Oberseminar Regelungs- und Steuerungstheorie

Maschinelles Lernen zum optimierungsbasierten Entwurf von stabilisierenden Rückführungen für nichtlineare dynamische Systeme

Max Pritzkoleit, Patrick Rüdiger
Institut für Regelungs- und Steuerungstheorie
Fakultät für Elektrotechnik und Informationstechnik
Technische Universität Dresden
{max.pritzkoleit, patrick.ruediger}@mailbox.tu-dresden.de

Zusammenfassung

Wir vergleichen den linear-quadratischen Regler (LQR), eine klassische Methode der linearen Regelungstheorie, mit modellbasierten und modellfreien Methoden des maschinellen Lernens zum optimierungsbasierten Entwurf von stabilisierenden Rückführungen für nichtlineare dynamische Systeme am Beispiel eines zweirädrigen einachsigen aufrechtfahrenden mobilen Roboters. Die vorgestellten Methoden eignen sich grundsätzlich für diesen Zweck. Es haben sich jedoch Unterschiede bezüglich der Regelgüte gezeigt.

Schlagwörter: *Maschinelles Lernen, Neural-Fitted Q-Iteration (NFQ), optimale Steuerung, LQR, nichtlineare dynamische Systeme*

1. EINLEITUNG

Im Wintersemester 2017/18 wurden im Rahmen des Oberseminars im Diplomstudiengang Elektrotechnik der Technischen Universität Dresden Untersuchungen zur Anwendung von Methoden des maschinellen Lernens in der Regelungs- und Steuerungstheorie durchgeführt.

In den letzten Jahren konnten mit den Methoden des maschinellen Lernens erstaunliche Fortschritte auf verschiedenen Gebieten erzielt werden [1]. Vor diesem Hintergrund scheint es sinnvoll, die Anwendbarkeit einzelner Methoden des maschinellen Lernens auf regelungstechnische Probleme zu untersuchen.

Die beiden von uns untersuchten Methoden stammen aus dem Gebiet des modellfreien bestärkenden bzw. des modellbasierten überwachten Lernens. In beiden Fällen ist es das Ziel, nichtlineare dynamische Systeme mithilfe einer optimalen Zustandsrückführung um eine instabile Ruhelage zu stabilisieren.

Der Ansatz des modellfreien bestärkenden Lernens wurde gewählt, da hierbei die Modellbildung als ein aufwendiger Entwurfsschritt beim gewöhnlichen Reglerentwurf entfällt. Zwar muss für die Simulation bei der Verfahrensentwicklung noch ein Modell gefunden werden, aber auf einem realen System implementiert, kann darauf verzichtet werden.

Der zweite Ansatz kombiniert Methoden des modellbasierten überwachten Lernens mit Methoden der optimalen Steuerung zur Trajektorienplanung unter Ausnutzung des Systemmodells. Motivation und Ziel ist der Entwurf einer optimalen stabilisierenden statischen Zustandsrückführung in Form einer explizit darstellbaren Funktion, welche die Ergebnisse einer aufwendigen Offline-Trajektorienplanung für beliebige Auslenkungen mit ausreichender Genauigkeit approximiert.

Übergeordnet soll neben der Frage der Realisierbarkeit beider Ansätze der Kompromiss aus Regelgüte und Implementierungsaufwand mit dem klassischen LQR-Entwurf anhand eines Beispielsystems verglichen werden. Das betrachtete Beispielsystem ist ein inverses Pendel auf Rädern (WIP, engl. wheeled inverted pendulum), ein zweirädriger einachsiger aufrecht in der Ebene fahrender Roboter. Die Untersuchung erfolgte simulativ, die Implementierung in *Python*.

In Abschnitt 2 erläutern wir die theoretischen Grundlagen der untersuchten Methoden um diese anschließend in Abschnitt 3 zu erläutern und in Abschnitt 4 miteinander zu vergleichen.

2. THEORETISCHE GRUNDLAGEN

2.1. OPTIMALE STEUERUNG

Gegeben sei ein unbeschränktes Optimalsteuerungsproblem der Form

$$\min_{\mathbf{u}(\cdot)} J(\mathbf{u}) = k\left(\mathbf{x}_\mathrm{f}\right) + \int_{t=t_0}^{t_\mathrm{f}} l\left(\mathbf{x}, \mathbf{u}\right) dt \tag{1}$$

mit Zustand $\mathbf{x}(t) \in \mathbb{R}^n$, Steuergröße $\mathbf{u}(t) \in \mathbb{R}^m$, beliebigem Anfangszustand $\mathbf{x}_0$, freiem Endzustand $\mathbf{x}_\mathrm{f}$ und fester Endzeit t_f. Dabei genüge der Zustand zu jedem Zeitpunkt $t \in [t_0, t_\mathrm{f}]$ den Systemgleichungen in Gestalt eines nichtlinearen Zustandsraummodells $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$.

Das Kostenfunktional J setzt sich gemäß Gl. (1) aus einer beliebigen Endzustandsbewertung $k(\mathbf{x}_\mathrm{f})$ und einem beliebigen Integralanteil $l(\mathbf{x},\mathbf{u})$ zusammen.

Es handelt sich um ein Problem der dynamischen Optimierung. Die Systemgleichungen entsprechen dynamischen Gleichungsbeschränkungen und können mithilfe von Lagrange-Multiplikatoren $\boldsymbol{\lambda}(t) \in \mathbb{R}^n$ im Kostenfunktional berücksichtigt werden:

$$\bar{J}(\mathbf{u}) = k(\mathbf{x}_\mathrm{f}) + \int_{t=t_0}^{t_\mathrm{f}} \Big(l(\mathbf{x},\mathbf{u}) + \boldsymbol{\lambda}^\mathrm{T}(\mathbf{f}(\mathbf{x},\mathbf{u}) - \dot{\mathbf{x}}) \Big)\, dt \quad (2)$$

Durch Anwenden der Variationsrechnung auf $\bar{J}$ und Einführen der Hamilton-Funktion $H(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}) = l(\mathbf{x},\mathbf{u}) + \boldsymbol{\lambda}^\mathrm{T}\mathbf{f}(\mathbf{x},\mathbf{u})$ erhält man notwendige Optimalitätsbedingungen für das durchGl.(1) definierte Optimalsteuerungsproblem in kompakter Form [2, vgl. S. 89ff.]:

$$\dot{\mathbf{x}} = \frac{\partial H}{\partial \boldsymbol{\lambda}} \quad (3a)$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \mathbf{x}} \quad (3b)$$

$$0 = \frac{\partial H}{\partial \mathbf{u}} \quad (3c)$$

Neben der Anfangsbedingung

$$\mathbf{x}(t_0) = \mathbf{x}_0 \quad (4a)$$

muss beim betrachteten Fall eines freien Endzustandes bei fester Endzeit zusätzlich die folgende Endbedingung erfüllt sein:

$$\boldsymbol{\lambda}(t_\mathrm{f}) = \frac{\partial k(\mathbf{x}_\mathrm{f})}{\partial \mathbf{x}_\mathrm{f}} \quad (4b)$$

Durch die notwendigen Optimalitätsbedingungen in Gl. (3) ist gemeinsam mit den Randbedingungen in Gl. (4) eine Zwei-Punkt-Randwertaufgabe (ZPRWA) definiert, deren Lösung die potentiell optimalen[1] Trajektorien $\mathbf{x}^*(t)$, $\boldsymbol{\lambda}^*(t)$ und $\mathbf{u}^*(t)$ auf $[t_0, t_\mathrm{f}]$ liefert.

2.1.1. Potentiell optimale Trajektorien für nichtlineare Systeme. Es sei angenommen, dass sich $\mathbf{u}$ durch Umstellen von Bedingung Gl. (3c) als Funktion von $\mathbf{x}$ und $\boldsymbol{\lambda}$ ausdrücken lässt:

$$\mathbf{u} = \boldsymbol{\phi}(\mathbf{x},\boldsymbol{\lambda}) \quad (5)$$

Setzt man Gl. (5) in Gl. (3a) und Gl. (3b) ein, so ergibt sich eine ZPRWA in $\mathbf{x}$ und $\boldsymbol{\lambda}$. Im Allgemeinen lässt sich diese Aufgabe nur numerisch lösen. Gängige Lösungsverfahren sind relativ aufwändig und beispielsweise in [2, S. 120ff.] beschrieben. Die Lösung liefert die Verläufe von $\mathbf{x}^*(t)$ und $\boldsymbol{\lambda}^*(t)$. Eine potentiell optimale Steuertrajektorie ergibt sich schließlich durch Einsetzen von $\mathbf{x}^*(t)$ und $\boldsymbol{\lambda}^*(t)$ in Gl. (5).

