LEARNING A SYSTEM MODEL FOR LEARNING

eingereichte
SEMINARARBEIT
von

Thomas Praxenthaler

geb. am 28.07.1987
wohnhaft in:
Gartenweg 3
83413 Fridolfing
Tel.: 0151 56955859

Dipl. Ing. Jochen Guck

geb. am 05.05.1985
wohnhaft in:
Carl-Orff-Weg 16
82110 Germering
Tel.: 0172 6568486

Lehrstuhl für
STEUERUNGS- UND REGELUNGSTECHNIK
Technische Universität München
Univ.-Prof. Dr.-Ing../Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

Betreuer/-in:	Prof. Dr. sc. Nat. J. Conradt
Beginn:	27.10.2010
Abgabe:	24.01.2011

TECHNISCHE UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK
ORDINARIUS: UNIV.-PROF. DR.-ING./UNIV. TOKIO MARTIN BUSS
EXTRAORDINARIA: UNIV.-PROF. DR.-ING. SANDRA HIRCHE

18.10.2010

A D V A N C E D S E M I N A R
for
Jochen Guck, Mat.-Nr. 18756188
Thomas Praxenthaler, Mat.-Nr. 18637362

Learning a System Model for Learning

Problem description:

In many robotic systems engineers carefully hand-design control algorithms to perform particular actions with high precission. Often, however, it is tideous or even impossible to estimate the exact mechanical and electric properties of the plant, such that control laws can only be tuned to approximations of such systems. An alternative approach to develop control laws is based on machine learning techniques, such as *reinforcement learning* (RL). In such settings, the RL setup will find parameters for a succesful controller by trial and error. Unfortunately, even simple control problems, such as balancing a pendulum, require a large number of trials for learning, typically in the range of several 100.000 iterations. This is impossible to perform on a real robot, so that state-of-the-art RL systems typically learn in and get applied to simulated robots. Those simulated robotic systems only resemble the real robot to a certain extend, which turns it very difficult if not impossible to apply a learned controller from simulation into a controller for a real robot.
In this "Hauptseminar" students shall investigate in existing approaches to learn controllers on real-world robots in connection with simulators. Starting from a few literature pointers and web addresses of research groups working in this area the students shall explore further literature and give a comprehensive review of existing systems that learn a controller in simulation, possibly update the simulator to better reflect a real robotic system, and later transfer such learned controllers to real-world robot applications.

Bibliography:

[1] Gloye, A., Gktekin, C., Egorova, A., Tenchio, O., Rojas, R. Learning to Drive and Simulate Autonomous Mobile Robots. RoboCup-2004: Robot Soccer World Cup VIII Springer-Verlag, 2004. http://www.idsia.ch/~alexander/2004/3/learning5.ps
[2] Abbeel, P., Coates, A., and Ng, A.Y. Autonomous Helicopter Aerobatics through Apprenticeship Learning. International Journal of Robotics Research (IJRR), 2010. http://ai.stanford.edu/~ang/papers/ijrr10-HelicopterAerobatics.pdf
[3] Conradt, J., Berner, R., Cook, M., and Delbruck, T. An Embedded AER Dynamic Vision Sensor for Low-Latency Pole Balancing. IEEE Workshop on Embedded Computer Vision (ECV09), Kyoto, Japan. http://www.lsr.ei.tum.de/fileadmin/publications/Conradt/ECV2009-JConradt.pdf

Supervisor: Jörg Conradt

(J. Conradt)
Professor

Zusammenfassung

Derzeit befassen sich viele Forschungsarbeiten damit, wie Roboter eigenständig lernen können Problemstellungen zu lösen. Ein vielversprechender Ansatz hierzu ist das Reinforcement Learning. Da diese Lernstrategie am echten Roboter nur schwer umsetzbar ist, wird in der Regel in der Simulation gelernt. Erst dann wird das Gelernte auf den realen Roboter übertragen.

Diese Arbeit gibt einen Überblick über verschiedene Möglichkeiten zur datenbasierten Modellierung von Systemen, welche zur Simulation verwendet werden und diskutiert deren Eigenschaften. Dabei lässt sich grundsätzlich eine Unterteilung in parametrisierte und nicht parametrisierte Modelle vornehmen. Zu den parametrisierten Modellen zählen die Anpassung von systembeschreibenden Differentialgleichungen, sowie die Verwendung von Simulationsumgebungen. Die nicht parametrisierten Modelle unterteilen sich in Neuronale Netze und Regressionsalgorithmen.

Als Ergebnis ist festzustellen, dass bei der Wahl des Modellierungsverfahrens die genauen Rahmenbedingungen der Problemstellung ausschlaggebend sind, da alle Verfahren ihre Berechtigung haben.

Abstract

Recently several papers have been released, dealing with robots learning solutions to problems on their own. One promising approach is Reinforcement Learning. As this method can hardly be applied to the real robotic system, the controller is usually learned in simulation. After successful learning in simulation, the control laws are applied to the real robot.

This paper gives a review of several possibilities of data-based modeling of systems that can be used for simulation. Furthermore it discusses the several properties. The different methods can basically be divided into parametric and non-parametric modeling. Parametric modeling includes the approximation of the system's differential equations as well as the use of simulation environments. Non-parametric modeling is either based on artificial neural networks or on regression algorithms.

