Rückführung π^* an, die Gesamtkosten sinken und somit sinkt auch die Schätzung für die Restkosten $\hat{V}^\mu_{\boldsymbol{\phi}}$. Diese konvergiert gegen die tatsächlichen Gesamtkosten. Die erlernte Schätzung der $\hat{V}^\mu_{\boldsymbol{\phi}}$ hat demnach eine gute Qualität.

In Abbildung 18 ist das Phasenportrait vor und nach dem Lernvorgang (erste und letzte Zeile in Abbildung 17) um die instabile Ruhelage vergrößert dargestellt, um zu verdeutlichen, wie sich das Systemverhalten durch Anwendung der Rückführung verändert.

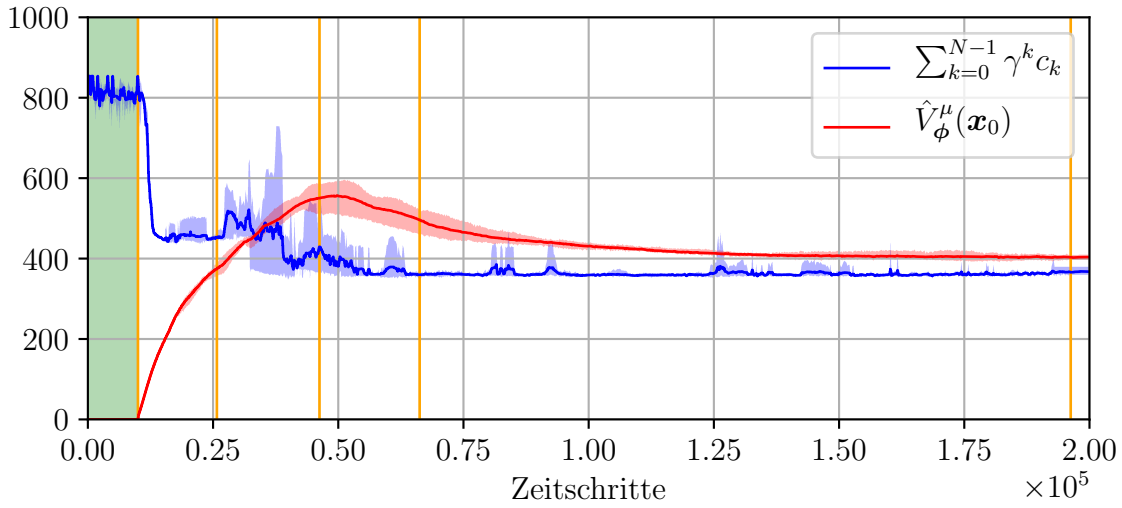

Abbildung 16 – Lernkurve für das inverse Pendel mit der Kostenfunktion (6.10). Die Abbildung zeigt die Mittelwerte der Gesamtkosten (blau) und der Schätzung der Restkosten (rot) über fünf Lernvorgängen (Details in Tabelle 3). Die maximalen und minimalen Werte sind als hellblauer bzw. hellroter Bereich dargestellt. Die Aufwärmphase, in der der Agent zufällig agiert, dauert 10000 Zeitschritte an. Nach bereits 60000 Zeitschritten konvergiert der Algorithmus. Dies entspricht ca. 40.0 min Interaktionszeit mit dem System. Die orangen Markierungen kennzeichnen die Zeitpunkte der Plots in Abbildung 17.

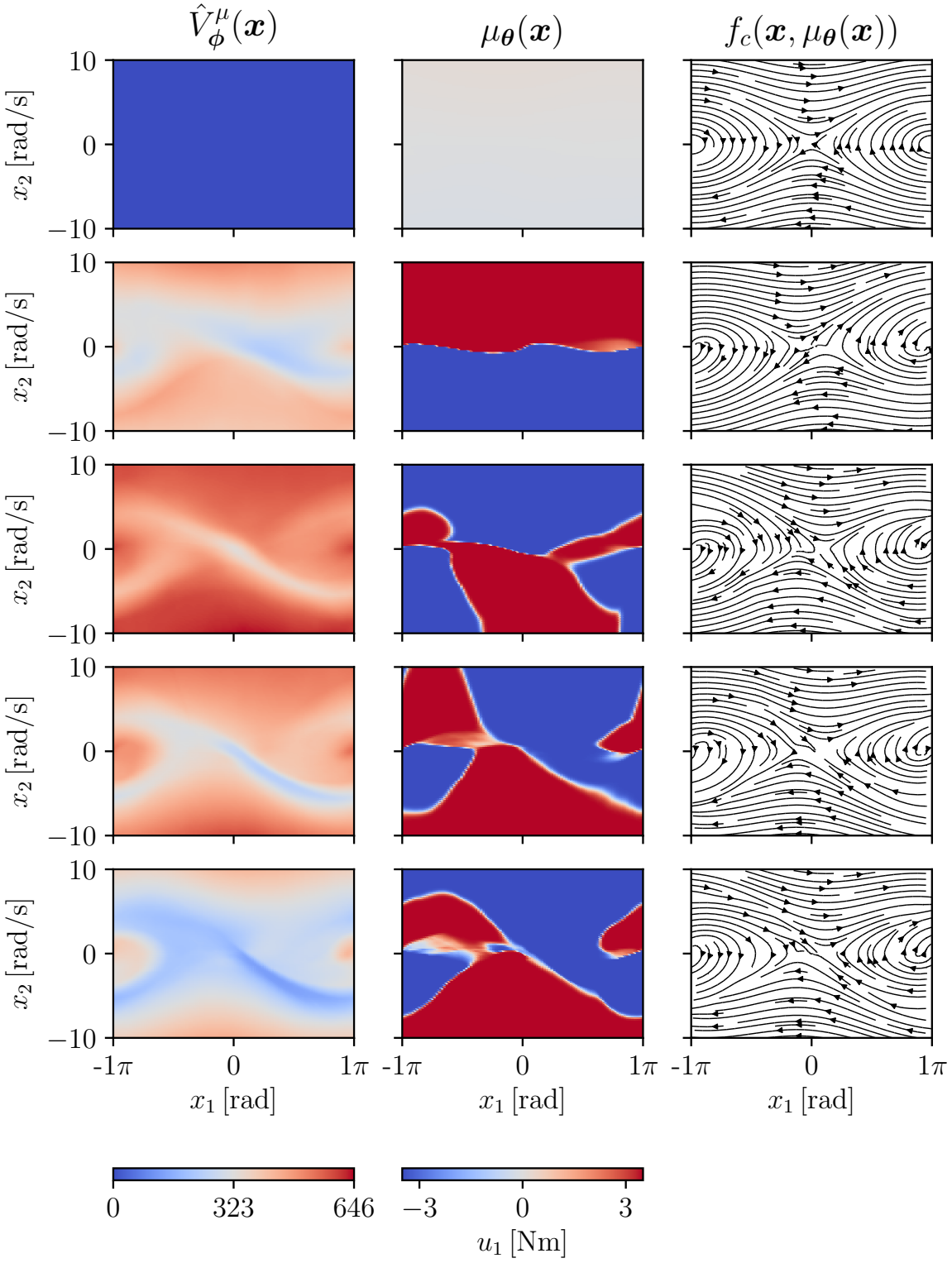

Abbildung 17 – Darstellung des Lernvorgangs von DDPG für das inverse Pendel. Von oben nach unten: nach (0, 15761, 36250, 56250, 186250) Zeitschritten Training (s. Abbildung 16, orange Markierungen). In der rechten Spalte ist das resultierende Vektorfeld des autonomen Systems dargestellt.

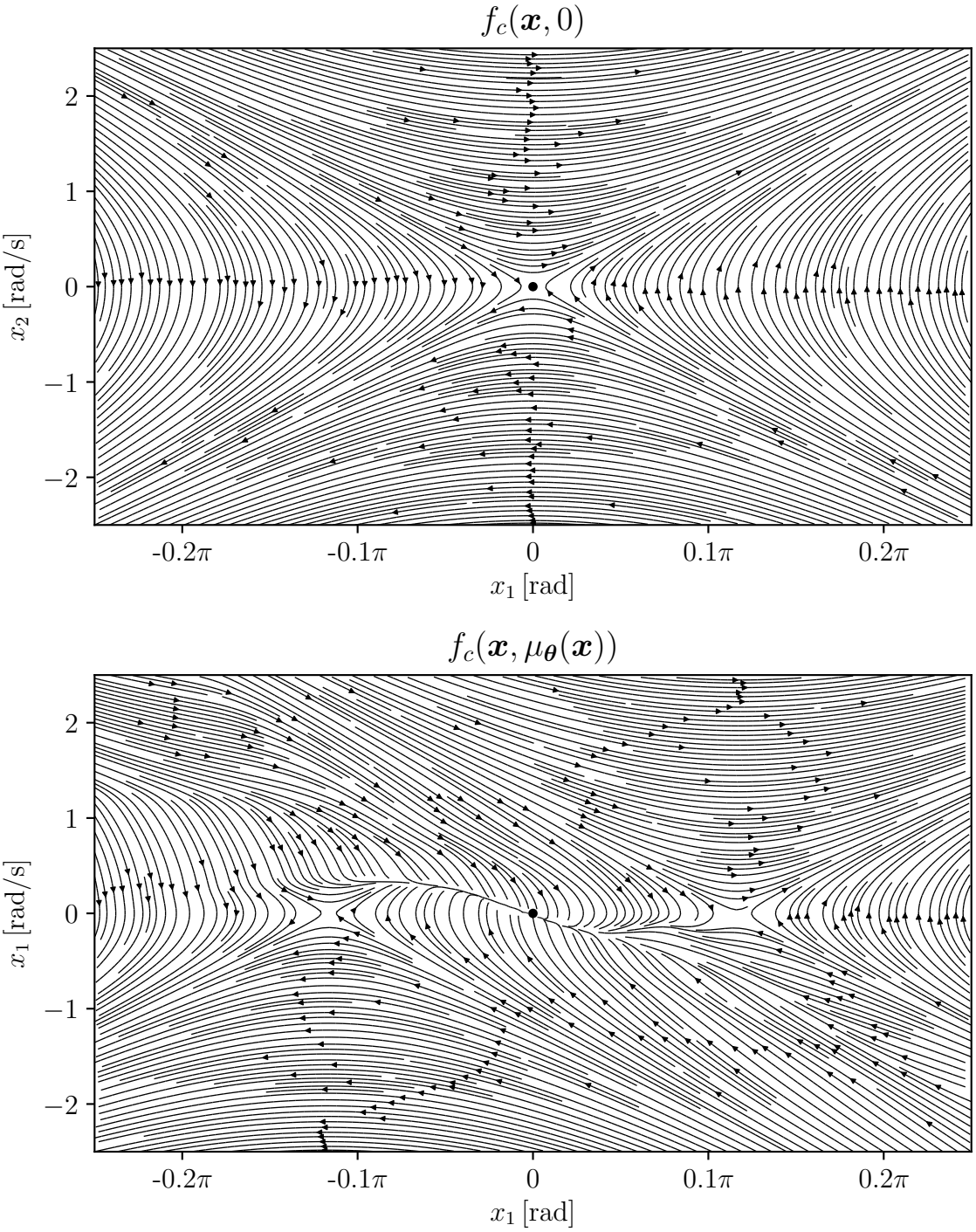

Abbildung 18 – Darstellung des Phasenportraits in der Umgebung der Ruhelage $\boldsymbol{x} = (0, 0)$. Vor dem Training (oben) ist die instabile Ruhelage ein Sattelpunkt, welcher im Laufe des Lernvorgangs in einen stabilen Knoten übergeht.