[1] Im Folgenden werden Trajektorien, welche die notwendigen Optimalitätsbedingungen in Gl. (3) erfüllen, kurz „potentiell optimal“ genannt. Eine tatsächliche Optimalität dieser Trajektorien ist nicht unbedingt gegeben, da die Bedingungen in Gl. (3) nur unter bestimmten Voraussetzungen auch hinreichend sind.

2.1.2. Optimale Zustandsrückführung für LTI-Systeme bei quadratischem Kostenfunktional. Für lineare zeitinvariante Systeme der Form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ mit $(\mathbf{A},\mathbf{B})$ stabilisierbar, $l(\mathbf{x},\mathbf{u}) = \mathbf{x}^\mathrm{T}\mathbf{Q}\mathbf{x} + \mathbf{u}^\mathrm{T}\mathbf{R}\mathbf{u}$ mit symmetrischen, positiv definiten Wichtungsmatrizen $\mathbf{Q}$ und $\mathbf{R}$ mit $(\mathbf{A},\mathbf{C})$ detektierbar, wobei $\mathbf{Q} = \mathbf{C}^\mathrm{T}\mathbf{C}$ sowie $t_\mathrm{f} \to \infty$ und damit notwendigerweise $\mathbf{x}(t_\mathrm{f}) = 0$ und folglich $k(\mathbf{x}_\mathrm{f}) = \mathbf{x}_\mathrm{f}^\mathrm{T}\mathbf{S}\mathbf{x}_\mathrm{f} = 0$ ergibt sich die optimale Steuergröße $\mathbf{u}^*(t)$ für alle t aus dem optimalen Rückführgesetz

$$\mathbf{u}^*(t) = -\mathbf{K}^*\mathbf{x}(t) \quad (6)$$

mit der optimalen Rückführverstärkungsmatrix

$$\mathbf{K}^* = \mathbf{R}^{-1}\mathbf{B}^\mathrm{T}\mathbf{P}, \quad (7)$$

wobei $\mathbf{P}$ die eindeutige positiv semidefinite Lösung der algebraischen Riccati-Gleichung (ARE, engl. algebraic Riccati equation)

$$\mathbf{P}\mathbf{A} + \mathbf{A}^\mathrm{T}\mathbf{P} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\mathrm{T}\mathbf{P} + \mathbf{Q} = 0 \quad (8)$$

ist [2, vgl. S. 102ff.]. Eine derartige optimale Zustandsrückführung ist auch als linear-quadratischer Regler (LQR) bekannt und in der Praxis beliebt.

Unter o. g. Einschränkungen folgen Existenz und Eindeutigkeit der optimalen Zustandsrückführung, welche das System aus beliebigen Anfangszuständen in die Ruhelage (Ursprung) überführt, wohingegen man für den unter 2.1.1 beschriebenen allgemeineren Fall nur eine potentiell optimale Steuertrajektorie für einen bestimmten Anfangszustand erhält. Zur Lösung der ARE existieren effiziente Lösungsverfahren.

2.2. BESTÄRKENDES LERNEN

Das bestärkende Lernen (engl. reinforcement learning) kann neben dem überwachten und unüberwachten Lernen als dritte Teildisziplin des maschinellen Lernens angesehen werden [3]. Es ist jedoch theoretisch und historisch eng verzahnt mit der optimalen Steuerung und kann auch als direkte adaptive Optimalsteuerung interpretiert werden [4].

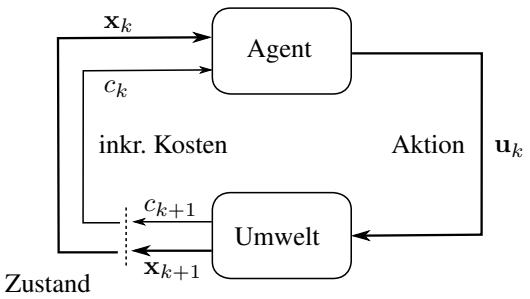

Abbildung 1. Die Agent-Umwelt Interaktion des bestärkenden Lernens (vgl. Abb. 3.1, [3])

2.2.1. Agent-Umwelt Interaktion. Beim bestärkenden Lernen agiert ein Agent mit seiner Umwelt (s. Abbildung 1).

Beide befinden sich zum Zeitpunkt k im Zustand $\mathbf{x}_k$ und der Agent kann mit einer Aktion $\mathbf{u}_k \in \mathcal{U}$ Einfluss auf die Umwelt nehmen, wodurch eine Transition $\mathbf{x}_k \to \mathbf{x}_{k+1}$ erfolgt. Daraufhin erhält dieser inkrementelle Kosten $c_{k+1} \in \mathbb{R}$. [2]

Die Aufgabe des Agenten ist es aus jedem gegeben Anfangszustand $\mathbf{x}_0 \in \mathcal{X}$ die optimale Aktionsfolge $\mathbf{u}_0^*, \mathbf{u}_1^*, ..., \mathbf{u}_T^*$ zum Endzustand $\mathbf{x}_T \in \mathcal{X}$ zu bestimmen, welche die Restkosten (engl. return) $V_k(\mathbf{x}_k) = \sum_{i=0}^{T} \gamma^i c_{k+i+1}$ mit $\gamma \in [0,1]$ minimiert. Diesen Rahmen nennt man Markov-Entscheidungsproblem (MEP).

Ein MEP ist ein Tupel $\mathcal{M} = \langle \mathcal{X}, \mathcal{U}, \mathcal{T}, \mathcal{J}, \gamma \rangle$, wobei $\mathcal{X}$ eine endliche Menge von Zuständen, $\mathcal{U}$ eine endliche Menge von Aktionen, $\mathcal{T}$ ein Transitionsmodell und $\mathcal{J}$ eine Kostenfunktion ist. Das Transitionsmodell erfüllt dabei die Markov-Eigenschaft, welche besagt, dass der Folgezustand $\mathbf{x}_{k+1}$ zum Zeitpunkt $k+1$ nur vom Istzustand $\mathbf{x}_k$ und der Aktion $\mathbf{u}_k$ abhängt. Zur Lösung des Markov-Entscheidungsproblems kann die BELLMAN-Gleichung verwandt werden:

$$V(\mathbf{x}_k) = c_k + \gamma V(\mathbf{x}_{k+1}), \quad \gamma \in [0,1]. \tag{9}$$

Bei bekanntem Modell kann dieses vom Agenten direkt zur Planung genutzt werden [5], [6], andernfalls kann eine Modellidentifikation durchgeführt werden [7]–[9]. Es gibt aber eine Vielzahl von Algorithmen, die auch ohne ein Modell der Umwelt auskommen [10]–[12]. Diese haben allerdings auch eine geringere Dateneffizienz, benötigen also eine zeitlich längere Aktion mit der Umwelt, um das MEP zu lösen. Die Dateneffizienz spielt eine entscheidende Rolle in der aktuellen Forschung des bestärkenden Lernens, um Probleme in der physischen Welt zu lösen. Ein serieller Manipulator bspw. verschleißt beim Erlernen einer Manipulationsaufgabe, weshalb der Lernalgorithmus sehr dateneffizient sein muss, um in der Praxis Anwendung zu finden [13].

2.2.2. Q-Learning. Um das Konzept der Restkosten V zu erweitern, wird eine Vorteilsfunktion $A(\mathbf{x}, \mathbf{u}) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ eingeführt. Addiert man diese mit den Restkosten erhält man die Bewertungsfunktion Q:

$$Q(\mathbf{x}, \mathbf{u}) = V(\mathbf{x}) + A(\mathbf{x}, \mathbf{u}) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}. \tag{10a}$$

Diese gibt die zu erwartenden minimalen Restkosten für ein gegebenes Zustands-Aktions-Paar $(\mathbf{x}, \mathbf{u})$ an:

$$Q(\mathbf{x}, \mathbf{u}) = \mathbb{E}[V(\mathbf{x}) | \mathbf{x} = \mathbf{x}_k, \mathbf{u} = \mathbf{u}_k,]. \tag{10b}$$

Durch die gierige Strategie [3]:

$$\pi(\mathbf{x}) = \arg\min_{\mathbf{u}^*} Q(\mathbf{x}, \mathbf{u}^*), \tag{11}$$

kann die optimale Aktion $\mathbf{u}^*$ bestimmt werden, welche bei gegebenem Zustand $\mathbf{x}$ die Restkosten minimiert. Kennt der Agent die optimale Bewertungsfunktion Q^*, kann er mit dieser Strategie die optimale Steuerfolge finden und das MEP lösen.