As a result, it can be concluded that the specific framework is the crucial factor for the choice of the proper method for modeling the system. All of the methods have their eligibility, depending on the specific circumstances.

Inhaltsverzeichnis

1 Einleitung

Die Anforderungen an Roboter sind in den letzten Jahren immer umfangreicher und komplexer geworden. Hierzu gehört unter anderem die Fähigkeit für immer längere Zeit autonom zu handeln. Um den steigenden Anforderungen gerecht zu werden, ist es nahezu unumgänglich, dass man von einer problemspezifischen, manuellen Programmierung abkommt. Vielmehr muss der Roboter dahingehend programmiert werden, dass er in der Lage ist, Problemstellungen durch Lernen eigenständig zu lösen. Ein vielversprechender Ansatz im Forschungsgebiet des Maschinellen Lernens (machine learning) ist das Reinforcement Learning (RL). Mit dieser Methode lassen sich Steuerstrategien (control policies) lernen und Parameter optimieren.

Ein Reinforcement Learning Algorithmus funktioniert in seiner einfachsten Form gemäße Abbildung 1.1. Dabei führt das RL-System eine Aktion aus, welche eine Auswirkung in der realen Welt hat. Abhängig davon, inwiefern die Auswirkung einem definierten Ziel dient, wird dieser Aktion im jeweiligen Zustand eine Belohnung bzw. eine Bestrafung zugeordnet. Über Sensoren müssen dabei die relevanten Zustände des Aktuators in der Welt erfasst werden.

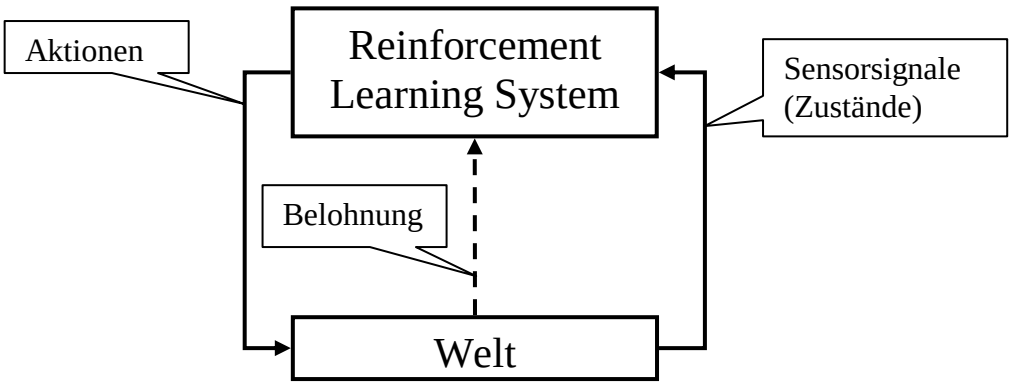

Abbildung 1.1: Reinforcement Learning

Durch geeignete Wahl der Belohnungs-/Bestrafungsfunktion ist es möglich, den einzelnen Zuständen spezifische Werte zuzuordnen, und basierend auf diesen Werten eine Strategie zu entwickeln, mit welchen Aktionen das gewünschte Ziel schnellst- und bestmöglich erreicht wird. Die jeweiligen Bewertungsfunktionen müssen problemspezifisch angepasst werden. Für das finden einer Strategie existieren viele verschiedene Algorithmen, die jedoch nicht Bestandteil dieser Arbeit sind. Ein Hauptproblem all dieser Algorithmen ist die Vielzahl an Iterationen, die benötigt werden, um eine Lösung zu finden. Aus diesem Grund ist der Einsatz von RL-Strategien in Verbindung mit einem realen System nur bedingt möglich, da es sehr kostspielig und zeitaufwendig ist, es während der Lernphase zu betreuen. Ein Beispiel dieser

Problematik wäre ein humanoider Roboter der beim Laufenlernen bei jeder „schlechten“ Aktion umfällt.

Um dieses Problem zu umgehen wird häufig ein Simulationsmodell des Roboters manuell erstellt. Mit diesem kann anschließend durch RL die beste Regelstrategie ermittelt werden. Diese Methode hat allerdings den Nachteil, dass es aufwendig ist, das Simulationsmodel zu erstellen, da viele Parameter, wenn überhaupt, nur mit großem Aufwand messtechnisch zu erfassen sind.

Um den aufwendigen Prozess der manuellen Modellbildung mit all seinen Schwierigkeiten und Ungenauigkeiten abzukürzen und zu verbessern, wurden verschiedene Ansätze entwickelt. Diese basieren auf realen Messwerten, mit welchen es möglich ist Simulationsmodelle automatisch zu erstellen, beziehungsweise Simulationsparameter automatisch zu finden.

In dieser Arbeit werden die parametrisierten Modellierungsansätze basierend auf System-DGLs bzw. höheren Modellierungssprachen vorgestellt. Ferner werden nichtparameterisierte Modellierungsansätze wie Neuronale Netze und verschiedene Regressionsmethoden aufgeführt. Dabei werden die jeweiligen Eigenschaften näher betrachtet und diskutiert.