6.2 Akrobot

Der in Abbildung 19 dargestellte sog. Akrobot ist ein unteraktuiertes mechanisches System, bestehend aus zwei seriellen Pendelarmen, welche freischwingend aufgehängt sind. Das Gelenk, welches beide Arme verbindet, ist durch das Drehmoment τ_1 aktuiert. Die untere stabile Ruhelage liegt bei $\boldsymbol{x}_{*,u} = ((1+2n)\pi, (1+2n)\pi, 0, 0)^{\mathrm{T}}$, die obere instabile bei $\boldsymbol{x}_{*,o} = (2n\pi, 2n\pi, 0, 0)^{\mathrm{T}}$ mit $n \in \mathbb{Z}$.

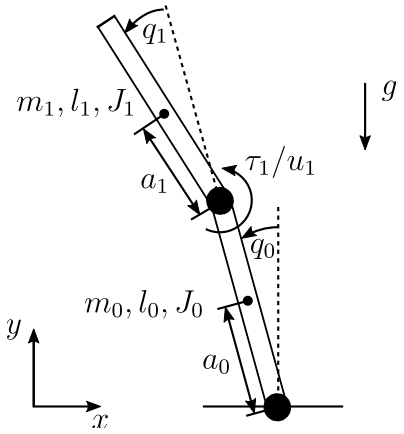

Abbildung 19 – Akrobot. Zustand $\boldsymbol{x} = (q_0, q_1, \dot{q}_0, \dot{q}_1)^{\mathrm{T}}$, Eingang $\boldsymbol{u} = (\ddot{q}_1)$. Verwendete Parameter und Größen zur Herleitung der Bewegungsgleichungen finden sich in Abschnitt A.2.

Die Kosten wurden mit

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \begin{cases} 200, & \text{wenn } \boldsymbol{x}_k \in \mathbb{X}^-, \\ x_{1,k}^2 + x_{2,k}^2 + 0.01x_{3,k}^2 + 0.01x_{4,k}^2 + 0.05u_{1,k}^2, & \text{sonst}, \end{cases} \tag{6.13}$$

und bei iLQR zusätzlich die Endkosten

$$c_N(\boldsymbol{x}_N) = 100x_{1,N}^2 + 100x_{2,N}^2 + 10x_{3,N}^2 + 10x_{4,N}^2 \tag{6.14}$$

angesetzt, wobei für die Menge der unzulässigen Zustände

$$\mathbb{X}^- := \{\boldsymbol{x}_k \in \mathbb{X}_k | |x_{3,k}| > 50 \vee |x_{4,k}| > 50\} \tag{6.15}$$

gilt.

Wie in Abbildung 21 zu sehen, ist es mit beiden Methoden möglich, das System aus der unteren in die obere Ruhelage zu überführen und dort zu stabilisieren. Mit DDPG gelingt das Aufschwingen etwas schneller, als mit iLQR, was mit der periodischen Kostenfunktion zusammenhängt, wodurch der Winkel x_2 nicht auf 0 gezwungen wird.

In Abbildung 22 wurde die mit DDPG erlernte Rückführung $\mu_{\boldsymbol{\theta}}$ für verschiedene Anfangsbedingungen auf den Akrobot angewandt. Für alle gewählten Anfangswerte wird das System erfolgreich in die obere Ruhelage überführt und stabilisiert.

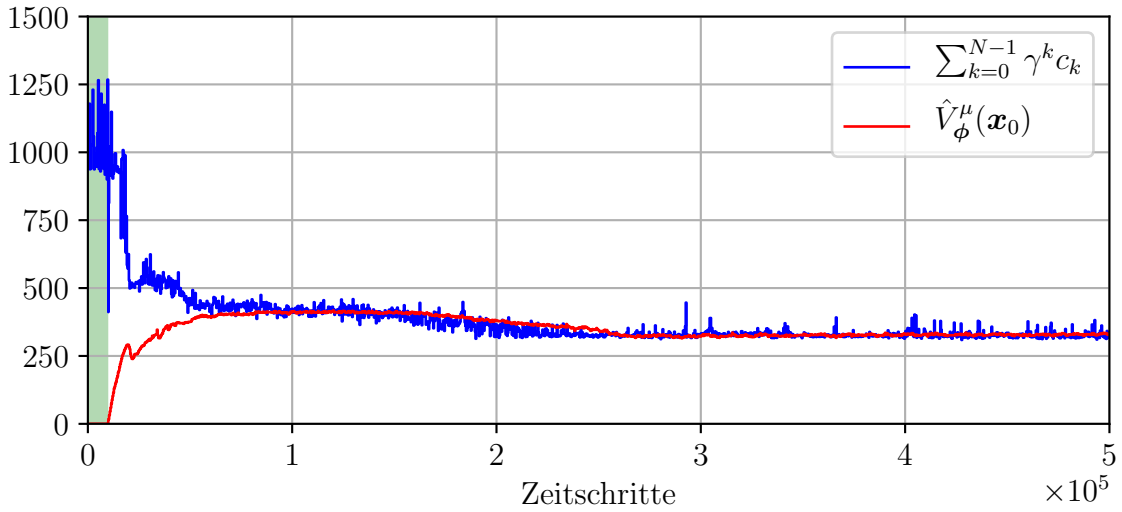

Abbildung 20 – Lernkurve für den Akrobot mit der Kostenfunktion (6.13). Die Aufwärmphase, in der der Agent zufällig agiert dauert 10000 Zeitschritte an. Der Algorithmus konvergiert nach ca. 300000 Schritten. Dies entspricht ca. 2.5 h Interaktionszeit mit dem System. Zusätzlich zu den diskontierten Gesamtkosten, die der Agent erfährt ist die aktuelle Schätzung von $\hat{V}_{\phi}^{\mu}$ für den Anfangszustand aufgetragen um den Lernvorgang zu verdeutlichen. Es ist deutlich erkennbar, dass die Schätzung von $\hat{V}_{\phi}^{\mu}$ nach kurzer Trainingsdauer sehr gut mit den tatsächlich akkumulierten Kosten übereinstimmt. Details zum Lernvorgang in Tabelle 6.

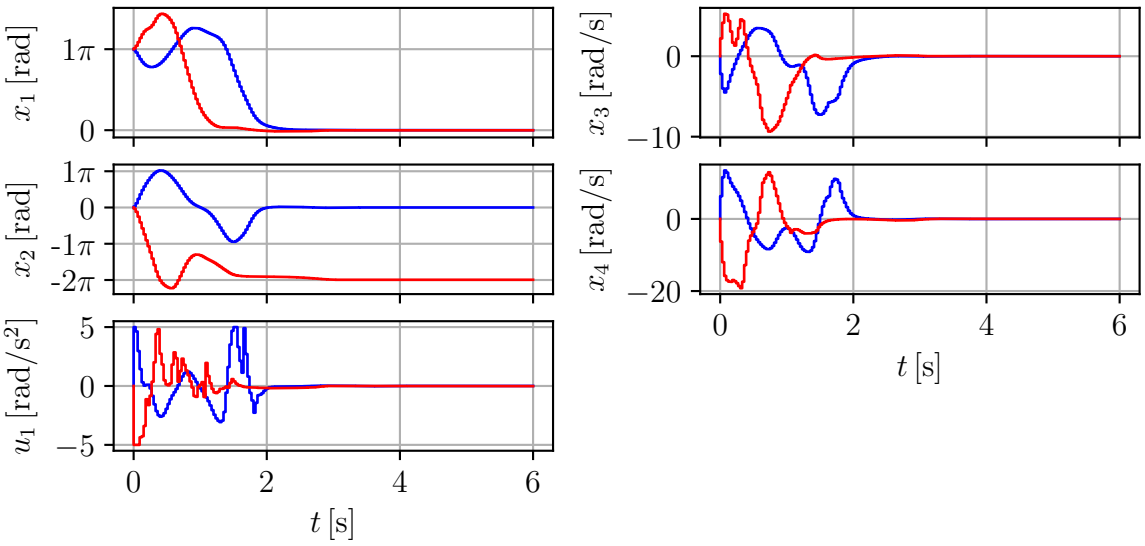

Abbildung 21 – Trajektorien des Akrobots bei iLQR (blau) und DDPG (rot) für den Anfangszustand $\boldsymbol{x}_0 = (\pi, 0, 0, 0)^{\mathrm{T}}$.

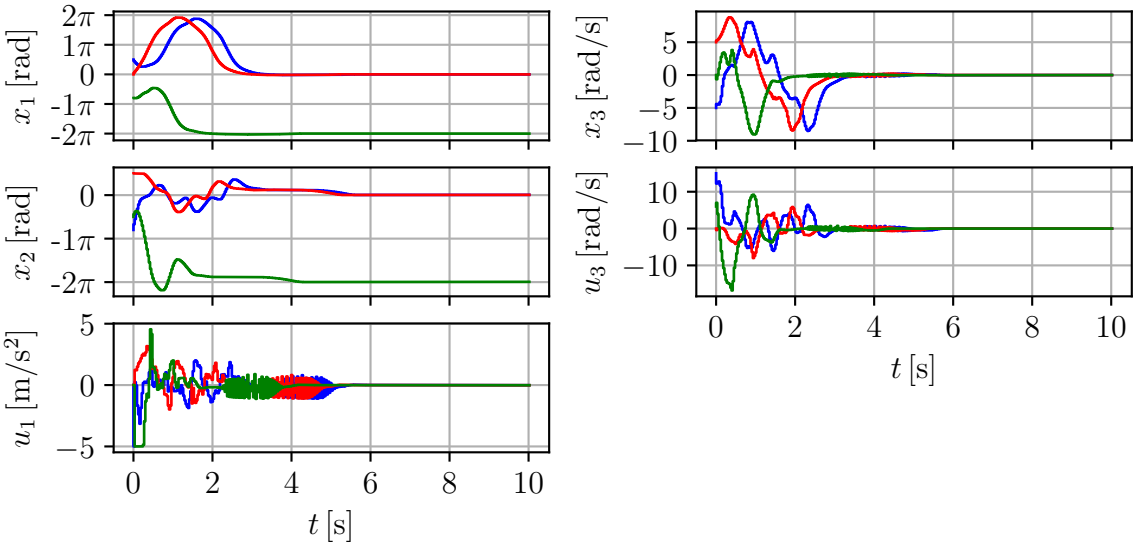

Abbildung 22 – Trajektorien des Akrobots bei Anwendung der mit DDPG erlernten Rückführung, für die Anfangszustände $\boldsymbol{X}_0 = \{(0.5\pi, -0.8\pi, -5, 15)^{\mathrm{T}}, (0, 0.5\pi, 5, 0)^{\mathrm{T}}, (-0.8\pi, -0.5\pi, 0, 6)^{\mathrm{T}}\}$.