Die optimale Bewertungsfunktion Q^* ist ohne Modellkenntnis jedoch unbekannt, weshalb beim Q-Learning versucht wird durch Interaktion mit der Umwelt eine ausreichend genaue Approximation dieser Bewertungsfunktion zu finden. Dazu werden sogenannte Q-Updates durchgeführt:

$$\begin{aligned} Q(\mathbf{x}_k, \mathbf{u}_k) := (1-\alpha) Q(\mathbf{x}_k, \mathbf{u}_k) \\ + \alpha [c_k + \gamma \underbrace{\min_{\mathbf{u}} Q(\mathbf{x}_{k+1}, \mathbf{u})}_{V(\mathbf{x}_{k+1})}], \quad \alpha, \gamma \in [0,1]. \end{aligned} \tag{12}$$

Mit der Lernrate α kann eingestellt werden, wie stark der alte Wert von Q beibehalten wird. Der Discount-Faktor γ gibt an, wie sich die Restkosten des Folgezustands auf den Istzustand auswirken. Bei stochastischer Umwelt sollte der Wert eher kleiner gewählt werden, da bei identischer Zustandstransition von $\mathbf{x}_{k+1}$ nach $\mathbf{x}_k$ unterschiedliche Kosten auftreten können. Bei einer deterministischen Umwelt kann $\alpha \equiv 1$ gesetzt werden, wodurch sich Gl. (12) zu Gl. (9) vereinfacht:

$$Q(\mathbf{x}_k, \mathbf{u}_k) := c_k + \gamma \min_{\mathbf{u}} Q(\mathbf{x}_{k+1}, \mathbf{u}), \tag{13}$$

$$:= c_k + \gamma V(\mathbf{x}_{k+1}). \tag{14}$$

Durch diese Werte-Iteration konvergiert Q für $k \to \infty$ gegen Q^*, sofern die Umwelt nur eine diskrete endiche Zustandsmenge $\mathcal{X}$ besitzt.

Beim tabularen Q-Learning [14] wird zu Beginn des Lernprozesses jedem der möglichen Zustands-Aktions-Paare des MEPs ein Wert in einer Tabelle zugeordnet. Damit das MEP gelöst werden kann, muss bei der sequentiellen Lösungsfindung nun aber jedes dieser Paare unendlich oft besucht werden, damit Q^* gefunden wird. Dies führt zum einen zu der Limitierung, dass der Zustands- und Aktionenraum diskret und begrenzt ist, zum anderen steigt der Rechenaufwand exponentiell zur Dimension dieser. Einen Ausweg aus diesem Dilemma bieten Methoden, welche die Bewertungsfunktion approximieren, bspw. mit einem neuronalen Netzwerk [11], wie in Abschnitt 3.3 näher erläutert, oder anderen Funktionsapproximatoren, wie bspw. Polynomen [5, S. 30], Tile-Coding [6] oder Gaußprozessen [9].

2.3. KÜNSTLICHE NEURONALE NETZWERKE (KNN)

Durch die Anwendung von KNN, besonders der tiefen KNN (engl. deep neural networks), konnten in den letzten Jahren erstaunliche Fortschritte in der Bild- und Sprachverarbeitung [15], aber bspw. auch in der Robotik [16], erreicht werden, von denen einige auch außerhalb der wissenschaftlichen Gemeinschaft, für Aufsehen gesorgt haben. Dies betrifft insbesondere die Verwendung von tiefen KNN für das bestärkende Lernen (engl. deep reinforcement learning) [1],[10].

2.3.1. Mehrschichtige Perzeptronen (MLP, engl. multilayer perceptron). Eine bestimmte Form der KNN ist das MLP (s. Abbildung 3), welches nach [17] aus simplen Recheneinheiten, den Perzeptronen (s. Abbildung 2) zusammengesetzt ist. Perzeptronen können als stark vereinfachtes Modell eines Neurons aufgefasst werden.

[2] Aus der Sicht eines Regelungstechnikers kann der Agent als eine Zustandsrückführung und die Umwelt als Regelstrecke aufgefasst werden.

[3] gierig (engl. greedy), da die inkrementellen Kosten in der Literatur of als Belohnung formuliert werden

Ein Perzeptron (s. Abbildung 2) besitzt einen Eingabevektor $\mathbf{x} = (x_1, x_2, ..., x_n)^{\mathrm{T}}$, einen Gewichtsvektor $\mathbf{w}_j = (w_{1,j}, w_{2,j}, ...w_{n,j})^{\mathrm{T}}$, eine Aktivierungsfunktion f_{Akt} (bspw. $\tanh$) sowie eine Ausgabe y_j. Die Ausgabe ist der Funktionswert der Aktivierungsfunktion, deren Argument das Skalarprodukt aus Eingabevektor und Gewichtsvektor ist:

$$y_j = f_{\mathrm{Akt}}(\mathbf{w}_j^{\mathrm{T}} \mathbf{x})$$

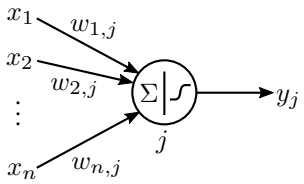

Abbildung 2. Perzeptron

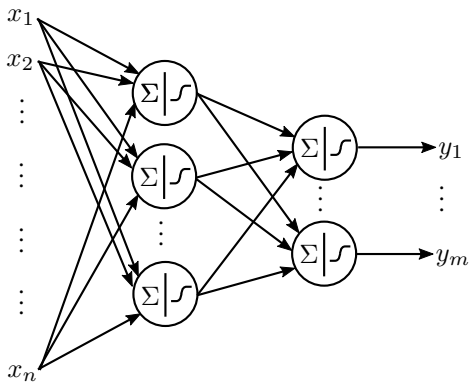

Abbildung 3. Mehrschichtiges Perzeptron

2.3.2. MLP zur Funktionsapproximation. Dreischichtige MLP sind universelle Funktionsapproximatoren und können eine beliebige nichtlineare stetige mehrdimensionale Funktion mit beliebiger Genauigkeit approximieren. Konkret wurde diese Eigenschaft für MLP mit linearer Eingabe- und Ausgabeschicht sowie bestimmten Aktivierungsfunktionen der verborgenen Schicht bewiesen. [18], [19] In der Praxis setzt dies eine geeignete Anzahl von Perzeptronen in den jeweiligen Schichten und einen geeigneten Algorithmus zur Anpassung der Gewichte voraus. Derartige Algorithmen basieren in der Regel auf Backpropagation, einem Gradientenabstiegsverfahren [20], bei dem ausgehend von der Ausgabeschicht die Gewichte derart angepasst werden, dass eine Fehlerfunktion schrittweise minimiert wird.

Es existieren leistungsfähige *Python*-Programmbibliotheken für maschinelles Lernen mit MLP. Aufgaben der Funktionsapproximation lassen sich beispielsweise mit der API *Keras* [21], welche mittlerweile fester Bestandteil von *Tensorflow* [22] ist, mit wenig Aufwand erledigen. Voraussetzung dafür ist insbesondere ein geeigneter Trainingsdatensatz, welcher einen hinreichenden Teil des Definitionsbereiches der gesuchten Funktion enthält.

3. VERWENDETE METHODEN

3.1. APPROXIMATION EINER OPTIMALEN ZUSTANDSRÜCKFÜHRUNG (optZRF)

Dieser Abschnitt behandelt den optimierungsbasierten Entwurf einer nichtlinearen statischen MLP-Zustandsrückführung zur Stabilisierung nichtlinearer dynamischer Systeme um eine instabile Ruhelage. Dazu soll eine begrenzte Anzahl optimaler Trajektorien mit dem Ansatz aus 2.1.1 berechnet und zur Approximation einer optimalen Zustandsrückführung mit Methoden des überwachten Lernens genutzt werden. Konkret soll die stetige Funktion $\mathbf{K}_{\mathrm{MLP}} : \mathbf{x} \mapsto \mathbf{u}^*$, die jedem Zustand eine optimale Steuergröße zuordnet, approximiert werden. Hierzu sei angenommen, dass das System als ideales nichtlineares Zustandsraummodell vorliegt, der Zustand zu jedem Zeitpunkt bekannt ist und keine Störgrößen auftreten. Das betrachtete System und das verwendete Kostenfunktional seien zudem so beschaffen, dass die notwendigen Optimalitätsbedingungen aus Gl. (3) auch hinreichend sind und die Existenz der gesuchten Funktion $\mathbf{K}_{\mathrm{MLP}}$ gewährleistet ist.