2 Parametrisierte Modelle

Jedes physikalische System kann theoretisch mit Differentialgleichungen (DGLs) beschrieben werden. Die Bildung eines Modells kann somit erfolgen, indem sämtliche DGLs eines Systems aufgestellt, und deren Parameter bestimmt werden. Wie in der Aufgabenstellung beschrieben, ist die manuelle Bestimmung der Parameter jedoch kein triviales Problem. Aus diesem Grund liegt der Gedanke nahe, die Parameter aus Messdaten automatisch bestimmen zu lassen. In diesem Kapitel werden zwei Verfahren zur parametrisierten Modellierung vorgestellt. Der grundlegende Aufbau (vgl. [1]) einer Parameteroptimierung ist in Abbildung 2.1 dargestellt.

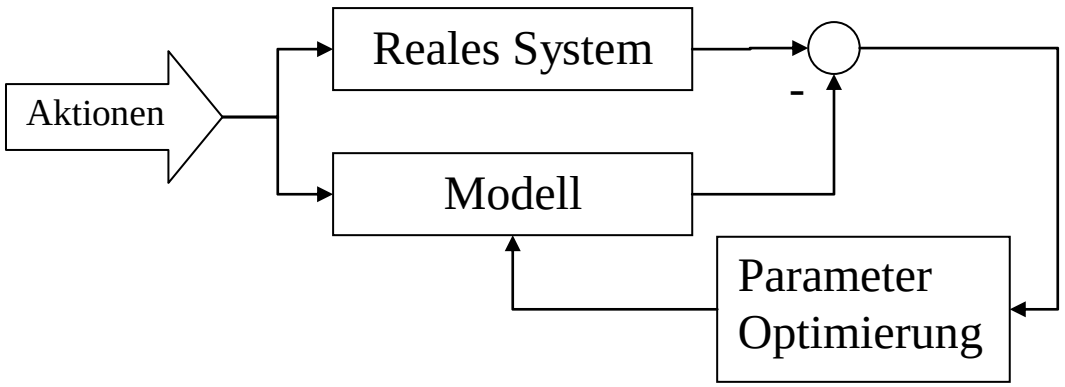

Abbildung 2.1: Aufbau Parameteroptimierung

Als erster Schritt ist ein Modell zu erzeugen, welches die Funktionsweise des realen Systems beschreibt. Dieses Modell muss eine Schnittstelle bieten, mit welcher Einfluss auf die internen Modellparameter genommen werden kann. Anschließend sind Messungen am realen System durchzuführen. Diese sollten den gesamten Arbeitsbereich abdecken um eine möglichst genaue Bestimmung der Modellparameter zu ermöglichen. Das Simulationsmodel ist mit den für die Messung verwendeten Steuersignalen zu beaufschlagen. Aus dem Ergebnis von Simulation und realem Versuch wird ein Fehler berechnet. Dieser wird mit Hilfe eines Optimierungsalgorithmus dazu verwendet, um die Modellparameter zu optimieren.

2.1 Modellierung mit Differentialgleichungen

Speziell bei einfachen Systemen, deren Dynamiken weitestgehend bekannt sind, können parametrisierte Differentialgleichungen aufgestellt werden, um das System zu modellieren. Die Differentialgleichungen müssen dabei alle Zusammenhänge zwischen

den Steuereingängen und den aktuellen Zuständen wiedergeben. Es ist offensichtlich, dass das manuelle Aufstellen dieser Differentialgleichungen, beispielsweise für Roboter mit mehreren Gliedern und Freiheitsgraden, sehr schnell unübersichtlich und zunehmend komplexer wird.

Zur Beschreibung abgeschlossener Systeme, wie beispielsweise einem Helikopter (vgl. [2]) kann jedoch auf bekannte, auch nicht lineare Systemmodellierungen zurückgegriffen werden. Dabei gilt der logische Zusammenhang, dass durch die Wahl eines komplexeren Modells die wahren Systemeigenschaften besser angenähert werden können.

Zur Anpassung des Modells an das tatsächliche System werden Datensätze dieses Systems benötigt, welche den Zusammenhang von Steuergrößen und Zuständen wiedergeben. Mit Hilfe dieser Daten können die bislang unbekannten Parameter der System-DGLs bestimmt werden. In [2] erfolgt diese Parameterbestimmung einerseits durch lineare Regression mittels Least-Squares Verfahren und andererseits durch Optimierung der Simulationsgenauigkeit des resultierenden Modells. Bei letzterem wird die Genauigkeit der Vorhersage nach einer simulierten Zeitspanne von mehreren Sekunden als Optimierungskriterium verwendet. Die Anpassung der Parameter erfolgt durch gradientenbasierte, numerische Optimierung. Der Vergleich der beiden Methoden zeigt, dass beim Least-Squares-Modell die Koeffizienten in der Regel kleiner sind als beim anderen Modell. Letzteres führte in einigen Experimenten zu besseren Ergebnissen für die Steuerung des Helikopters.