6.3 Wagen-Pendel

Das in Abbildung 23 dargestellte Wagen-Pendel besteht aus einem Wagen sowie einem an diesem aufgehängten, freischwingenden Pendelarm. Der Stelleingriff ist die in x-Richtung wirkende Kraft F_0, die durch einen Riemen auf den Wagen übertragen wird.

Die unteren stabilen Ruhelagen liegen bei ${}^{x_1}\boldsymbol{x}_{*,u} = (x_1 \in \mathbb{R}, (1+2n)\pi, 0, 0)^{\mathrm{T}}$, die oberen instabilen bei ${}^{x_1}\boldsymbol{x}_{*,o} = (x_1 \in \mathbb{R}, 2n\pi, 0, 0)^{\mathrm{T}}$ mit $n \in \mathbb{Z}$ und sind demnach unabhängig von der Position des Wagens.

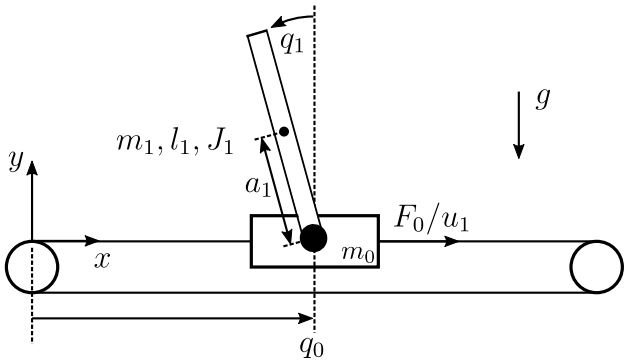

Abbildung 23 – Wagen-Pendel. Zustand $\boldsymbol{x} = (q_0, q_1, \dot{q}_0, \dot{q}_1)^{\mathrm{T}}$, Eingang $\boldsymbol{u} = (\ddot{q}_0)$. Verwendete Parameter und Größen zur Herleitung der Bewegungsgleichungen befinden sich in Abschnitt A.3.

Die Kosten wurden mit

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \begin{cases} 200, & \text{wenn } \boldsymbol{x}_k \in \mathbb{X}^-, \\ 0.5x_{1,k}^2 + x_{2,k}^2 + 0.02x_{3,k}^2 + 0.05x_{4,k}^2 + 0.05u_{1,k}^2 & \text{sonst.} \end{cases} \tag{6.16}$$

und bei iLQR zusätzlich die Endkosten

$$c_N(\boldsymbol{x}_N) = 100x_{1,N}^2 + 100x_{2,N}^2 + 10x_{3,N}^2 + 10x_{4,N}^2 \tag{6.17}$$

angesetzt, wobei für die Menge der unzulässigen Zustände

$$\mathbb{X}^- := \{\boldsymbol{x}_k \in \mathbb{X}_k | |x_{1,k}| > 1.5\} \tag{6.18}$$

gilt.

Wie in Abbildung 25 zu sehen, ist es mit beiden Methoden möglich das System aus der unteren in die obere Ruhelage zu überführen und dort zu stabilisieren. Mit DDPG gelingt das Aufschwingen etwas langsamer, als mit iLQR. Zudem wird die untere Ruhelage beim Aufschwingen zweimal durchlaufen und das System wird bei $\boldsymbol{x}_2 = 2\pi$ und nicht bei $\boldsymbol{x}_2 = 0$ stabilisiert.

In Abbildung 26 wurde die mit DDPG erlernte Rückführung $\mu_{\boldsymbol{\theta}}$ für verschiedene Anfangswerte auf das Wagen-Pendel angewandt. Für alle gewählten Anfangswerte

wird das System erfolgreich in die obere Ruhelage überführt und stabilisiert. Bei der Überführung von der Ruhelage ${}^{1}\boldsymbol{x}_{*,\mathrm{o}}$ in die Ruhelage ${}^{0}\boldsymbol{x}_{*,\mathrm{o}}$ wird das Pendel nicht aufrecht stehend überführt. Auch wenn die Überführung gelingt, zeigt der Agent ein suboptimales Verhalten. Das Pendel fällt zunächst nach unten und wird dann wieder aufgefangen. Für die Anfangswerte ${}^{0.7}\boldsymbol{x}_{*,\mathrm{o}}$ und ${}^{-1.1}\boldsymbol{x}_{*,\mathrm{o}}$ wird das Pendel hingegen aufrecht stehend in die obere Ruhelage überführt, was wesentlich kostengünstiger ist (hier nicht dargestellt).

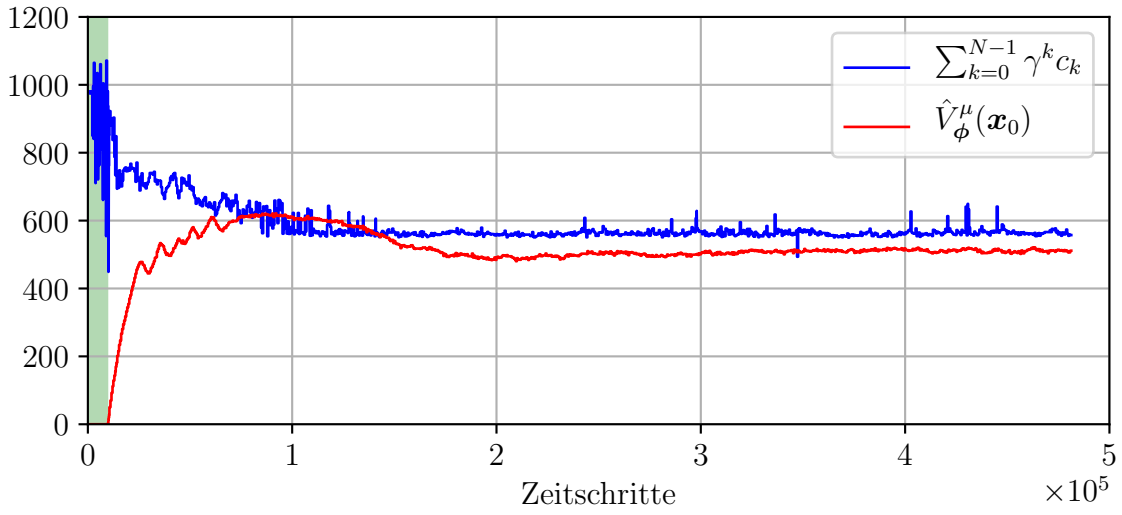

Abbildung 24 – Lernkurve für das Wagen-Pendel mit der Kostenfunktion (6.16). Die Aufwärmphase, in der der Agent zufällig agiert, dauert 10000 Zeitschritte an. Der Algorithmus konvergiert nach ca. 200000 Zeitschritten, was ca. 1.1 h Interaktionszeit mit dem System entspricht. Zusätzlich zu den diskontierten Gesamtkosten, die der Agent erfährt, ist die aktuelle Schätzung von $\hat{V}_{\boldsymbol{\phi}}^{\mu}$ für den Anfangszustand aufgetragen um den Lernvorgang zu verdeutlichen. Details zum Lernvorgang befinden sich in Tabelle 3.

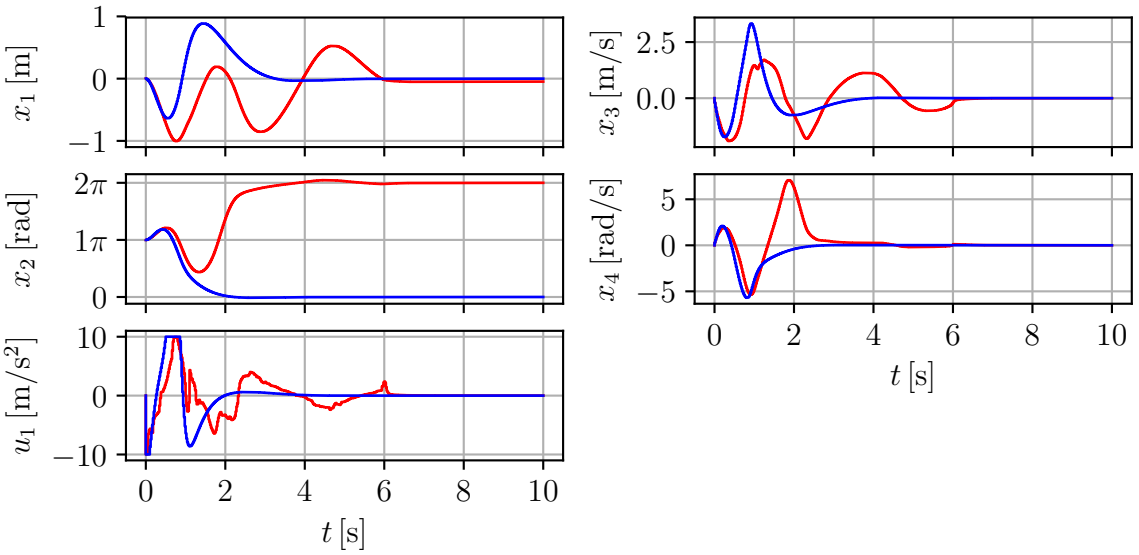

Abbildung 25 – Trajektorien des Wagen-Pendels bei iLQR (blau) und DDPG (rot) für den Anfangszustand $\boldsymbol{x}_0 = (0, \pi, 0, 0)^{\mathrm{T}}$.

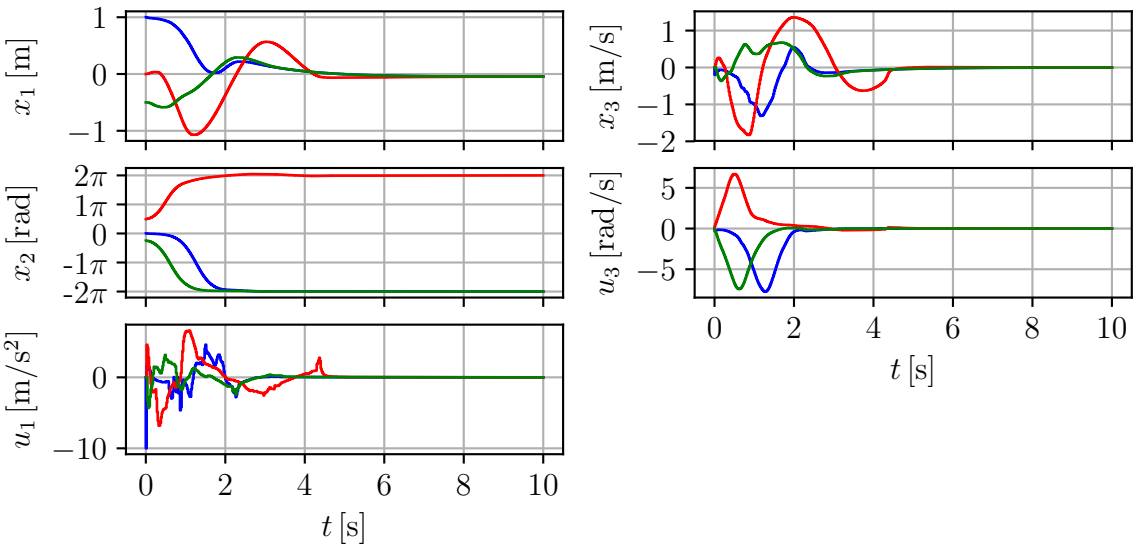

Abbildung 26 – Trajektorien des Wagen-Pendels bei der mit DDPG erlernten Rückführung, für die Anfangszustände $\boldsymbol{X}_0 = \{(1, 0, 0, 0)^{\mathrm{T}}, (0, 0.5\pi, 0, 0)^{\mathrm{T}}, (-0.5, -0.25\pi, 0, 0)^{\mathrm{T}}\}$.