3.1.1. Lösung der ZPRWA. Die *Python*-Programmbibliothek *SciPy* [23] bietet unter der Funktion *integrate.solve_bvp* einen Algorithmus zur Lösung der ZPRWA aus 2.1.1. Nach Berechnung einer optimalen Trajektorie wird die Einhaltung der notwendigen Optimalitätsbedingungen, Randbedingungen und ggf. Zustandsbeschränkungen sowie Steuergrößenbeschränkungen überprüft.

3.1.2. Trainingsdatensatz. Für das vorliegende Problem sollen optimale Trajektorien von Zustand und Steuergröße ausgehend von verschiedenen Auslenkungen von der Ruhelage als Trainingsdatensatz verwendet werden. Es wird vorgeschlagen, dass hierzu eine hinreichend große Anzahl von Realisierungen N der gleichverteilten Zufallsvariable $\mathbf{X}_0 \sim \mathcal{U}(\mathbf{x}_{0\,\min}, \mathbf{x}_{0\,\max})$ verwendet und $\mathbf{x}_{0\,\min}, \mathbf{x}_{0\,\max} \in \mathbb{R}^n$ zweckmäßig gewählt wird. Dabei ist $\mathcal{A} = [\mathbf{x}_{0\,\min}, \mathbf{x}_{0\,\max}]$ die Menge aller zulässigen Auslenkungen. Es ist davon auszugehen, dass N mit zunehmender Mächtigkeit von $\mathcal{A}$ und steigender Zustandsdimension sehr schnell wachsen muss, um einen hinreichenden Teil des Zustandsraumes abdecken zu können.

3.2. LQR

Zur Stabilisierung nichtlinearer dynamischer Systeme um eine instabile Ruhelage muss zur Anwendung LQR-Entwurfs zunächst eine Linearisierung des Systems um diese Ruhelage erfolgen. Durch die beiden Wichtungsmatrizen kann Einfluss auf das Verhalten des geschlossenen Regelkreises genommen werden, wobei die beiden konfliktären Ziele quadratischer Abstand zur Ruhelage und Stellaufwand gezielt gewichtet werden können. Zur Berechnung der optimalen Rückführverstärkungsmatrix wird die Lösung der ARE benötigt. Einen effizienten Lösungsalgorithmus bietet die Funktion *linalg.solve_continuous_are* der *Python*-Programmbibliothek *SciPy*. Die Berechnung eines Vorfilters ist zur Überführung des Systems in die Ruhelage (Ursprung des linearisierten Systems) nicht nötig.

Es ist zu erwarten, dass die Regelgüte gemäß des durch die Wichtungsmatrizen definierten Kostenfunktionals für steigende Auslenkungen des Systems von der Ruhelage immer stärker vom Optimum abweicht. Ggf. bestehende Zustands- oder Steuergrößenbeschränkungen bleiben bei dieser Methode unberücksichtigt. Deren Einhaltung muss somit ggf. bei der Simulation des geschlossenen Regelkreises überprüft werden.

3.3. NEURAL FITTED Q-ITERATION (NFQ)

Ein Algorithmus, der die Probleme des Q-Learning (s. Abschnitt 2.2.2) umgeht, ist NFQ [11]. Dieser nutzt ein MLP als Funktionsapproximator für die Bewertungsfunktion Q. Und führt iterative Updates dieses Q-Netzwerks durch um eine Näherungslösung für Q^* zu finden. Der Algortihmus wurde als Ansatz für das Beispielsystem gewählt, da die Autoren in [11], sowie in [24] gezeigt haben, dass NFQ sich auf dynamische System, wie das inverse Wagenpendelsystem, anwenden lässt und der dabei erlernte Regler, eine gute Regelgüte aufweist.

3.3.1. Der Lernvorgang des Agenten. Der Lernvorgang setzt sich aus Episoden zusammen. In jeder dieser Episoden wird der Zustand des Agenten bzw. der Umwelt auf $\mathbf{x}_0 \in \mathcal{X}_0 \subseteq \mathcal{X}$ gesetzt. Der Agent kann nun für T Schritte mit der Umwelt interagieren und erhält in jedem Schritt inkrementelle Kosten $c_j, j \in [0,T]$. Für jede Transition vom Zeitschritt j zum Zeitschritt $j+1$ wird ein Tupel (u_j, x_j, x_{j+1}, c_j) in einem Datensatz $\mathcal{D}$ gespeichert. Diese Daten bilden das „Gedächtnis“ des Agenten.

Eine Episode ist beendet, wenn $k = T$ oder ein nichtzulässiger Zustand $\mathbf{x}^- \in \mathcal{X}^- \subseteq \mathcal{X}$ erreicht wird. Ob ein Zustand zulässig ist, wird beim Entwurf des Lernvorgangs durch die Kostenfunktion festgelegt:

$$c_j(\mathbf{x}_j, \mathbf{u}_j) = \begin{cases} 0 & \text{für } \mathbf{x}_j \in \mathcal{X}^+, \\ 1 & \text{für } \mathbf{x}_j \in \mathcal{X}^-, \\ 0.01 & \text{sonst.} \end{cases} \tag{15}$$

Mit dieser werden auch Zielzustände $\mathbf{x}^+ \in \mathcal{X}^+ \subseteq \mathcal{X}$ festgelegt, die der Agent möglichst schnell erreichen muss, um die Gesamtkosten einer Episode zu minimieren. Die Entscheidung die Ausgabe der Kostenfunktion auf das Intervall $[0,1]$ zu beschränken hängt mit der Struktur des MLP zusammen, welches aufgrund seiner Aktivierungsfunktion (Sigmoidfunktion) in der Ausgabeschicht nur Werte aus diesem Intervall annehmen kann.

3.3.2. Der Trainingsgsvorgang des MLP. Am Ende einer Episode werden aus jeder Transition $(u_j, x_j, x_{j+1}, c_j) \in \mathcal{D}$ ein Ein- und Ausgabepaar für das Training des Q-Netzwerks aufgestellt. Die Eingabedaten sind $\mathbf{x}_j, \mathbf{u}_j$. Mittels Gl. (12) werden durch Vorwärtsrechnung des Q-Netzwerks die Sollwerte für die Ausgabeschicht erzeugt:

$$y_j = \begin{cases} 1 & \text{für } \mathbf{x}_{j+1} \in \mathcal{X}^-, \\ c_j + \gamma \min\limits_{\mathbf{u}} \hat{Q}_n(\mathbf{x}_{j+1}, \mathbf{u}; \theta) & \text{sonst.} \end{cases} \tag{16}$$

Anhand dieser Daten wird dann das Q-Netzwerk trainiert. Dazu wird eine Regression durchgeführt, welche durch Anpassung der Gewichte θ eine vorgegebene Fehlerfunktion minimiert. Der Lernvorgang kann beschleunigt werden, in dem künstliche Zustandstransitionen eingeführt werden, die die Ausgabe des Q-Netzwerks in $\mathcal{X}^+$ auf 0 fixieren. Durch die Generalisierung des Netzwerks kann die Konvergenz des Algorithmus beschleunigt werden [25, S. 10].