Gemäß [2] reicht diese Art der Modellierung aus, um autonome Flüge im Bereich der Grundposition auszuführen. Für aggressivere Manöver ist dieses Systemmodel jedoch nicht ausreichend genau, so dass offline gelernte Kommandoabfolgen zum Reproduzieren von Trajektorien in der Realität nicht zum gewünschten Ziel führen. Grund dafür ist, dass die angenommene 12-dimensionale Festkörper-Zustandsbeschreibung die tatsächlichen Zustände des Helikopters nur bedingt beschreibt. Aus diesem Grund wird in dieser Arbeit die Modellierung verfeinert. Dies geschieht, indem die Parameter des obigen Modells jeweils lokal für einen begränzten, lokalen Trajektorienabschnitt bestimmt werden. Für die Abschnitte der verschiedenen Manöver wird somit eine Vielzahl von Modellen gelernt, die ineinander übergehen. Auf diese Weise gelingt es, nach geeignetem Lernen der Kommandoabfolgen, dass der Helikopter schwierige, aggressive Manöver fliegen kann.

Als zusammenfassende Aussage zu diesem Kapitel kann festgehalten werden, dass Modellierung mittels DGLs möglich ist, wobei berücksichtigt werden muss, dass oft nicht alle Zusammenhänge modelliert werden. Anwendungsspezifisch kann deshalb eine Anpassung der Modellierung erforderlich sein.

2.2 Simulatormodelle

Ein weiterer wichtiger Modellierungsansatz ist die Verwendung von numerischer Simulationssoftware. In der Arbeit von Hultgren und Jonasson [1] wurde eine komplexe Toolchain zur Erstellung und Optimierung von Simulationsmodellen vorgestellt.

Zum Erstellen der Modelle wurde das Programm Dymola verwendet, welches auf der objektorientierten Modellierungssprache Modelika basiert. Diese Sprache wurde speziell für physikalische Modellbeschreibung entwickelt. Durch die Grafische Oberfläche von Dymola, welche zu einer erhöhten Übersichtlichkeit führt, können komplexe Modelle auf komfortable Art erstellt werden.

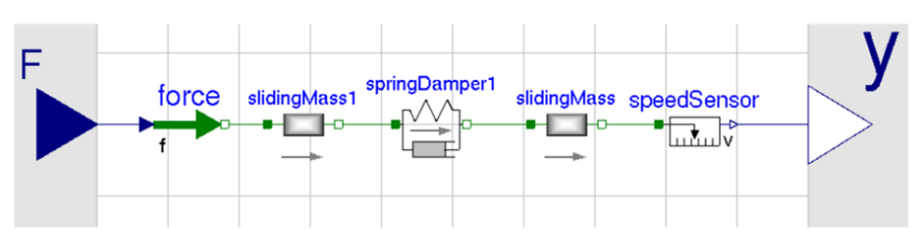

Abbildung 2.2: Servomodell [1]

In Abbildung 2.2 ist das Dymola-Beispielmodell eines Servoantriebs zu sehen. Dieses Modell ist nicht blockorientiert, wie z.B. Simulik, da beim Übersetzen die zugrundeliegenden Differentialgleichungen ineinander eingesetzt werden. Aufgrund der umfangreichen Programmbibliotheken kann bei der Entwicklung von neuen Modellen auf eine Vielzahl von bereits existierenden Teilmodellen zurückgegriffen werden. Ein Beispiel hierfür ist die Vehicle Dynamic Library (VDL) welche Modelle für die Simulation von Automobilen zur Verfügung stellt. Des Weiteren existiert eine Bibliothek für dreidimensionale Kinematik, welche zur Simulation von Manipulatoren geeignet ist.

Zur Anpassung der Parameter des Modells wurde die mathematische Beschreibungssprache AMPL verwendet. Diese Sprache ist dazu geeignet komplexe mathematische Optimierungsprobleme zu lösen. Es wird allerdings ein externer Solver benötigt. In diesem Fall wurde IPOPT verwendet. Mit AMPL können die diskreten Signale des Simulationsmodells unter Verwendung von Splines in kontinuierliche Signale umgewandelt werden, was eine anschließende, nicht lineare Parameteroptimierung ermöglicht. In „Automatic Calibration of Vehicle Models“ [1] (Seite 36, Figure 4.8) ist das Beispiel einer erfolgreichen Parameteranpassung des in Abbildung 2.2 abgebildeten Servomodells dargestellt. Zu erwähnen ist hierbei, dass der Datenaustausch zwischen Dymola und AMPL eine nicht triviale Aufgabe ist, da eine Umformatierung an den Schnittstellen notwendig ist.

Eine andere Modellierungsmöglichkeit bieten Physik Engines wie z.B. Bullet oder Newton (vgl. [3]). Diese Engines wurden für den Einsatz in 3D Spielen entwickelt und sollen dort für realistische Bewegungen der Objekte sorgen. Die Adaption dieser Simulationsumgebung an einen Reinforcement Lerningalgorithmus gestaltete sich

allerdings schwierig, da die Schnittstellen nicht für diese Art von Problemstellung ausgelegt wurden.