6.4 Doppel- und Dreifach-Wagen-Pendel

Das Doppel- und Dreifach-Wagen-Pendel stellen eine Erweiterung des Wagen-Pendels dar, bei der eine serielle Pendelkette mit zwei, respektive drei Pendelarmen an einem fahrbaren Wagen aufgehangen ist.

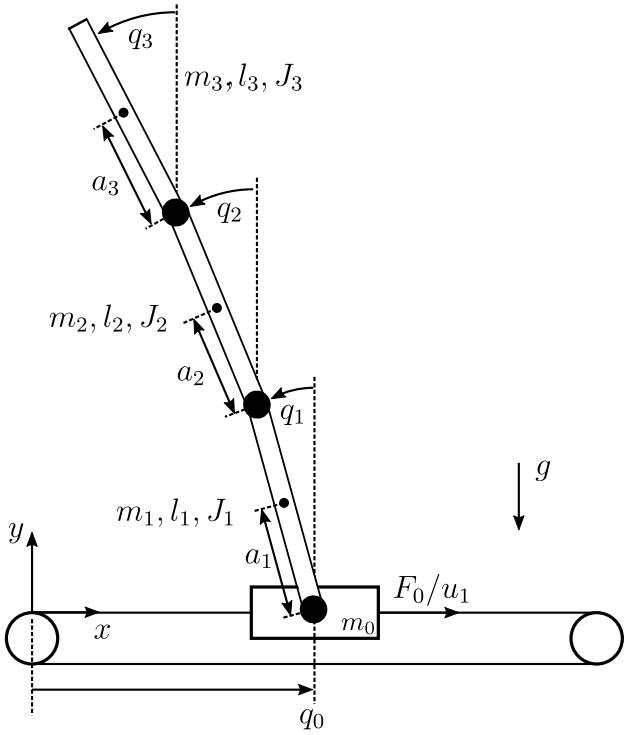

Abbildung 27 – Dreifach-Wagen-Pendel. Zustand $\boldsymbol{x} = (q_0, q_1, q_2, q_3, \dot{q}_0, \dot{q}_1, \dot{q}_2, \dot{q}_3)^{\mathrm{T}}$, Eingang $\boldsymbol{u} = (\ddot{q}_0)$. Verwendete Parameter und Größen zur Herleitung der Bewegungsgleichungen befinden sich in Abschnitt A.4.

Mit DDPG war es – im Rahmen der vorliegenden Diplomarbeit – nicht möglich für das Doppelpendel eine Rückführung $\mu_{\boldsymbol{\theta}}$ zu erlernen, die das System aus der unteren Ruhelage $\boldsymbol{x}_{*,\mathrm{u}} = (0, \pi, \pi, 0, 0, 0)^{\mathrm{T}}$ in die obere Ruhelage $\boldsymbol{x}_{*,\mathrm{o}} = (0, 0, 0, 0, 0, 0)^{\mathrm{T}}$ überführt. Auch ein Stabilisieren in der oberen Ruhelage war nur bedingt möglich. Für das Dreifach-Wagen-Pendel wurden deshalb mit DDPG keine weiteren Simulationsstudien durchgeführt. Mit iLQR können hingegen für beide Systeme alle Steuerungsaufgaben gelöst werden, wie in Abbildung 28 und Abbildung 29 zu sehen ist. Die Rechenzeit für eine Überführungsaufgabe auf einem gewöhnlichen[4] Desktop-PC beträgt weniger als 30 min.

[4]Prozessor: Intel i5-6500, RAM: 8GB, Grafikkarte: Intel HD Graphics 530

Beim Doppelpendel wurden die Kosten mit

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = 15x_{1,k}^2 + 10x_{2,k}^2 + 10x_{3,k}^2 + 0.1u_{1,k}^2, \tag{6.19a}$$
$$c_N(\boldsymbol{x}_N) = 100x_{1,N}^2 + 100x_{2,N}^2 + 100x_{3,N}^2 + 10x_{4,N}^2 + 10x_{5,N}^2 + 10x_{6,N}^2 \tag{6.19b}$$

und beim Dreifach-Wagen-Pendel mit

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = 15x_{1,k}^2 + 10x_{2,k}^2 + 10x_{3,k}^2 + 10x_{4,k}^2 + 0.1u_{1,k}^2, \tag{6.20a}$$
$$c_N(\boldsymbol{x}_N) = 100x_{1,N}^2 + 100x_{2,N}^2 + 100x_{3,N}^2 + 100x_{4,N}^2 + 10x_{5,N}^2 + 10x_{6,N}^2 + 10x_{7,N}^2 + 10x_{8,N}^2 \tag{6.20b}$$

angesetzt.

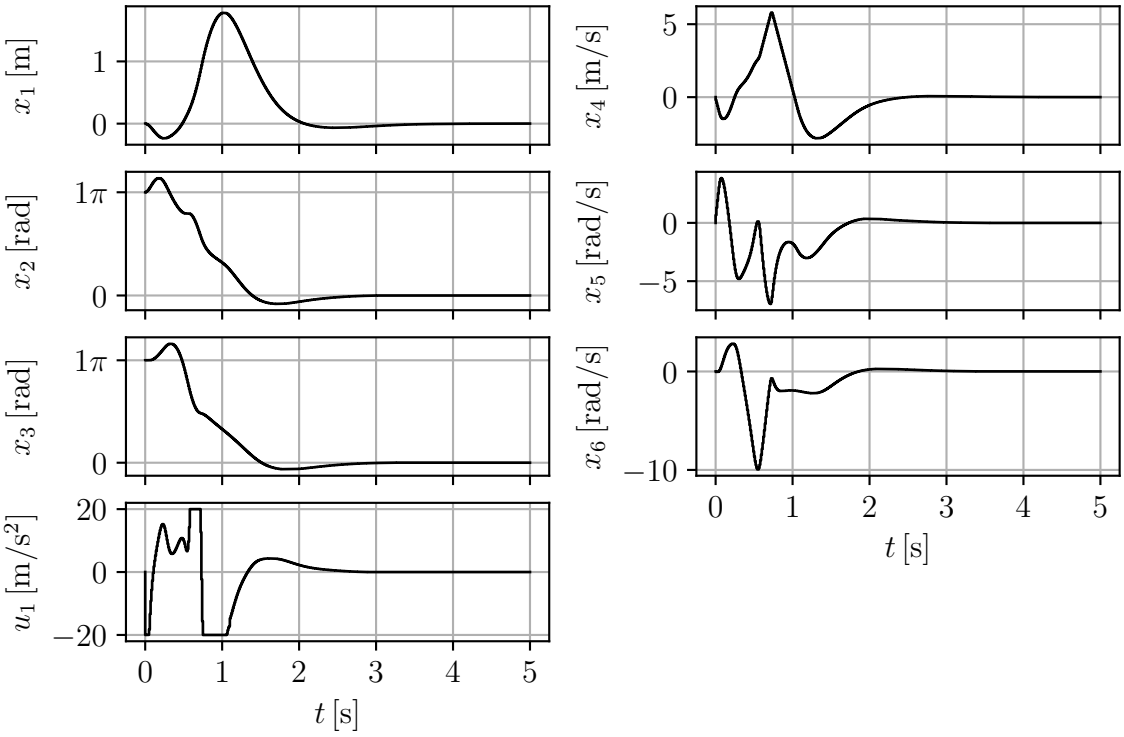

Abbildung 28 – Trajektorie des Doppel-Wagen-Pendels bei Anwendung der mit iLQR bestimmten Rückführung. Dargestellt ist die Überführung aus der unteren Ruhelage $\boldsymbol{x}_{*,\mathrm{u}} = (0, \pi, \pi, 0, 0, 0)^{\mathrm{T}}$ in die obere Ruhelage $\boldsymbol{x}_{*,\mathrm{o}} = (0, 0, 0, 0, 0, 0)^{\mathrm{T}}$.