initialisiere Datensatz $\mathcal{D}$;
initialisiere MLP $\hat{Q}(\mathbf{x}, \mathbf{u}; \theta_0) \leftarrow \theta_0 \sim \mathcal{U}(-0.5, 0.5)$;
for *n-Episoden* **do**
  $\mathbf{x}_k := \mathbf{x}_0 \in \mathcal{X}_0$;
  for $k = 0, T$ **do**
    Agent bestimmt Aktion:
    $\mathbf{u}_k = \pi(\mathbf{x}_k)$ - (ϵ-gierig);
    Simulation der Umwelt:
    $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$;
    Berechnung der inkrementellen Kosten:
    $c_k(\mathbf{x}_k, \mathbf{u}_k)$;
    Abspeichern der Transition:
    $(\mathbf{u}_k, \mathbf{x}_k, \mathbf{x}_{k+1}, c_k)$ in $\mathcal{D}$;
    Bestimmung der Trainingsdaten:
    $y_j = \begin{cases} 1 & \text{für } \mathbf{x}_{j+1} \in \mathcal{X}^-, \\ c_j + \gamma \min\limits_{\mathbf{u}} \hat{Q}(\mathbf{x}_{j+1}, \mathbf{u}; \theta_n) & \text{sonst.} \end{cases}$;
    $\mathbf{x}_k := \mathbf{x}_{k+1}$;
  end
  Training des MLP:
  $\theta_{n+1} := argmin_\theta \sum_j (y_j - \hat{Q}(\mathbf{x}_j, \mathbf{u}_j; \theta_n))^2$;
end

Algorithmus 1 : NFQ

Der Berechnungsaufwand von $\mathbf{u}_k$ durch

$$\min_{\mathbf{u}_k} \hat{Q}_n(\mathbf{x}_{j+1}, \mathbf{u}_k, \theta)$$

steigt mit der Anzahl möglicher Aktionen. In [10] wird dies umgangen, in dem das Q-Netzwerk für jede Aktion ein Augabeperceptron besitzt. Ein weiterer Ansatz ist das Netzwerk bezüglich $\mathbf{u}$ numerisch zu minimieren [26, S.84] und NFQ so auf kontinuierliche Aktionen zu erweitern.

4. METHODENVERGLEICH

Die drei vorgestellten Methoden werden nun auf das Beispielsystem angewendet und in Bezug auf die erreichte Regelgüte bei verschiedenen Auslenkungen von der instabilen Ruhelage verglichen.

4.1. BEISPIELSYSTEM

Betrachtet wird ein Dreikörpersystem mit zwei Rädern und einem Pendel als Einzelkörper. Abbildungen 4 und 5 zeigen dieses System. Die Systemgrößen sind Abstand des Achsmittelpunktes P_A zum Ursprung $d = \sqrt{x^2 + y^2}$, Drehwinkel um die eigene Achse φ, Auslenkwinkel des Pendelkörpers θ und den eingeprägten Raddrehmomenten τ_1, τ_2. Die Systemparameter sind die Massen der Einzelkörper $m_1 = m_2 =: m_\mathrm{w}$, $m_3 =: m_\mathrm{p}$, Trägheitsmomente bezüglich der Massenmittelpunkte der Einzelkörper $J_1 = J_2 =: J_\mathrm{w}$, $J_3 =: J_\mathrm{p}$

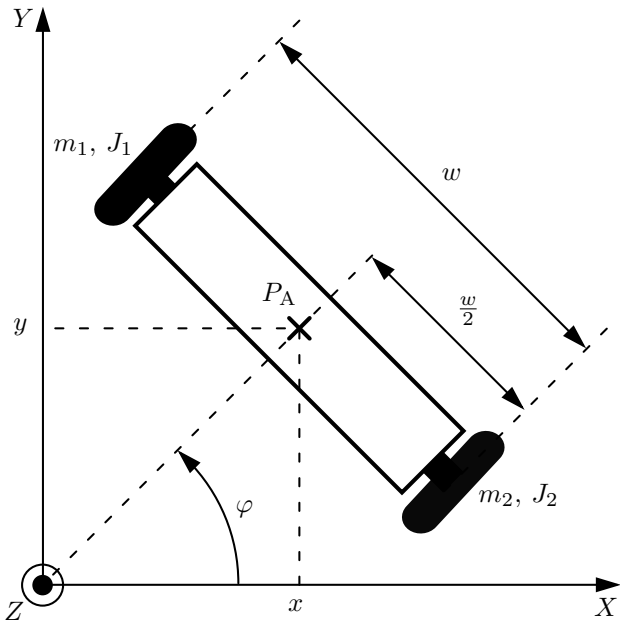

Abbildung 4. Beispielsystem (Draufsicht)

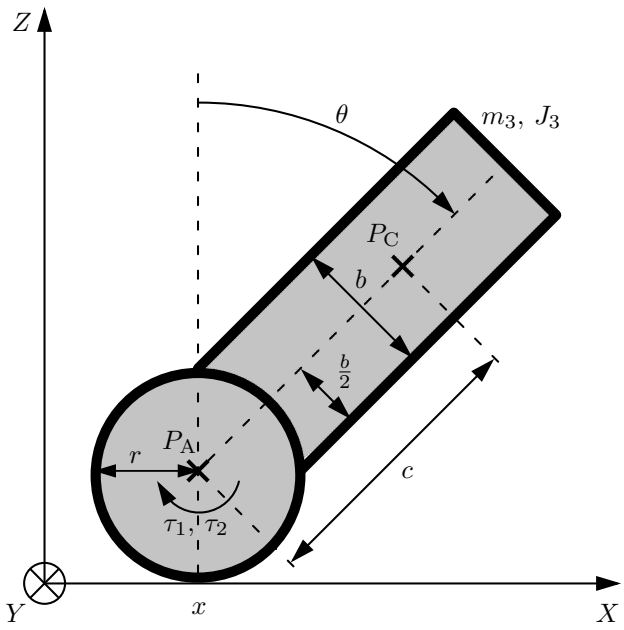

Abbildung 5. Beispielsystem (Seitenansicht)

mit P_C Massenmittelpunkt des Pendelkörpers, Erdbeschleunigung g und den geometrischen Größen $r_1 = r_2 =: r,\ b,\ c$ und w. Um die Komplexität des Systems zu begrenzen, setzen wir als Steuergröße $u := \tau := \tau_1 = \tau_2$ woraus $\varphi = 0$ und $d = x$ folgt, was einer Reduktion der Steuergrößen- und Zustandsdimension entspricht. Als Zustand wurde $\mathbf{x} := (x_1, x_2, x_3, x_4)^\mathrm{T} := \left(x, \theta, \dot{x}, \dot{\theta}\right)^\mathrm{T}$ festgelegt. Mithilfe des Lagrange-Formalismus wurden aus der kinetischen und potentiellen Energie die Bewegungsgleichungen des Systems und daraus schließlich das nichtlineare Zustandsraummodell

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u) = \begin{pmatrix} \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} \\ \mathbf{M}^{-1}(x_2) \cdot (\mathbf{F}(u) - \mathbf{C}(x_2, x_4) - \mathbf{K}(x_2)) \end{pmatrix}$$

bestimmt (s. Anhang), welches sich aus einem definitorischen Teil und den umgestellten Bewegungsgleichungen zusammensetzt. Das System ist unteraktuiert und besitzt instabile Ruhelagen bei $\mathbf{x}_\mathrm{RL} = (x_\mathrm{RL},\ 0,\ 0,\ 0)^\mathrm{T}$ mit $x_\mathrm{RL} \in \mathbb{R}$. Als Zielzustand wird die Ruhelage mit $x_\mathrm{RL} = 0$ festgelegt.

Es sei angemerkt, dass dabei die nichtholonome Zwangsbedingung $|\theta| < \frac{\pi}{2}$ der Einfachheit halber vernachlässigt wurde, da diese auch beim optimierungsbasierten Steuerungsentwurf implizit oder explizit als Zustandsbeschränkung berücksichtigt werden kann. Als Richtwert für die betragsmäßige Begrenzung der Steuergröße wird ein Drehmoment von 5 Nm verwendet.

4.2. SIMULATIONSEXPERIMENTE

Um die Regelgüte der drei beschrieben Methoden vergleichen zu können, wird das Stabilisierungsverhalten aus sieben verschieden Auslenkungen des Roboters untersucht. Grundlage des Vergleiches ist das Kostenfunktional aus (1) mit $l(\mathbf{x}, \mathbf{u}) = \mathbf{x}^\mathrm{T}\mathbf{Q}\mathbf{x} + Ru^2$ und $k(\mathbf{x}_\mathrm{f}) = \mathbf{x}_\mathrm{f}^\mathrm{T}\mathbf{S}\mathbf{x}_\mathrm{f}$ (mit $\mathbf{Q}$, R, $\mathbf{S}$ gemäß Anhang). Dieses wurde für den Entwurf einer optimalen Zustandsrückführung nach 3.1 und den LQR-Entwurf (ohne Endzustandsbewertung) verwendet. Für den NFQ-Entwurf kam davon abweichend ein diskretes Funktional mit inkrementellen Kosten gemäß (15) zum Einsatz.