2.3 Gegenüberstellung

Beiden Methoden der nicht parametrisierten Modellierung sind bestimmte Eigenschaften gemeinsam. Die wohl wichtigste ist dabei, dass beide auf der datenbasierten, automatisierten Bestimmung von Parametern eines DGL-Systems beruhen, welches zuvor, entweder manuell, oder durch zu Hilfenahme eines Simulationstools, erstellt werden muss. Diese Tatsache hat den Vorteil, dass aus den bestimmten Parametern jeweils Rückschlüsse auf die Eigenschaften des Systems getroffen werden können. Auf diese Weise können beispielsweise physikalische Größen abgeschätzt werden, deren messtechnische Erfassung oft nur sehr umständlich möglich wäre. Grundsätzlich kann auch festgehalten werden, dass für beide Verfahren eine Vielzahl an Optimierungsalgorithmen verwendet werden kann, um die parametrisierten Differentialgleichungen an die Messdaten anzunähern. Ein Vorteil dieser Verfahren ist auch, dass bereits mit wenigen Messdaten eine erste Abschätzung der Systemparameter und somit des Modells erfolgen kann, da dieses aufgrund der dynamischen Zusammenhänge auf den gesamten Arbeitsbereich generalisierbar ist. Selbstverständlich nimmt die Genauigkeit des Modells mit der Anzahl an Messdaten und der Abdeckung des Arbeitsraums stetig zu.

Der größte Unterschied der Verfahren liegt darin, dass bei der Verwendung von System-DGLs sämtliche dynamische Zusammenhänge manuell in Form von Differentialgleichungen modelliert werden müssen. Bei der Verwendung eines Simulationstools, wie z.B. Dymola, können komplexe Systeme aus einzelnen, oft bereits in Bibliotheken hinterlegten Teilkomponenten zusammengesetzt werden. Die Generierung des zusammengesetzten DGL-Systems erfolgt dann automatisch durch das Simulationstool.

Zu den Vorteilen der Verwendung von Simulationstools gehört somit, dass weniger Wissen über die globalen dynamischen Zusammenhänge erforderlich ist. Sehr hilfreich ist unter anderem auch, dass eine große Anzahl an umfangreichen Bibliotheken existiert. Diese Tatsache erleichtert und beschleunigt den Vorgang der Modellbildung enorm, da in erster Linie nur problemspezifische Komponenten neu modelliert und integriert werden müssen. Eine schöne Eigenschaft von Simulationstools ist zudem, dass diese oft eine visuell anschauliche Darstellung des Problems liefern. So kann, beispielsweise bei der Simulation eines Roboterarmes, die Bewegung jedes einzelnen Gliedes im Gesamtverbund dargestellt werden. Auf diese Weise kann schnell festgestellt werden ob zumindest alle offensichtlichen Komponenten modelliert wurden und deren Verschaltung korrekt durchgeführt wurde. Da Simulationstools auf Modellierungssprachen basieren, ist es zudem leicht möglich festzustellen, ob neue Dynamiken richtig in die Modellierung eingebunden wurden, weil eventuelle Fehler beim Kompilieren des Modells ausgegeben werden.

Um diese Vorteile nutzen zu können, müssen jedoch auch einige Kompromisse eingegangen werden. So ist zu erwähnen, dass ein Simulationstool wie beispielsweise Dymola mitsamt seiner Bibliotheken einen hohen Anschaffungspreis hat. Zusätzlich ist für jede Modellierung ein relativ hoher Overhead notwendig, welcher auf das komplexe Framework des Programms zurückzuführen ist. Die komplexen Zusammenhänge sind mitunter auch ein Grund für die hohe Rechenintensität, welche verhindert, dass ein derartiges System beispielsweise auf einem Microcontroller laufen könnte.

Zur Verwendung von herkömmlichen Physics-Engines als Simulationstool müssen viele Anpassungen getroffen werden, da diese in erster Linie für die virtuelle Darstellung von Objekten, beispielsweise in PC-Spielen, verwendet werden und sich somit viele Eigenschaften der realen Systeme nur schwer modellieren lassen. Aus diesem Grund sind viele Physics-Engines nur bedingt zur Simulation in diesem Sinn geeignet.

Als Vorteil des manuellen Aufstellens von DGLs kann erwähnt werden, dass die Eigenschaften des Systems besser verstanden werden, und somit die Bedeutung der einzelnen Parameter klarer ist. Bei einer groben Vorstellung in welchem Bereich sich diese Parameter befinden sollten, könnte somit im besten Fall ein falsch bestimmter Parameter erkannt und daraus resultierend die Modellierung oder der Datensatz hinterfragt werden. Ferner ist das manuell aufgestellte Modell in der Regel um ein vielfaches schlanker als das automatisch generierte. Das hat wiederum den Vorteil, dass ein derartiges Modell eventuell auch online angepasst werden könnte.