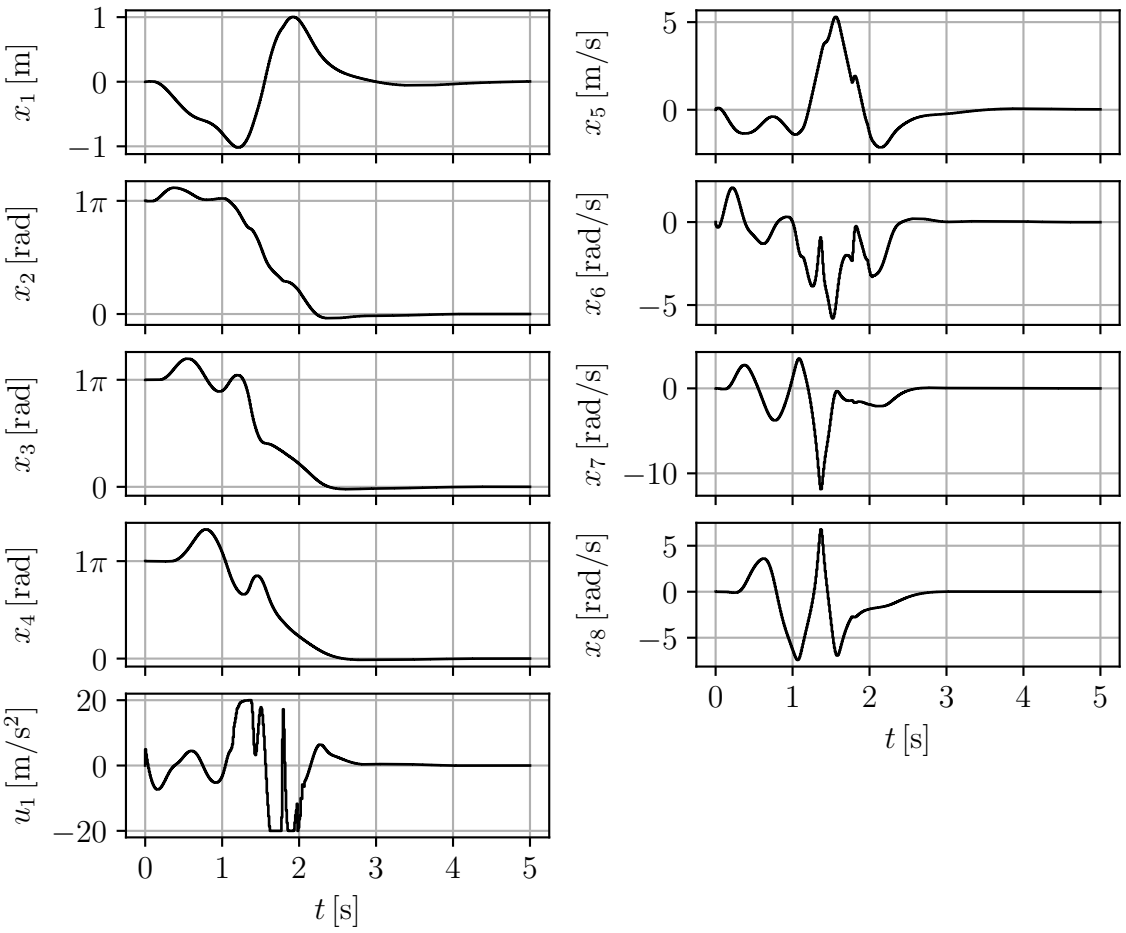

Abbildung 29 – Trajektorie des Dreifach-Wagen-Pendels bei Anwendung der mit iLQR bestimmten Rückführung. Dargestellt ist die Überführung aus der unteren Ruhelage $\boldsymbol{x}_{*,\mathrm{u}} = (0, \pi, \pi, \pi, 0, 0, 0, 0)^{\mathrm{T}}$ in die obere Ruhelage $\boldsymbol{x}_{*,\mathrm{o}} = (0, 0, 0, 0, 0, 0, 0, 0)^{\mathrm{T}}$.

Kapitel 7

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde das bestärkende Lernen erfolgreich zur Steuerung und Regelung nichtlinearer dynamischer Systeme eingesetzt.

Zu Beginn wurden die Grundlagen der optimalen Steuerung und des maschinellen Lernens erläutert, die für die Betrachtungen der Arbeit relevant sind. Im Gegensatz zum herkömmlichen Vorgehen in der Regelungs- und Steuerungstheorie wurde dabei ein stochastischer Ansatz verfolgt. Anschließend wurden ausgewählte Grundkonzepte des bestärkende Lernens erläutert. Es wurde ein detaillierter Einblick in Policy-Gradients (PG) geliefert, der für ein tiefergehendes Verständnis der Materie unerlässlich ist. Es wurde gezeigt, wie KNN verwendet werden können, um Rückführungen und Restkosten zu repräsentieren und daraus ein Lernsystem zu konstruieren.

Aufbauend auf den eingeführten Konzepten wurden drei Algorithmen des tiefen bestärkenden Lernens näher beschrieben. NFQ und DQN eignen sich nur für diskrete Eingangsräume, weshalb DDPG für die weiteren Untersuchungen ausgewählt wurde. Um DDPG mit einem Ansatz der Regelungs- und Steuerungstheorie vergleichen zu können, wurde der iLQR als modellbasierter Ansatz zur Lösung des Optimalsteuerungsproblems (2.19) hergeleitet und detailliert untersucht. Weiterhin wurde aufgezeigt, wie Eingangsbeschränkungen explizit in der Optimierung berücksichtigt werden können.

Abschließend wurden DDPG und iLQR auf verschiedene nichtlineare mechanische Beispielsysteme angewendet. Mit DDPG konnten für alle Systeme – bis auf das Doppel- und Dreifach-Wagen-Pendel – eine *globale* Rückführung erlernt werden, die das Optimalsteuerungsproblem (2.20) approximativ löst. Der Berechnungsaufwand, der bei der simulativen Untersuchung von DDPG überwog, war das Training der verwendeten KNN. Dies war einer der Hauptfaktoren, warum der Lernvorgang beim Doppel-Wagen-Pendel nach rund 1.5 Millionen Zeitschritten abgebrochen wurde.

Auch wenn DDPG – wie alle modellfreien[1] Ansätze – datenineffizient ist, wird eine gute Näherungslösung des Optimalsteuerungsproblems gefunden. Auch eine Anwendung

[1] Modellfrei bezieht sich an dieser Stelle darauf, dass der Algorithmus nicht versucht das dynamische Verhalten des Systems zu erlernen.

auf einem realen System scheint, bei Betrachtung der benötigten Interaktionszeit mit dem System, nicht unrealistisch. Es ist jedoch eventuell nötig mehrere Lernvorgänge mit unterschiedlich angesetzten Kosten durchzuführen. Dies verlängert die benötigte Interaktionszeit mit dem System, weshalb bei vorhandenem Systemmodell zunächst Simulationsstudien durchgeführt werden sollten.

Stabilitätskriterien, wie sie in der Regelungs- und Steuerungstheorie von zentraler Bedeutung sind, können für die Systeme des bestärkenden Lernens aufgrund der verwendeten Approximationsmethoden oft nur schwer oder gar nicht hergeleitet werden. Um diesem Dilemma zu entgehen, gibt es Bestrebungen, moderne Datentechnologien unter Einbeziehung von Expertenwissen zu nutzen, um selbstlernende Regler zu entwerfen, die definierten Stabilitätskriterien genügen [24]. Aber auch für modellfreie Methoden des bestärkenden Lernens ist es erstrebenswert, Aussagen hinsichtlich der Stabilität des Lernvorgangs sowie der erlernten Regler zu treffen [3].

Ein generelles Problem besteht darin, dass Lernsysteme des maschinellen Lernens hinsichtlich ihrer Funktionalität eine sehr geringe Transparenz aufweisen, schwer interpretierbar sind und sich leicht manipulieren lassen [64]. Dadurch wird deren Einsatz in sicherheitskritischen Anwendungen problematisch. Derzeit gibt es Bestrebungen diesem Missstand entgegenzuwirken und Methoden zu entwickeln, um die Entscheidungen eines Lernsystems transparent zu machen. So soll eine Überprüfung der Entscheidungsfaktoren ermöglicht werden [76, 39, 54].

Abschließend ist festzuhalten, dass die untersuchten Methoden des bestärkenden Lernens und der Steuerungs- und Regelungstheorie jeweils Vor- und Nachteile haben. Es scheint daher sinnvoll die Methoden zu kombinieren und so die Schwächen zu kompensieren. Für die betrachteten Beispielsysteme scheint es sinnvoll Lernsysteme zu entwerfen, die Wissen über die Systemdynamik aufbauen und dieses nutzen um gezielte Entscheidungen zu treffen. Dieser Ansatz wird auch in der aktuellen Forschung verfolgt [21, 26, 45]. So kann die Dateneffizienz der modellfreien Ansätze des bestärkenden Lernens verbessert und eine Anwendung auf realen Systemen ermöglicht werden. Ein aktueller Ansatz der Systemidentifikation für nichtlineare dynamische Systeme [8] könnte hier interessant sein.

Weiterhin wäre es möglich, die Daten der Aufwärmphase bei DDPG nicht durch zufällige Interaktion mit dem System zu generieren, sondern mit mittels iLQR – für verschiedene Anfangswerte – bestimmten Trajektorien. So kann möglicherweise die Erkundung beschleunigt werden. Zudem könnte eine mittels Imitationslernen bestimmte Rückführung zur Initialisierung eines modellfreien Ansatzes, wie DDPG, benutzt werden. Als *Experten* könnten der LQR-Entwurf oder eine Trajektorienplanung genutzt werden, wie bspw. in [51].

Die Untersuchungen der Arbeit beschränken sich auf deterministische Systeme, da die theoretische Ausarbeitung aber auch stochastische Systeme zulässt, ist von Interesse, wie sich die vorgestellten Lernsysteme unter dem Einfluss von Störgrößen verhalten.

Literatur

[1] Marcin Andrychowicz u. a. “Learning Dexterous In-Hand Manipulation”. In: (2018). arXiv: 1808.00177.

[2] Alberto Bemporad u. a. “The explicit linear quadratic regulator for constrained systems”. In: *Automatica* 38.1 (2002). DOI: 10.1016/S0005-1098(01)00174-1.

[3] Felix Berkenkamp u. a. “Safe model-based reinforcement learning with stability guarantees”. In: *Advances in Neural Information Processing Systems* (2017), S. 908–918. arXiv: 1705.08551.

[4] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. 3. Aufl. Bd. 1. Bellmont, MA: Athena Sientific, 2005.

[5] Dimitri P. Bertsekas. *Nonlinear Programming*. 2. Aufl. Bellmont, MA: Athena Scientific, 1999.

[6] Mariusz Bojarski u. a. “End to end learning for self-driving cars”. In: (2016). arXiv: 1604.07316.

[7] Steven Boyd. *Lecture 5 - Linear Quadratic Stochastic Control*. 2008. URL: https://stanford.edu/class/ee363/lectures/stoch_lqr.pdf (besucht am 17. 06. 2019).

[8] Steven L. Brunton, Joshua L. Proctor und J. Nathan Kutz. “Sparse Identification of Nonlinear Dynamics with Control (SINDYc)”. In: *IFAC-PapersOnLine* 49.18 (2016). 10th IFAC Symposium on Nonlinear Control Systems NOLCOS 2016. DOI: https://doi.org/10.1016/j.ifacol.2016.10.249.

[9] Murray Campbell, A. Joseph Hoane Jr. und Feng-Hsiung Hsu. “Deep Blue”. In: *Artificial Intelligence* 134.1-2 (2002), S. 57–83. DOI: 10.1016/S0004-3702(01)00129-1.

[10] François Chollet u. a. *Keras*. 2015. URL: https://keras.io (besucht am 17. 06. 2019).

[11] Anna Choromanska u. a. “The loss surfaces of multilayer networks”. In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS* 38 (2015).

[12] Thomas H. Cormen u. a. *Introduction to algorithms*. MIT Press, 2009.

[13] Marc Deisenroth und Carl E. Rasmussen. “PILCO: A model-based and data-efficient approach to policy search”. In: *Proceedings of the 28th International Conference on Machine Learning, ICML* (2011).

[14] P. Dorato und A. Levis. “Optimal linear regulators: The discrete-time case”. In: *IEEE Transactions on Automatic Control* 16.6 (Dez. 1971). DOI: 10.1109/TAC.1971.1099832.

[15] Lars Gaede. *Watson, wir haben ein Problem*. 22. Sep. 2016. URL: https://www.zeit.de/wirtschaft/2016-09/kuenstliche-intelligenz-maschinen-menschenersatz-jobs (besucht am 17. 06. 2019).

[16] T. Glück, A. Eder und A. Kugi. “Swing-up control of a triple pendulum on a cart with experimental validation”. In: *Automatica* 49.3 (2013). DOI: 10.1016/j.automatica.2012.12.006.

[17] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (besucht am 17. 06. 2019).

[18] Ian Goodfellow u. a. “Generative adversarial nets”. In: *Advances in Neural Information Processing Systems* (2014), S. 2672–2680. arXiv: 1406.2661.