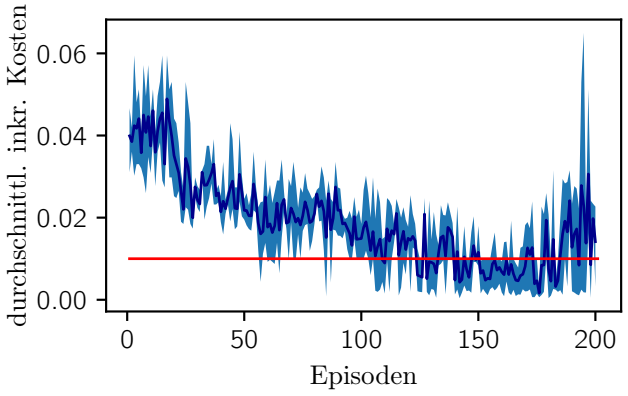

Abbildung 6. Lernkurven von je 200 Episoden (blau = Mittelwert über alle Lernvorgänge, rot = darüber: Episode wurde abgebrochen)

Die in Abbildung 6 dargestellten Lernkurven zeigen, dass der Zustand $\mathcal{X}^-$ bereits nach rund 50 Episoden nicht erreicht wird. Nach rund 150 Episoden überführt der Agent den Roboter zuverlässig in den Zielzustand $\mathcal{X}^+$. Der Algorithmus kann jedoch auch divergieren, wie die Werte am Ende der Lernkurve verdeutlichen.

4.3. ERGEBNISSE

Tabelle 1 zeigt die Ergebnisse der einzelnen Simulationsexperimente. Die beiden nicht aufgeführten Anfangswerte $\dot{x}_0$ und $\dot{\theta}_0$ wurden jeweils null gewählt.

Tabelle 1. Gesamtkosten der einzelnen Zustandsrückführungen für verschiedene Auslenkungen

#	x_0/m	θ_0/rad	J_{LQR}	J_{optZRF}	J_{NFQ}
1	0	$\frac{\pi}{9}$	1,047	**1,046**	106,243
2	0	$\frac{2\pi}{9}$	7,329	**6,931**	79,720
3	0,3	0	**0,578**	0,579	39,310
4	0,7	0	**3,122**	**3,122**	44.684
5	0,5	$\frac{\pi}{6}$	7,644	**7,499**	70,778
6	0,5	$\frac{\pi}{4}$	30,916	**17,384**	-
7	0	$\frac{\pi}{3}$	-	**44,886**	-

Die niedrigsten Gesamtkosten sind jeweils hervorgehoben. Abbildung 7 zeigt den Verlauf von Zustand und Steuergröße für LQR, optZRF und der mittels NFQ entworfenen Zustandsrückführung für Auslenkung 2. In Abbildung 8 sind diese Verläufe für den LQR und die optZRF für Auslenkung 6 abgebildet. Insgesamt zeigt sich, dass alle entworfenen Zustandsrückführungen sind das Beispielsystem für Auslenkungen 1 bis 5 stabilisieren können. Hinsichtlich der Gesamtkosten werden mit der optZRF mit der Ausnahme von kleinen x-Auslenkungen die besten Ergebnisse erreicht. Für Auslenkungen nahe der betrachteten Ruhelage sind die Abweichungen zwischen LQR und optZRF sehr gering. Insbesondere für große Auslenkungen des Pendelwinkels wird mit der optZRF eine wesentlich höhere Regelgüte erreicht. Dies zeigt sich auch qualitativ in den Verläufen in Abbildung 8. Bei Auslenkung 7 kann nur noch die optZRF das System stabilisieren.

Die Suboptimalität des LQR war aufgrund der Linearisierung des Systems beim Entwurf zu erwarten. Die hohe Regelgüte der optZRF in Gestalt eines MLP ist mit Blick auf die begrenzte Anzahl der berechneten optimalen Trajektorien interessant. Hervorzuheben ist auch die Stabilisierung bei Auslenkung 7, welche nicht in den Trainingsdaten enthalten war. Bei derartig großen Auslenkungen stellt sich jedoch die Frage nach der physikalischen Realisierbarkeit.

Die hohen Kosten des NFQ-Algorithmus (s. Tabelle 1) lassen sich neben dem unterschiedlichen Entwurfs-Kostenfunktional auf die diskrete Steuergröße zurückführen. Zudem hat darf nicht außer Acht gelassen werden, dass bei der NFQ das Modell des Roboters nicht in den Entwurf mit einfließt.

5. FAZIT UND AUSBLICK

Der Implementierungsaufwand für NFQ und die optZRF ist überschaubar und wird durch vorhandene Bibliotheken wie *Keras* und *TensorFlow* bedeutend vereinfacht. Bei der optZRF überwiegt der Aufwand zur Berechnung optimaler Steuertrajektorien sehr deutlich. Bei der NFQ hat sich gezeigt, dass die Wahl von geeigneten Aktionen für den Agenten nicht trivial ist, wodurch der Algorithmus auch schwer vergleichbar mit den beiden anderen vorgestellten Methoden ist. Ein weitere Entwursfreiheit sind die in Tabelle 5 aufgeführten Hyperparameter, deren Bestimmung eine große Schwierigkeit darstellte, wobei

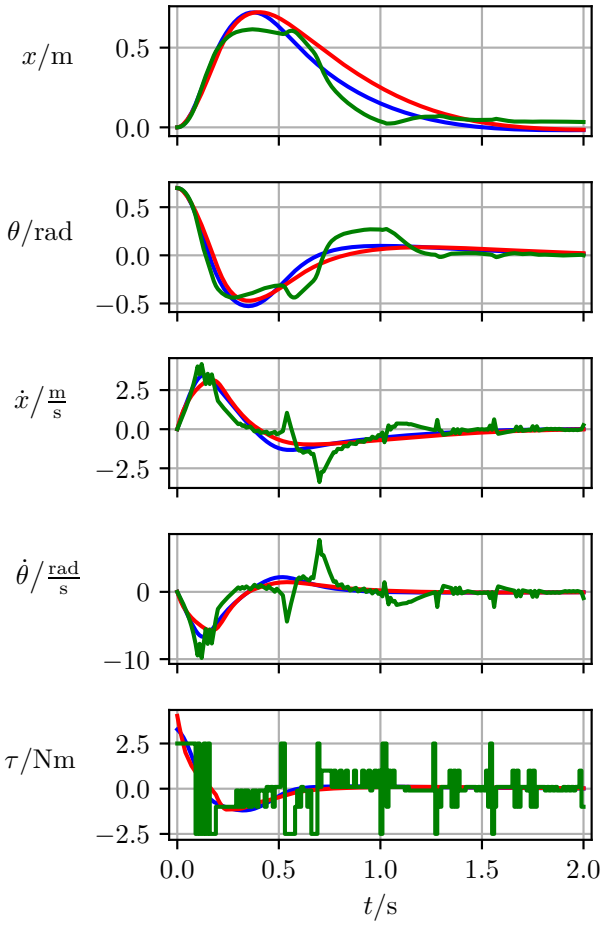

Abbildung 7. Verlauf von Zustandskomponenten und Steuergröße (LQR, NFQ, optZRF) für Auslenkung 2 (vgl. Tabelle 1)

jedoch [24] eine große Hilfe war. Bei der Wahl der Anfangswerte $\mathcal{X}_0$ ist besonders darauf zu achten, dass die Erreichbarkeit des Zielzustandes $\mathcal{X}_+$ gewährleistet ist.

Die Erwartung an NFQ, dass nur durch Anpassung des Simulationsmodells und der Kostenfunktion weitere Beispielsysteme ohne aufwendige Anpassungen des Algorithmus geregelt werden können hat sich nicht bestätigt.

Ein Algorithmus, der die Probleme von NFQ zu lösen vermag ist Deep-Deterministic Policy-Gradient (DDPG) [12], bei dem für die Strategie ein zweites MLP verwendet wird. Es stellt in gewisser Weise eine Erweiterung von NFQ dar und wurde von den Autoren an verschiedenen Systemen, wie bspw. dem Wagen-Pendel erprobt. Ein weiterer Vorteil im Vergleich zu NFQ ist, dass die Hyperparameter des nicht an das System angepasst werden müssen.