Ein gemeinsames Problem der beiden Verfahren besteht in der Genauigkeit der Simulation. Da die Zusammenhänge durch Gleichungen beschrieben sind, kann die Simulation lediglich diese Abbildungsfunktion ausführen. Nichtmodellierte Zusammenhänge können in der Regel nicht simuliert werden. Um diese Ungenauigkeiten ins Modell zu integrieren, müssen komplexere DGLs als Grundlage verwendet werden, was wiederum einen erhöhten Modellierungs- und Anpassungsaufwand mit sich bringt. Aus diesem Grund muss diesbezüglich ein Kompromiss als Lösung gefunden werden. Bei der Verwendung von DGLs kann, beispielsweise durch Lokalisierung der Problemstellung, wie in Kapitel 2.1 beschrieben, das Problem der nicht modellierten Zusammenhänge teilweise umgangen werden.

3 Nichtparametrisierte Modelle

Viele Systeme der Robotik lassen sich nicht einfach analytisch beschreiben, da deren nicht lineare Dynamik oft sehr komplex oder teilweise unbekannt ist. Der Entwurf eines DGL Modells (vgl. Kapitel 2) basierend auf den bekannten Gegebenheiten, wie Maßen und Gewichten der einzelnen Bestandteile, und den physikalischen Zusammenhängen ist deshalb oft sehr aufwendig und liefert häufig nur bedingt gute Ergebnisse. Grund dafür sind unter anderem Nichtlinearitäten welche bei der Modellierung nicht bekannt sind. Diese nicht linearen Zusammenhänge stammen beispielsweise aus Hydraulikleitungen, Getrieben und komplexen Reibungskräften. Diese unbekannten Faktoren können dazu führen, dass ein analytisches Modell teilweise starke Abweichungen vom realen System zeigt.

Für die nicht parametrisierte Modellbildung, welche das gesamte System mitsamt allen Nichtlinearitäten berücksichtigt, ist nur eine bedingte Kenntnis der physikalischen Eigenschaften notwendig. Vielmehr basiert sie auf aufgezeichneten Daten des realen Systems. Die Aufzeichnung kann beispielsweise bei einer Expertendemonstration erfolgen. Die gesammelten Daten beschreiben dabei den Zusammenhang zwischen Eingangskommandos (Aktionen) und den daraus resultierenden Zuständen. Durch nicht parametrisierte Modelle kann die Abbildung von Aktionsvektor A auf Zustandsvektor X rekonstruiert werden. Einen Simulator erhält man, indem jeweils der geschätzte Zustand zeitverzögert zurückgeführt und zusammen mit der externen Aktion erneut dem Modell zugeführt wird. Das Schema hierzu ist in Abbildung 3.1 dargestellt.

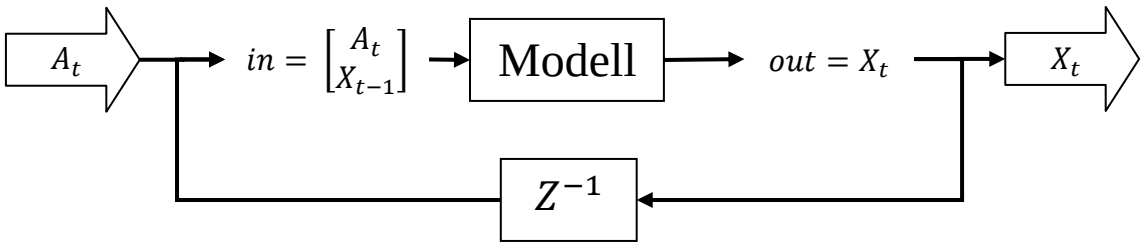

Abbildung 3.1: Simulator Schema

3.1 Neuronales Netz

Eine Möglichkeit der Funktionsapproximation bieten Neuronale Netze (NN). Im Buch von Hertz, Krogh und Palmer [4] wird der grundlegende Aufbau dargestellt. Die NN sind eine aus der Natur kopierte Methode, welche zur Funktionsannäherung verwendet

werden kann. Dabei werden Erkenntnisse über den Aufbau und die Funktionsweise des Gehirns verwendet.

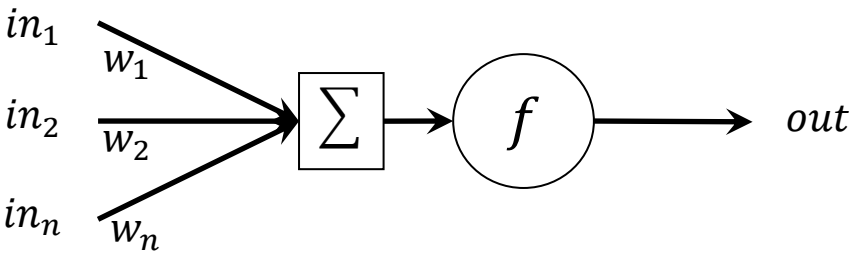

Abbildung 3.2: Künstliches Neuron

In Abbildung 3.2 ist ein künstliches Neuron dargestellt. Dieses ist der Grundbaustein eines jeden Neuronalen Netzes. Ein Neuron besitzt n Eingänge in_i deren Werte mit den dazugehörigen Gewichten w_i multipliziert werden. Die Ergebnisse werden aufsummiert und über eine Aktivierungsfunktion f auf den Ausgang out gegeben. Formel (3.1) stellt diesen Zusammenhang da.

$$out = f\left(\sum_{i=1}^{n} in_i \, w_i\right) \tag{3.1}$$

In Abbildung 3.3 ist eine Auswahl an möglichen Aktivierungsfunktionen dargestellt. Durch spezifische Auswahl der Aktivierungsfunktion lassen sich Neuronen mit unterschiedlichem Verhalten erzeugen. Außerdem ist die Lernstrategie der Funktion entsprechend zu wählen.