[19] Knut Graichen. *Skript - Methoden der Optimierung und optimalen Steuerung. Wintersemester 2018/2019*. Institut für Mess–, Regel– und Mikrotechnik, Uni Ulm, 2018.

[20] Lars Grüne und Jürgen Pannek. *Nonlinear Model Predictive Control*. Springer International Publishing, 2017. DOI: 10.1007/978-3-319-46024-6.

[21] Shixiang Gu u. a. “Continuous deep q-learning with model-based acceleration”. In: *International Conference on Machine Learning* (2016), S. 2829–2838. arXiv: 1603.00748.

[22] Shixiang Gu u. a. “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”. In: *2017 IEEE International Conference on Robotics and Automation, ICRA* (2017), S. 3389–3396. arXiv: 1610.00633.

[23] Hado van Hasselt, Arthur Guez und David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: (2015). arXiv: 1509.06461.

[24] Michael Hertneck u. a. “Learning an approximate model predictive controller with guarantees”. In: *IEEE Control Systems Letters* 2.3 (2018). DOI: 10.1109/LCSYS.2018.2843682.

[25] Ahmed Hussein u. a. “Imitation learning: A survey of learning methods”. In: *ACM Computing Surveys* 50.2 (2017). DOI: 10.1145/3054912.

[26] Sanket Kamthe und Marc Peter Deisenroth. “Data-efficient reinforcement learning with probabilistic model predictive control”. In: (2017). arXiv: 1706.06491.

[27] H. Jin Kim u. a. “Autonomous helicopter flight via reinforcement learning”. In: *Advances in Neural Information Processing Systems* (2004), S. 799–806.

[28] C. Knoll. *Regelungstheoretische Analyse- und Entwurfsansätze für unteraktuierte mechanische Systeme*. Dissertation, TU-Dresden 2016, epubli, Berlin bzw. online verfügbar: http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-209765.

[29] Jens Kober, J. Andrew Bagnell und Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), S. 1238–1274. DOI: 10.1177/0278364913495721.

[30] David Kriesel. *Ein kleiner Überblick über Neuronale Netze*. 2007. URL: http://www.dkriesel.com (besucht am 17. 06. 2019).

[31] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems* (2012), S. 1097–1105.

[32] Sergey Levine. *Deep Reinforcement Learning*. 2018. URL: http://rail.eecs.berkeley.edu/deeprlcourse/ (besucht am 17. 06. 2019).

[33] Sergey Levine u. a. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), S. 1334–1373.

[34] Sergey Levine u. a. “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection”. In: *The International Journal of Robotics Research* 37.4-5 (2018), S. 421–436. DOI: 10.1177/0278364917710318.

[35] Weiwei Li und Emanuel Todorov. “Iterative linear quadratic regulator design for nonlinear biological movement systems”. In: *International Conference on Informatics in Control, Automation and Robotics, ICINCO* (2004), S. 222–229.

[36] Timothy P. Lillicrap u. a. “Continuous control with deep reinforcement learning”. In: *4th International Conference on Learning Representations, ICLR* (2016). arXiv: 1509.02971.

[37] John Markoff. *Computer wins on ‘jeopardy!’: trivial, it’s not*. 2011. URL: https://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html (besucht am 17. 06. 2019).

[38] Martín Abadi u. a. *TensorFlow: Large-scale machine learning on heterogeneous systems*. 2015. URL: https://www.tensorflow.org/ (besucht am 17. 06. 2019).

[39] David Mascharka u. a. “Transparency by design: closing the gap between performance and interpretability in visual reasoning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), S. 4942–4950. DOI: 10.1109/CVPR.2018.00519.

[40] David Mayne. “A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems”. In: *International Journal of Control* 3.1 (1966), S. 85–95. DOI: 10.1080/00207176608921369.

[41] Francisco S. Melo. *Convergence of Q-learning: a simple proof*. 2001. URL: http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf (besucht am 17. 06. 2019).

[42] Thomas M. Mitchell. *Machine Learning*. 1. Aufl. McGraw-Hill, Inc., 1997.

[43] Volodymyr Mnih u. a. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), S. 529. DOI: 10.1038/nature14236.

[44] Kevin P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012.

[45] A. Nagabandi u. a. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *2018 IEEE International Conference on Robotics and Automation, ICRA* (Mai 2018). DOI: 10.1109/ICRA.2018.8463189.

[46] M. Neunert u. a. "Fast nonlinear model predictive control for unified trajectory optimization and tracking". In: *IEEE International Conference on Robotics and Automation, ICRA* (2016). DOI: 10.1109/ICRA.2016.7487274.

[47] Adam Paszke u. a. *Automatic differentiation in Pytorch*. 2017.

[48] Jan Peters und J Andrew Bagnell. "Policy gradient methods". In: *Encyclopedia of Machine Learning* (2010), S. 774–776. DOI: 10.1007/978-0-387-30164-8.

[49] Jan Peters und Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural Networks* 21.4 (2008). DOI: 10.1016/j.neunet.2008.02.003.

[50] Matthias Plappert u. a. "Parameter Space Noise for Exploration". In: *Computing Research Repository (CoRR)* (2017). arXiv: 1706.01905.

[51] Max Pritzkoleit und Patrick Rüdiger. *Oberseminar Regelungs- und Steuerungstheorie - Maschinelles Lernen zum optimierungsbasierten Entwurf von stabilisierenden Rückführungen für nichtlineare dynamische Systeme*. 2018. URL: https://github.com/mpritzkoleit/Oberseminar/blob/master/OberseminarPaper.pdf (besucht am 17. 06. 2019).

[52] Martin L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, 2014. DOI: 10.1002/9780470316887.

[53] Aravind Rajeswaran u. a. *Learning complex dexterous manipulation with deep reinforcement learning and demonstrations*. 2017. DOI: 10.15607/rss.2018.xiv.049.

[54] Marco Tulio Ribeiro, Sameer Singh und Carlos Guestrin. "Why should I trust you?: Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), S. 1135–1144. DOI: 10.1145/2939672.2939778.

[55] Martin Riedmiller. "Neural Fitted Q-Iteration - first experiences with a data efficient reinforcement learning method". In: *Machine Learning: ECML 2005. Lecture Notes in Computer Science* 3720 (2005). DOI: 10.1007/11564096_32.

[56] Klaus Röbenack. *Nichtlineare Regelungssysteme: Theorie und Anwendung der exakten Linearisierung*. Springer-Verlag, 2017. DOI: 10.1007/978-3-662-44091-9.

[57] Klaus Röbenack. *Regler-und Beobachterentwurf für nichtlineare Systeme mit Hilfe des Automatischen Differenzierens*. Shaker, 2005.

[58] Choe Sang-Hun. *Google's Computer Program Beats Lee Se-dol in Go Tournament*. 16. März 2016. URL: https://www.nytimes.com/2016/03/16/world/asia/korea-alphago-vs-lee-sedol-go.html (besucht am 17. 06. 2019).

[59] David Silver. *Reinforcement Learning: An Introduction*. 2015. URL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html (besucht am 17. 06. 2019).

[60] David Silver u. a. "Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017), S. 354. DOI: 10.1038/nature24270.

[61] David Silver u. a. "Deterministic Policy Gradient Algorithms". In: *Proceedings of the 31th International Conference on Machine Learning, ICML* 32 (2014).

[62] Satinder P. Singh, Tommi Jaakkola und Michael I. Jordan. "Learning without state-estimation in partially observable Markovian decision processes". In: *Machine Learning Proceedings 1994*. Elsevier, 1994. DOI: 10.1016/B978-1-55860-335-6.50042-8.

[63] Burrhus F. Skinner. "Operant behavior". In: *American psychologist* 18.8 (1963). DOI: 10.1037/h0045185.

[64] Jiawei Su, Danilo Vasconcellos Vargas und Sakurai Kouichi. "One Pixel Attack for Fooling Deep Neural Networks". In: *IEEE Transactions on Evolutionary Computation* (2017). DOI: 10.1109/TEVC.2019.2890858.

[65] Masashi Sugiyama. *Statistical Reinforcement Learning - Modern Machine Learning Approaches*. CRC Press, 2015.

[66] Richard S. Sutton und Andrew B. Barto. *Reinforcement Learning: An Introduction*. 2. Aufl. MIT Press, 2018.

[67] Richard S. Sutton, Andrew G. Barto und Ronald J. Williams. "Reinforcement learning is direct adaptive optimal control". In: *IEEE Control Systems* 12.2 (1992), S. 19–22. DOI: 10.1109/37.126844.

[68] Richard S. Sutton u. a. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in Neural Information Processing Systems* (1999).

[69] Yuval Tassa, Tom Erez und Emanuel Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS* (2012), S. 4906–4913. DOI: 10.1109/IROS.2012.6386025.

[70] Yuval Tassa, Nicolas Mansard und Emo Todorov. "Control-limited differential dynamic programming". In: *2014 IEEE International Conference on Robotics and Automation, ICRA* (2014), S. 1168–1175. DOI: 10.1109/ICRA.2014.6907001.

[71] Gustav Theile. *Wie man künstliche Gesichter enttarnt*. 4. März 2019. URL: https://www.faz.net/aktuell/wirtschaft/diginomics/wie-man-von-einer-ki-generierte-gesichter-enttarnt-16071253.html (besucht am 17. 06. 2019).

[72] Aaron Van Den Oord u. a. “Wavenet: A generative model for raw audio”. In: (2016). arXiv: 1609.03499.

[73] Niklas Wahlström, Thomas B. Schön und Marc Peter Deisenroth. “From Pixels to Torques: Policy Learning with Deep Dynamical Models”. In: (2015). arXiv: 1502.02251.

[74] Christopher J. Watkins und Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), S. 279–292. DOI: 10.1007/BF00992698.

[75] Ronald J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8 (1992), S. 229–256. DOI: 10.1007/BF00992696.

[76] Matthew D. Zeiler und Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European Conference on Computer Vision* (2014). DOI: 10.1007/978-3-319-10590-1_53.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Beispielsysteme

Im Folgenden sind die Beispielsysteme der vorliegenden Arbeit aufgeführt. Die Modellbildung erfolgte nach dem Lagrange-Formalismus. Im Folgenden sind die Modellgleichungen nicht explizit angeben. Es werden jedoch die Terme der potentiellen Energie V, der kinetischen Energie T, der Dissipationsfunktion D, sowie der externen und verallgemeinerten Kräfte Q_i für das jeweilige System angegeben.

Aus der Lagrange-Funktion

$$L = T - V \tag{A.1}$$

können mit den Lagrange-Gleichungen 2. Art

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} + \frac{\partial D}{\partial \dot{q}_i} = Q_i, \quad i = 1, ..., n \tag{A.2}$$

die Bewegungsgleichungen hergeleitet werden.

Für mechanische Systeme ergibt sich aus (A.2) die folgende Form [28]:

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{m}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = B(\boldsymbol{q})\boldsymbol{f}, \tag{A.3}$$

mit der Massenmatrix M, der Matrix B, die beschreibt, wie die Stellkräfte $\boldsymbol{f}$ auf das System wirken und dem Vektor $\boldsymbol{m}$, der die restlichen Terme zusammenfasst.