Die vorgestellte Methode zum Entwurf einer optimalen Zustandsrückführung hat sich für das betrachtete System als leistungsfähiger Ansatz erwiesen. Der Mehraufwand bei der Implementierung im Vergleich zum LQR-Entwurf ist erheblich und lohnt sich somit nur, falls ein System für größere Auslenkungen stabilisiert werden soll.

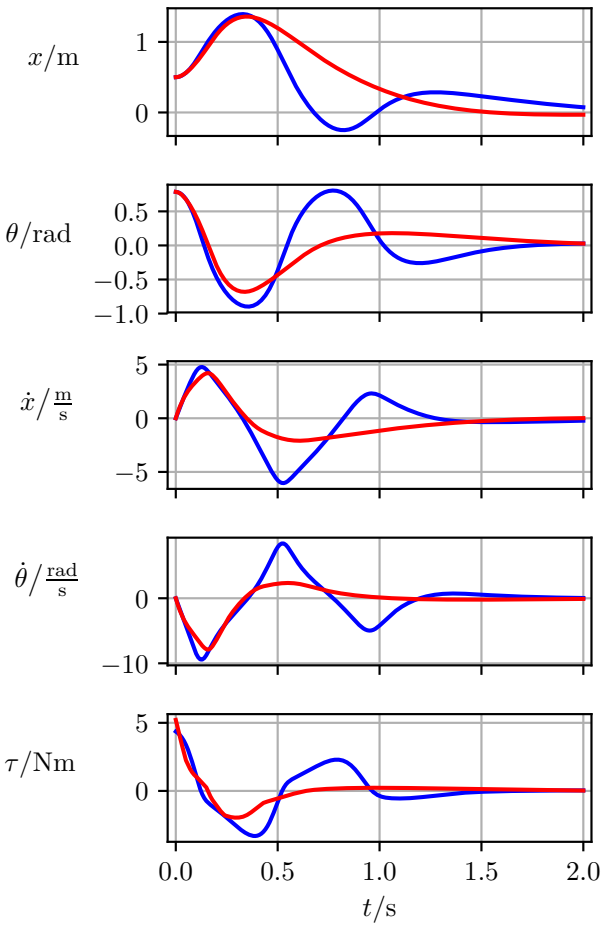

Abbildung 8. Verlauf von Zustandskomponenten und Steuergröße (LQR , optZRF) für Auslenkung 6 (vgl. Tabelle 1)

Inwiefern sich mit der Methode optimale Trajektorien für komplexere Systeme hinsichtlich Zustands- und Steuergrößendimension sowie Nichtlinearitäten berechnen lassen, ist offen. Dies gilt sowohl aus theoretischer Sicht mit Blick auf die Optimalitätsbedingungen als auch aus praktischer Sicht mit Blick auf den genutzten Algorithmus zur Lösung der ZPRWA.

Bei der Berechnung der optZRF ließe sich zusätzlich auch eine Steuergrößenbeschränkung explizit berücksichtigen. Die dazu anzupassende Optimalitätsbedingung ist als PONTRJAGINs Maximumprinzip bekannt [2, S. 110ff.]. Weiterhin könnte untersucht werden, ob sich die Methode zum Entwurf eines MLP-Vorfilters erweitern lässt, um einen Zustandsregler zu implementieren.

Die vorgestellten Ansätze können als Beispiel dafür dienen, dass sich Methoden des maschinellen Lernens zur Anwendung in der Regelungs- und Steuerungstheorie prinzipiell eignen und weitere Entwicklungen in diesem Gebiet nicht außer Acht gelassen werden sollten.

LITERATURVERZEICHNIS

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton u. a., „Mastering the game of go without human knowledge“, *Nature*, Jg. 550, Nr. 7676, S. 354, 2017. DOI: 10.1038/nature24270.

[2] K. Graichen, *Methoden der Optimierung und optimalen Steuerung, Wintersemester 2017/2018*, Institut für Mess–, Regel– und Mikrotechnik, Uni Ulm, 2017. Adresse: https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.110/Downloads/Vorlesung/Optimierung/Skript/Skript_MOOS_WS1718.pdf.

[3] R. S. Sutton und A. B. Barto, *Reinforcement Learning: An Introduction*. The MIT Press Cambridge, Massachusetts, 1998.

[4] R. S. Sutton, A. G. Barto und R. J. Williams, „Reinforcement learning is direct adaptive optimal control“, *IEEE Control Systems*, Jg. 12, Nr. 2, S. 19–22, 1992. DOI: 10.1109/37.126844.

[5] C. Szepesvári, „Algorithms for reinforcement learning“, *Synthesis lectures on artificial intelligence and machine learning*, Jg. 4, Nr. 1, S. 1–103, 2010. DOI: 10.2200/S00268ED1V01Y201005AIM009.

[6] C. G. Atkeson und J. C. Santamaria, „A comparison of direct and model-based reinforcement learning“, in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, IEEE, Bd. 4, 1997, S. 3557–3564. DOI: 10.1109/ROBOT.1997.606886.

[7] S. Gu, T. Lillicrap, I. Sutskever und S. Levine, „Continuous deep q-learning with model-based acceleration“, in *International Conference on Machine Learning*, 2016, S. 2829–2838. arXiv: 1603.00748.

[8] M. Deisenroth und C. E. Rasmussen, „PILCO: A model-based and data-efficient approach to policy search“, in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, S. 465–472. Adresse: http://mlg.eng.cam.ac.uk/pub/pdf/DeiRas11.pdf.

[9] S. Kamthe und M. P. Deisenroth, „Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control“, 2017. arXiv: 1706.06491.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski u. a., „Human-level control through deep reinforcement learning“, *Nature*, Jg. 518, Nr. 7540, S. 529, 2015. DOI: 10.1038/nature14236.

[11] M. Riedmiller, „Neural Fitted Q-Iteration - First Experiences with a Data Efficient Reinforcement Learning Method“, in *Machine Learning: ECML 2005. ECML 2005. Lecture Notes in Computer Science*, Bd. 3720, 2005. DOI: 10.1007/11564096_32.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver und D. Wierstra, „Continuous control with deep reinforcement learning“, 2015. arXiv: 1509.02971.

[13] S. Gu, E. Holly, T. Lillicrap und S. Levine, „Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates“, in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, S. 3389–3396. arXiv: 1610.00633.

[14] C. J. Watkins und P. Dayan, „Q-learning“, *Machine learning*, Jg. 8, Nr. 3-4, S. 279–292, 1992. DOI: 10.1007/BF00992698.

[15] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior und K. Kavukcuoglu, „Wavenet: A generative model for raw audio“, *arXiv preprint arXiv:1609.03499*, 2016.

[16] J. Kober, J. A. Bagnell und J. Peters, „Reinforcement learning in robotics: A survey“, *The International Journal of Robotics Research*, Jg. 32, Nr. 11, S. 1238–1274, 2013. DOI: 10.1177/0278364913495721.

[17] D. Kriesel, *Ein kleiner Überblick über Neuronale Netze*. 2007. Adresse: http://www.dkriesel.com/_media/science/neuronalenetze-de-zeta2-2col-dkrieselcom.pdf.

[18] K. Hornik, M. Stinchcombe und H. White, „Multilayer feedforward networks are universal approximators“, *Neural networks*, Jg. 2, Nr. 5, S. 359–366, 1989. DOI: 10.1016/0893-6080(89)90020-8.

[19] S. Sonoda und N. Murata, „Neural network with unbounded activation functions is universal approximator“, *Applied and Computational Harmonic Analysis*, Jg. 43, Nr. 2, S. 233–268, 2017. DOI: 10.1016/j.acha.2015.12.005.

[20] S. Ruder, „An overview of gradient descent optimization algorithms“, 2016. arXiv: 1609.04747.

[21] F. Chollet u. a., *Keras*, https://keras.io, 2015.

[22] *TensorFlow*, 2015. Adresse: https://www.tensorflow.org/.

[23] E. Jones, T. Oliphant, P. Peterson u. a., *SciPy: Open source scientific tools for Python*, 2001–. Adresse: http://www.scipy.org/.

[24] M. Riedmiller, „10 steps and some tricks to set up neural reinforcement controllers“, in *Neural networks: tricks of the trade*, Springer, 2012, S. 735–757.

[25] ——, „A Framework for RL Learning Controllers“, 2008. Adresse: http://people.csail.mit.edu/russt/iros2008_workshop_talks/Riedmiller.pdf.