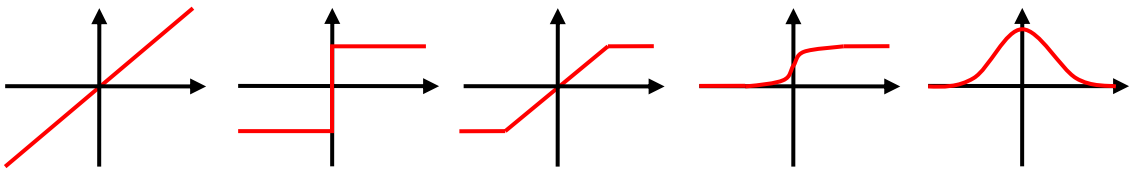

Abbildung 3.3: Aktivierungsfunktionen

Ein Neuron zu trainieren bedeutet die Gewichte so anzupassen, dass bestimmte Eingangswerte vorgegebene Ausgangswerte erzeugen. Mit einem einzelnen Neuron ist eine Annäherung an komplexe Funktionen nicht möglich.

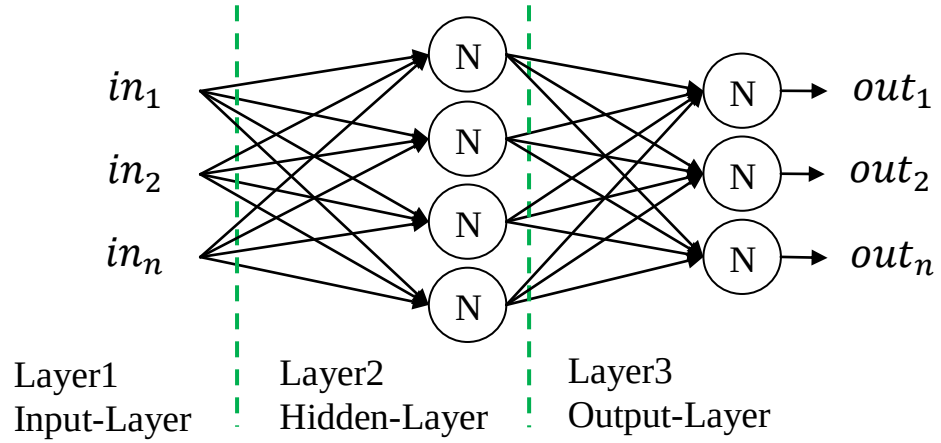

Abbildung 3.4: Neuronales Netz

Aus diesem Grund werden Netzwerke von Neuronen verwendet, welche komplizierte Funktionen abbilden können. Abbildung 3.4 stellt ein typisches Netzwerk dar. Da keine internen Rückkopplungen vorhanden sind, spricht man auch von einem Feedforward Netz. Dieses besteht standardmäßig aus drei Schichten, auch Layer genannt, wobei auch mehr Schichten denkbar sind. Der Input-Layer enthält die Eingänge des Netzes. Die Werte werden in dieser Schicht allerdings nicht verändert. Die Neuronen im Hidden-Layer haben den größten Anteil an der Datenverarbeitung. Je mehr Neuronen dort verwendet werden, desto komplexere Funktionen können approximiert werden. Außerdem ist es wichtig, in dieser Schicht eine nicht lineare Aktivierungsfunktion zu verwenden, also z.B. Sigmoidal oder Gauß. Wird eine Gauß-Funktion verwendet, so spricht man auch von einem Radial Basis Network [5]. Im Output-Layer werden die gewünschten Ausgänge des Netzes bereitgestellt. Die Anzahl der Neuronen entspricht der Anzahl der Ausgänge. Die Aktivierungsfunktion dieser Neuronen ist meistens die lineare.

Diese Form von Netzen verwendet überwachtes Lernen um eine Anpassung der Gewichte vorzunehmen. Bei dieser Strategie ist es notwendig, einen Satz an Trainingsdaten zu erzeugen. Diese Daten enthalten die Eingangswerte und die dazugehörigen Ausgangswerte. Um das Netz zu trainieren werden im ersten Schritt die Gewichte zufällig initialisiert. Mit Hilfe dieses Netzes werden aus den Eingangswerten der Trainingsdaten Ausgangswerte berechnet. Aus den berechneten Ausgangswerten und den Ausgangswerten der Trainingsdaten lässt sich ein Fehler berechnen. Anhand dieses Fehlers ist es möglich die einzelnen Gewichte des Netzwerkes schrittweise so einzustellen, dass er minimiert wird. Das bedeutet, dass die grundlegende Funktion der Daten nachgebildet wird. Ein wichtiges Verfahren, um ein Netzwerk zu trainieren, ist das Backpropagation-Verfahren, hierzu sei auf die Standardliteratur zu diesem Thema verwiesen.