Anschließend können diese in ein Zustandsraummodell überführt werden, wobei sich der Zustand aus den generalisierten Koordinaten und deren Geschwindigkeiten zusammensetzt:

$$\boldsymbol{x} := \begin{pmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{pmatrix}. \tag{A.4}$$

Die Systemdynamik folgt aus der Zeitableitung von (A.4) mit (A.3):

$$\dot{\boldsymbol{x}} := \begin{pmatrix} \dot{\boldsymbol{q}} \\ \ddot{\boldsymbol{q}} \end{pmatrix} = \begin{pmatrix} \dot{\boldsymbol{q}} \\ M(\boldsymbol{q})^{-1}[B(\boldsymbol{q})\boldsymbol{f} - \boldsymbol{m}(\boldsymbol{q}, \dot{\boldsymbol{q}})] \end{pmatrix}. \tag{A.5}$$

Diese lässt sich in die allgemeine Form

$$\dot{\boldsymbol{x}} = f_c(\boldsymbol{x}, \boldsymbol{u}) \tag{A.6}$$

überführen.

A.1 Inverses Pendel

Das inverse Pendel in Abbildung 30 besteht aus einem an einer Achse aufgehängten Pendelarm. Mit der Stellgröße τ kann das Pendel um die Achse gedreht werden. Aufgrund der Stellgrößenbeschrenkung reicht das Moment $\tau \in [-3.5\,\mathrm{N\,m}, 3.5\,\mathrm{N\,m}]$ nicht aus, um das unten hängende Pendel ohne Schwingen aufzurichten.

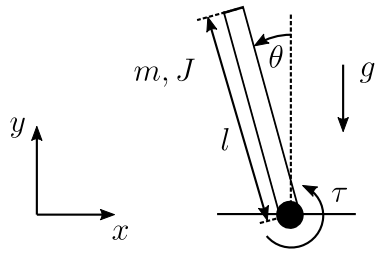

Abbildung 30 – Inverses Pendel.

Tabelle 2 – Physikalische Parameter des inversen Pendels.

Parameter		Wert
l	Pendellänge	$1.0\,\mathrm{m}$
m	Pendelmasse	$1.0\,\mathrm{kg}$
J	Rotatorisches Trägheitsmoment	$1.0\,\mathrm{kgm^2}$
d	Dämpfungskonstante	$0.02\,\mathrm{N\,m\,s}$
$u_{1,max}$	Eingangsbeschränkung	$3.5\,\mathrm{N\,m}$

Tabelle 3 – Parameter für das Training mit DDPG.

Parameter		Wert
Δt	Abtastzeit	0.05 s
N	Zeithorizont	200 (10.0 s)
	Trainingsschritte	200000
$p(\boldsymbol{x}_0)$	Anfangszustand	$\boldsymbol{x}_0 = (\mathcal{U}(0.999\pi, 1.001\pi), \mathcal{U}(-0.001, 0.001))$
	Seed-Werte	0, 56 221, 563, 779

A.2 Akrobot

Der in Abbildung 31 dargestellte sog. Akrobot ist ein unteraktuiertes mechanisches System, bestehend aus zwei seriell aufgehängten Pendelarmen. Das Gelenk, an dem die beiden Pendelarme verbunden sind, ist aktuiert.

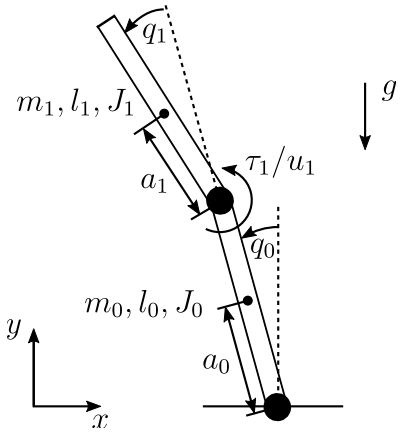

Abbildung 31 – Akrobot.

Tabelle 4 – Zur Herleitung der Bewegungsgleichungen des Akrobot verwendete Größen.

Körper	zwei Pendelarme
Generalisierte Koordinaten $\boldsymbol{q}$	Winkel q_0, Winkel q_1
Zustandsvektor $\boldsymbol{x}$	$\boldsymbol{x} = \begin{pmatrix} q_0 & q_1 & \dot{q}_0 & \dot{q}_1 \end{pmatrix}$
Eingangsvektor $\boldsymbol{u}$	$\boldsymbol{u} = \begin{pmatrix} \ddot{q}_1 \end{pmatrix}$
Stellgröße	Drehmoment τ_1
Positionsvektoren der Massenschwerpunkte	$\boldsymbol{p}_0 = \begin{pmatrix} -a_0 \sin(q_0) \\ a_0 \cos(q_0) \end{pmatrix} =: \begin{pmatrix} p_{x,0} \\ p_{y,0} \end{pmatrix}$ $\boldsymbol{p}_1 = \begin{pmatrix} -l_0 \sin(q_0) - a_1 \sin(q_0 + q_1) \\ l_0 \cos(q_0) + a_1 \cos(q_0 + q_1) \end{pmatrix} =: \begin{pmatrix} p_{x,1} \\ p_{y,1} \end{pmatrix}$
Kinetische Energie T	$\frac{1}{2} \left(m_0 \langle \dot{\boldsymbol{p}}_0, \dot{\boldsymbol{p}}_0 \rangle + J_0 \dot{q}_0^2 + m_1 \langle \dot{\boldsymbol{p}}_1, \dot{\boldsymbol{p}}_1 \rangle + J_1 (\dot{q}_0 + \dot{q}_1)^2 \right)$
Potentielle Energie V	$m_0 g p_{y,0} + m_1 g p_{y,1}$
Dissipationsfunktion D	$D = \frac{1}{2}(d_0 \dot{q}_0^2 + d_1 \dot{q}_1^2)$
Verallg. ext. Kräfte Q_i	$Q_0 = 0, \quad Q_1 = \tau_1$

Tabelle 5 – Physikalische Parameter des Akrobots.

Parameter		Wert
a_0	Abstand des Massenschwerpunkts	$0.25\,\mathrm{m}$
a_1	Abstand des Massenschwerpunkts	$0.25\,\mathrm{m}$
m_0	Pendelmasse	$0.3583\,\mathrm{kg}$
m_1	Pendelmasse	$0.3583\,\mathrm{kg}$
l_0	Pendellänge	$2a_0 = 0.5\,\mathrm{m}$
d_0	Dämpfungskonstante	$6.588 \times 10^{-3}\,\mathrm{N\,m\,s}$
d_1	Dämpfungskonstante	$6.588 \times 10^{-3}\,\mathrm{N\,m\,s}$
J_0	Rotatorisches Trägheitsmoment	$37.9999 \times 10^{-3}\,\mathrm{kgm^2}$
J_1	Rotatorisches Trägheitsmoment	$37.9999 \times 10^{-3}\,\mathrm{kgm^2}$
$u_{1,max}$	Eingangsbeschränkung	$5.0\,\mathrm{m/s^2}$

Tabelle 6 – Parameter für das Training mit DDPG.

Parameter		Wert
Δt	Abtastzeit	0.03 s
N	Zeithorizont	330 (10.0 s)
	Trainingsschritte	500000
$p(\boldsymbol{x}_0)$	Anfangszustand	$\boldsymbol{x}_0 = (\mathcal{U}(0.99\pi, 1.01\pi), \mathcal{U}(-0.01, 0.01), 0, 0)$

A.3 Wagen-Pendel

Das in Abbildung 32 dargestellte Wagen-Pendel besteht aus einem an einen Wagen aufgehängten, freischwingenden Pendelarm. Der Stelleingriff ist die in x-Richtung wirkende Kraft F_0, die durch einen Riemen auf den Wagen übertragen wird.

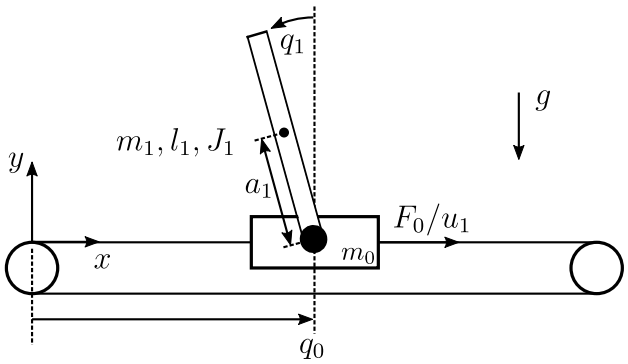

Abbildung 32 – Wagen-Pendel.

Tabelle 7 – Zur Herleitung der Bewegungsgleichungen des Wagen-Pendels verwendete Größen.

Körper	ein Wagen, ein Pendelarm
Generalisierte Koordinaten $\boldsymbol{q}$	Position des Wagens q_0, Winkel q_1
Zustandsvektor $\boldsymbol{x}$	$\boldsymbol{x} = \begin{pmatrix} q_0 & q_1 & \dot{q}_0 & \dot{q}_1 \end{pmatrix}$
Eingangsvektor $\boldsymbol{u}$	$\boldsymbol{u} = \begin{pmatrix} \ddot{q}_0 \end{pmatrix}$
Stellgröße	Kraft F_1
Positionsvektoren der Massenschwerpunkte	$\boldsymbol{p}_0 = \begin{pmatrix} q_0 \\ 0 \end{pmatrix} =: \begin{pmatrix} p_{x,0} \\ p_{y,0} \end{pmatrix}$ $\boldsymbol{p}_1 = \boldsymbol{p}_0 + \begin{pmatrix} -a_1 \sin(q_1) \\ a_1 \cos(q_1) \end{pmatrix} =: \begin{pmatrix} p_{x,1} \\ p_{y,1} \end{pmatrix}$
Kinetische Energie T	$\frac{1}{2}\left(m_0 \langle \dot{\boldsymbol{p}}_0, \dot{\boldsymbol{p}}_0 \rangle + J_1 \dot{q}_1^2 + m_1 \langle \dot{\boldsymbol{p}}_1, \dot{\boldsymbol{p}}_1 \rangle\right)$
Potentielle Energie V	$m_1 g p_{y,1}$
Dissipationsfunktion D	$D = \frac{1}{2}(d_0 \dot{q}_0^2 + d_1 \dot{q}_1^2)$
Verallg. ext. Kräfte Q_i	$Q_0 = F_1, \quad Q_1 = 0$

Tabelle 8 – Parameter des Wagen-Pendels.

Parameter		Wert
a_1	Abstand des Massenschwerpunkts	$0.43\,\mathrm{m}$
m_0	Wagenmasse	$3.34\,\mathrm{kg}$
m_1	Pendelmasse	$0.3583\,\mathrm{kg}$
l_1	Pendellänge	$0.5\,\mathrm{m}$
d_0	Dämpfungskonstante	$0.1\,\mathrm{kg/s}$
d_1	Dämpfungskonstante	$6.588 \times 10^{-3}\,\mathrm{N\,m\,s}$
J_1	Rotatorisches Trägheitsmoment	$37.9999 \times 10^{-3}\,\mathrm{kg\,m^2}$
$u_{1,max}$	Eingangsbeschränkung	$10.0\,\mathrm{m/s^2}$

Tabelle 9 – Parameter für das Training mit DDPG.