[26] R. Hafner, *Dateneffiziente selbstlernende neuronale Regler*, 2009. Adresse: https://d-nb.info/998931659/34.

ANHANG

BEISPIELSYSTEM

Bestimmung des nichtlinearen Zustandsraummodells. Generalisierte Koordinaten:

$$\mathbf{q} = (q_1, q_2)^{\mathrm{T}} = (x, \theta)^{\mathrm{T}}$$

Kinetische Energie T und potentielle Energie U:

$$T(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\left(2m_{\mathrm{w}} + 2\frac{J_{\mathrm{w}}}{r^2} + m_{\mathrm{p}}\right)\dot{x}^2 + m_{\mathrm{p}} c \cos(\theta)\dot{\theta}\dot{x} + \frac{1}{2}\left(J_{\mathrm{p}} + m_{\mathrm{p}} c^2\right)\dot{\theta}^2$$

$$U(\mathbf{q}, \dot{\mathbf{q}}) = m_{\mathrm{p}} g c \cos(\theta)$$

LAGRANGE-Funktion:

$$L(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - U(\mathbf{q}, \dot{\mathbf{q}})$$

Bewegungsgleichungen:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{q}_j}\right) - \frac{\partial L(\mathbf{q}, \dot{\mathbf{q}})}{\partial q_j} = F_j,\ j = 1, 2$$

$$F_1 = 2\frac{\tau}{r},\ F_2 = 0$$

Bewegungsgleichungen in Matrixform:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{K}(\mathbf{q}) = \mathbf{F}$$

$$\mathbf{M}(\mathbf{q}) = \begin{pmatrix} 2m_{\mathrm{w}} + 2\frac{J_{\mathrm{w}}}{r^2} + m_{\mathrm{p}} & m_{\mathrm{p}} c \cos(\theta) \\ m_{\mathrm{p}} c \cos(\theta) & J_{\mathrm{p}} + m_{\mathrm{p}} c^2 \end{pmatrix},\ \mathbf{F} = \begin{pmatrix} 2\frac{\tau}{r} \\ 0 \end{pmatrix}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{pmatrix} -m_{\mathrm{p}} c \sin(\theta)\dot{\theta}^2 \\ 0 \end{pmatrix},\ \mathbf{K}(\mathbf{q}) = \begin{pmatrix} 0 \\ -m_{\mathrm{p}} g c \sin(\theta) \end{pmatrix}$$

Nichtlineares Zustandsraummodell (explizit):

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u) = \begin{pmatrix} x_3 \\ x_4 \\ \frac{r}{2}\frac{c^2 m_{\mathrm{p}}^2 g r \sin(2x_2) - 2(J_{\mathrm{p}} + c^2 m_{\mathrm{p}})(c m_{\mathrm{p}} r x_4^2 \sin(x_2) + 2u)}{c^2 m_{\mathrm{p}}^2 r^2 \cos^2(x_2) - (J_{\mathrm{p}} + c^2 m_{\mathrm{p}})(2J_{\mathrm{w}} + r^2(m_{\mathrm{p}} + 2m_{\mathrm{w}}))} \\ c m_{\mathrm{p}} \frac{r(c m_{\mathrm{p}} r x_4^2 \sin(x_2) + 2u)\cos(x_2) - g(2J_{\mathrm{w}} + r^2(m_{\mathrm{p}} + 2m_{\mathrm{w}}))\sin(x_2)}{c^2 m_{\mathrm{p}}^2 r^2 \cos^2(x_2) - (J_{\mathrm{p}} + c^2 m_{\mathrm{p}})(2J_{\mathrm{w}} + r^2(m_{\mathrm{p}} + 2m_{\mathrm{w}}))} \end{pmatrix}$$

Modellparameter.

$$m_{\mathrm{w}} = 0{,}5\ \mathrm{kg},\ m_{\mathrm{p}} = 3{,}0\ \mathrm{kg},\ g = 9{,}81\frac{\mathrm{m}}{\mathrm{s}^2},$$
$$b = 0{,}2\ \mathrm{m},\ c = 0{,}3\ \mathrm{m},\ r = 0{,}05\ \mathrm{m},$$
$$J_{\mathrm{w}} = \frac{1}{2} m_{\mathrm{w}} r^2,\ J_{\mathrm{p}} = \frac{1}{12} m_{\mathrm{p}}\left(b^2 + c^2\right)$$

IMPLEMENTIERUNGSDETAILS

Wichtungsmatrizen.

$$\mathbf{Q} = \mathrm{diag}\left(8{,}7\ \tfrac{1}{\mathrm{m}}\quad 8{,}7\ \tfrac{1}{\mathrm{rad}}\quad 10^{-5}\ \tfrac{\mathrm{s}}{\mathrm{m}}\quad 10^{-5}\ \tfrac{\mathrm{s}}{\mathrm{rad}}\right),$$
$$R = 4{,}8\ \frac{1}{\mathrm{Nm}},\ \mathbf{S} = \mathrm{diag}\left(8{,}7\ \tfrac{1}{\mathrm{m}}\quad 8{,}7\ \tfrac{1}{\mathrm{rad}}\quad 4\ \tfrac{\mathrm{s}}{\mathrm{m}}\quad 3\ \tfrac{\mathrm{s}}{\mathrm{rad}}\right)$$

Parameter der Methode optZRF nach 3.1.
Trainingsdatensatz:

$$N = 100,\ \mathbf{x}_{0\ \max} = \left(1\ \mathrm{m},\ \frac{\pi}{4}\ \mathrm{rad},\ 1\ \frac{\mathrm{m}}{\mathrm{s}},\ 1\frac{\mathrm{rad}}{\mathrm{s}}\right)^{\mathrm{T}},$$

$$\mathbf{x}_{0\ \min} = -\mathbf{x}_{0\ \max},\ t_{\mathrm{f}} = 3\,\mathrm{s},\ \text{Schrittweite: } 10\ \mathrm{ms}$$

MLP: Eingabedimension: 4; Ausgabedimension: 1; 20 Perzeptronen in der Eingabeschicht mit Rectified-Linear-Unit (ReLU) als Aktivierungsfunktion, keine verborgene Schicht, ein Perzeptron in der linearen Ausgabeschicht, insgesamt 121 Gewichte (inklusive Bias), *Keras*-Fehlerfunktion: *mean_ squared_ error*, *Keras*-Optimierer: *nadam*

MLP-Training: Zyklen: 30, Stapel-Größe: 32, Validierungsdatensatz: 10 % des Trainingsdatensatzes, Approximationsfehler nach dem Training: $< 10^{-3}$

Parameter der Methode NFQ nach 3.3.
MLP: Eingabedimension: $\dim(\mathbf{x}) + dim(\mathbf{u}) = 5$; Ausgabedimension: $dim(\hat{Q}) = 1$; die zwei verborgenen Schichten haben jeweils 20 Perzeptronen mit Tangens Hyperbolicus als Aktivierungsfunktion, bei der Ausgabeschicht wurde die Sigmoid-Funktion verwendet. Insgesamt 561 Gewichte, *TensorFlow*-Fehlerfunktion: *mean_ squared_ error*, *TensorFlow*-Optimierer: *tf.train.AdamOptimizer()*

MLP-Training: Zyklen: 300, Stapel-Größe: ≤ 15.000, Schrittweite: $0,001$, Initialisierung der Gewichte: Gleichverteilung $[-0,5,\ 0,5]$

Tabelle 2. Parameter des Lernvorgangs

Episoden n	200
Dauer T	6 s
Δt	10 ms
γ	0,99
ϵ	0,1
ϵ_{exp}	0,98
τ	$\big(-2,5 \quad -1,0 \quad -0,1 \quad 0,1 \quad 1,0 \quad 2,5\big)\mathrm{Nm}$
$\mathcal{X}^0$	$\big([-0,5, \quad 0,5]\mathrm{m} \quad [-0,3, \quad 0,3]\mathrm{rad} \quad * \quad *\big)$
$\mathcal{X}^+$	$\big([-0,05, \quad 0,05]\mathrm{m} \quad [-0,03, \quad 0,03]\mathrm{rad} \quad * \quad *\big)$
$\mathcal{X}^-$	$\big([-1, \quad 1]\mathrm{m} \quad [-\frac{\pi}{2}, \quad \frac{\pi}{2}]\mathrm{rad} \quad * \quad *\big)$