Parameter		Wert
Δt	Abtastzeit	0.02 s
N	Zeithorizont	500 (10.0 s)
	Trainingsschritte	500000
$p(\boldsymbol{x}_0)$	Anfangszustand	$\boldsymbol{x}_0 = (\mathcal{U}(-0.001, 0.001), \mathcal{U}(0.999\pi, 1.001\pi), 0, 0)$

A.4 Doppel- und Dreifach-Wagen-Pendel

Das Doppel- und Dreifach-Wagen-Pendel stellen eine Erweiterung des Wagen-Pendels dar, bei der eine serielle Pendelkette mit zwei, respektive drei Pendelarmen an einem fahrbaren Wagen aufgehangen ist.

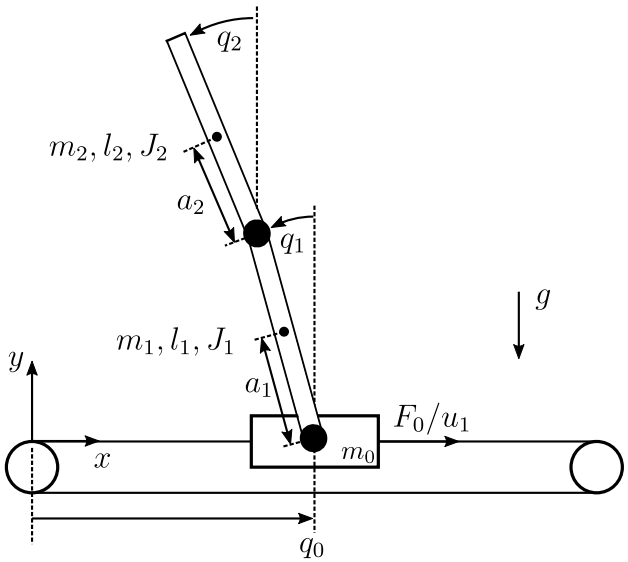

Abbildung 33 – Doppel-Wagen-Pendel.

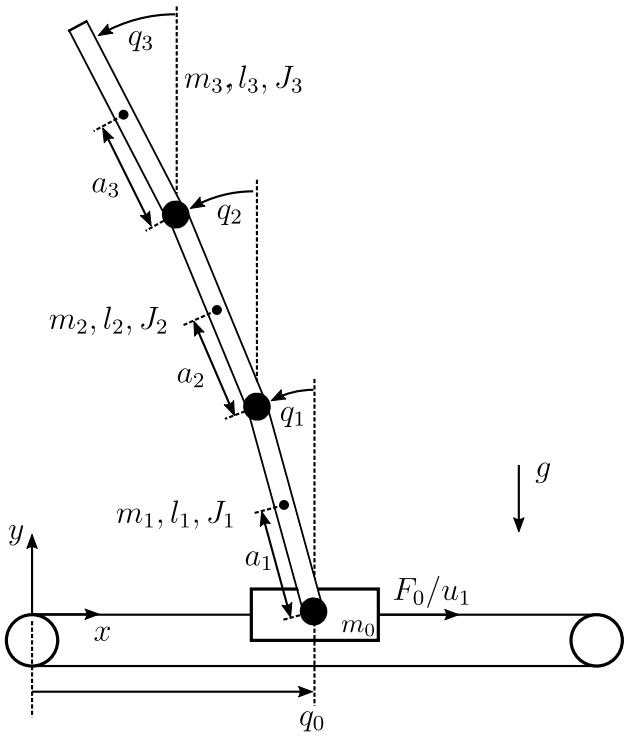

Abbildung 34 – Dreifach-Wagen-Pendel.

Tabelle 10 – Zur Herleitung der Bewegungsgleichungen des Doppel-Wagen-Pendels verwendete Größen.

Körper	ein Wagen, zwei serielle Pendelarme
Generalisierte Koordinaten $\boldsymbol{q}$	Position des Wagens q_0, Winkel q_1, Winkel q_2
Zustandsvektor $\boldsymbol{x}$	$\boldsymbol{x} = \begin{pmatrix} q_0 & q_1 & q_2 & \dot{q}_0 & \dot{q}_1 & \dot{q}_2 \end{pmatrix}$
Eingangsvektor $\boldsymbol{u}$	$\boldsymbol{u} = \begin{pmatrix} \ddot{q}_0 \end{pmatrix}$
Stellgröße	Kraft F_1
Positionsvektoren der Massenschwerpunkte	$\boldsymbol{p}_0 = \begin{pmatrix} q_0 \\ 0 \end{pmatrix} = \begin{pmatrix} p_{x,0} \\ p_{y,0} \end{pmatrix}$
	$\boldsymbol{p}_1 = \begin{pmatrix} q_0 - a_1 \sin(q_1) \\ a_0 \cos(q_1) \end{pmatrix} =: \begin{pmatrix} p_{x,1} \\ p_{y,1} \end{pmatrix}$
	$\boldsymbol{p}_2 = \begin{pmatrix} q_0 - l_1 \sin(q_1) - a_2 \sin(q_2) \\ l_1 \cos(q_1) + a_2 \cos(q_2) \end{pmatrix} =: \begin{pmatrix} p_{x,2} \\ p_{y,2} \end{pmatrix}$
Kinetische Energie T	$\frac{1}{2}\left(m_0 \langle \dot{\boldsymbol{p}}_0, \dot{\boldsymbol{p}}_0 \rangle + \sum_{i=1}^{2} J_i \dot{q}_i^2 + m_i \langle \dot{\boldsymbol{p}}_i, \dot{\boldsymbol{p}}_i \rangle\right)$
Potentielle Energie V	$m_1 g p_{y,1} + m_2 g p_{y,2}$
Dissipationsfunktion D	$D = \frac{1}{2}(d_0 \dot{q}_0^2 + d_1 \dot{q}_1^2 + d_2(\dot{q}_2 - \dot{q}_1)^2)$
Verallg. ext. Kräfte Q_i	$Q_0 = F_0, \quad Q_1 = 0, \quad Q_2 = 0$

Tabelle 11 – Zur Herleitung der Bewegungsgleichungen des Dreifach-Wagen-Pendels verwendete Größen.

Körper	ein Wagen, drei serielle Pendelarme
Gen. Koordinaten $\boldsymbol{q}$	Position des Wagens q_0, Winkel q_1, Winkel q_2, Winkel q_3
Zustandsvektor $\boldsymbol{x}$	$\boldsymbol{x} = \begin{pmatrix} q_0 & q_1 & q_2 & q_3 & \dot{q}_0 & \dot{q}_1 & \dot{q}_2 & \dot{q}_3 \end{pmatrix}$
Eingangsvektor $\boldsymbol{u}$	$\boldsymbol{u} = \begin{pmatrix} \ddot{q}_0 \end{pmatrix}$
Stellgröße	Kraft F_1
Positionsvektoren der Massenschwerpunkte	$\boldsymbol{p}_0 = \begin{pmatrix} q_0 \\ 0 \end{pmatrix} =: \begin{pmatrix} p_{x,0} \\ p_{y,0} \end{pmatrix}$ $\boldsymbol{p}_1 = \begin{pmatrix} q_0 - a_1 \sin(q_1) \\ a_0 \cos(q_1) \end{pmatrix} =: \begin{pmatrix} p_{x,1} \\ p_{y,1} \end{pmatrix}$ $\boldsymbol{p}_2 = \begin{pmatrix} q_0 - l_1 \sin(q_1) - a_2 \sin(q_2) \\ l_1 \cos(q_1) + a_2 \cos(q_2) \end{pmatrix} =: \begin{pmatrix} p_{x,2} \\ p_{y,2} \end{pmatrix}$ $\boldsymbol{p}_3 = \begin{pmatrix} q_0 - l_1 \sin(q_1) - l_2 \sin(q_2) - a_3 \sin(q_3) \\ l_1 \cos(q_1) + l_2 \cos(q_2) + a_3 \cos(q_3) \end{pmatrix} =: \begin{pmatrix} p_{x,3} \\ p_{y,3} \end{pmatrix}$
Kinetische Energie T	$\frac{1}{2} \left(m_0 \langle \dot{\boldsymbol{p}}_0, \dot{\boldsymbol{p}}_0 \rangle + \sum_{i=1}^{3} J_i \dot{q}_i^2 + m_i \langle \dot{\boldsymbol{p}}_i, \dot{\boldsymbol{p}}_i \rangle \right)$
Potentielle Energie V	$m_1 g p_{y,1} + m_2 g p_{y,2} + m_3 g p_{y,3}$
Dissipationsfunktion D	$D = \frac{1}{2} \left(d_0 \dot{q}_0^2 + d_1 \dot{q}_1^2 + d_2 (\dot{q}_2 - \dot{q}_1)^2 + d_3 (\dot{q}_3 - \dot{q}_2)^2 \right)$
Verallg. ext. Kräfte Q_i	$Q_0 = F, \quad Q_1 = 0, \quad Q_2 = 0, \quad Q_3 = 0$

Tabelle 12 – Parameter des Doppel- und Dreifach-Wagen-Pendels nach [16].

Parameter		Wert
a_1	Abstand des Massenschwerpunkts	$0.215\,\mathrm{m}$
a_2	Abstand des Massenschwerpunkts	$0.269\,\mathrm{m}$
a_3	Abstand des Massenschwerpunkts	$0.226\,\mathrm{m}$
m_0	Wagenmasse	$3.34\,\mathrm{kg}$
m_1	Pendelmasse	$0.876\,\mathrm{kg}$
m_2	Pendelmasse	$0.938\,\mathrm{kg}$
m_3	Pendelmasse	$0.553\,\mathrm{kg}$
l_1	Pendellänge	$0.323\,\mathrm{m}$
l_2	Pendellänge	$0.419\,\mathrm{m}$
l_3	Pendellänge	$0.484\,\mathrm{m}$
d_0	Dämpfungskonstante	$0.1\,\mathrm{kg/s}$
d_1	Dämpfungskonstante	$0.215\,\mathrm{N\,m\,s}$
d_2	Dämpfungskonstante	$0.002\,\mathrm{N\,m\,s}$
d_3	Dämpfungskonstante	$0.002\,\mathrm{N\,m\,s}$
J_1	Rotatorisches Trägheitsmoment	$13.0 \times 10^{-3}\,\mathrm{kg\,m^2}$
J_2	Rotatorisches Trägheitsmoment	$24.0 \times 10^{-3}\,\mathrm{kg\,m^2}$
J_3	Rotatorisches Trägheitsmoment	$18.0 \times 10^{-3}\,\mathrm{kg\,m^2}$
$u_{1,max}$	Eingangsbeschränkung	$20.0\,\mathrm{m/s^2}$