Wird ein Radial Basis Network verwendet, so müssen zusätzlich zu den Gewichten die Parameter der Gauß-Funktionen trainiert werden. Dabei repräsentiert jedes Neuron im Hidden-Layer eine Gaußkurve. Für ein einfaches Mapping von X auf Y könnten die nötigen Gaußfunktionen wie in Abbildung 3.5 dargestellt platziert sein. Durch die Funktionen G1, G2 und G3 wird allerdings die Funktion $f(\mathrm{x})$ nicht genau abgebildet. Um diesen Umstand zu ändern, könnten noch weitere Neuronen, also Gaußkurven,

eingeführt werden. Außerdem sind die Breiten der einzelnen Kurven noch anzupassen. Die linearen Neuronen im Ausgangs-Layer fassen die Einzelwerte der Gaußneuronen wieder zu einem Ausgangswert zusammen.

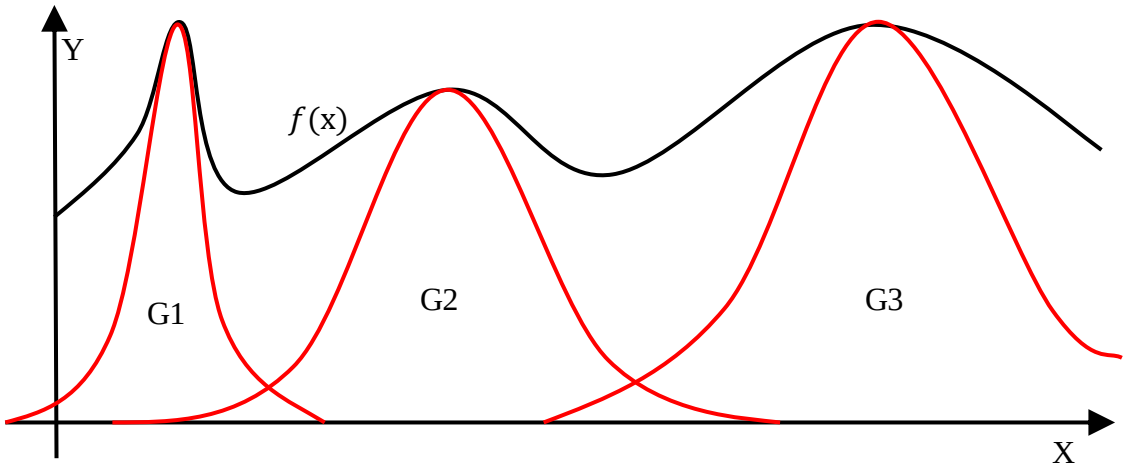

Abbildung 3.5: Funktionsweise RBN

In der Arbeit „RBF NN Based Marine Diesel Engine Generator Modeling“ [6] wurde ein Radial Basis Network verwendet um das Verhalten eines Schiffs-Generators zu simulieren. Es wird ein kleines Netzwerk mit 99 Radial Basis Neuronen verwendet. Dieses wird zur Echtzeit Simulation auf einem Digitalen Signal Prozessor (DSP) mit 30MIPS bei einer Periodendauer von 158 Mikrosekunden eingesetzt. Dieser Prozessor berechnet zusätzlich das Verhalten des Dieselmotors. Aus den Signalverläufen ([6], Seite 2748, Fig. 5-8) kann die Leistungsfähigkeit dieses Simulators abgeschätzt werden. Es ist zu erkennen, dass kurze Ereignisse nicht mitsimuliert wurden. Außerdem mussten, um eine ausreichende Genauigkeit in der Simulation zu erreichen, zwei Netze verwendet werden. Das erste für das Verhalten des Generators im Normalbetrieb und das Zweite für das Verhalten im Fehlerfall.

In „Model-based Reinforcement Learning with Model Error and Its Application“ [7] wird die Anwendung eines Neuronalen Netzes als Modell dokumentiert. Das Netz wird klein gehalten, indem verschiedene Bewegungszustände als separates Netz modelliert werden.

3.2 Regressionsalgorithmen

Zur nicht parametrisierten Approximation von Werten werden neben Neuronalen Netzen auch verschiedene Arten von Regressionsalgorithmen verwendet. Obwohl die verschiedenen Algorithmen als gemeinsames Ziel eine möglichst genaue Abbildung der Eingänge auf die daraus resultierenden Zustände besitzen, so unterscheiden sich diese teilweise stark in ihrer Vorgehensweise, aber auch in Genauigkeit und Rechenintensität. Speziell letzteres ist von großer Bedeutung, wenn das Modell nicht nur einmalig erstellt,

sondern während des Betriebes immer weiter verbessert, beziehungsweise angepasst werden soll.

In den folgenden Kapiteln folgt eine Darstellung einiger der zurzeit wohl bekanntesten, und am häufigsten verwendeten, nichtpararametrisierten Algorithmen:

- Gaussian Process Regression (GPR)
- Locally Weighted Projection Regression (LWPR)
- Local Gaussian Processes (LGP)

Dabei wird auf Gemeinsamkeiten und Unterschiede zwischen den Algorithmen eingegangen und eine Beurteilung der Algorithmen abgegeben.

An dieser Stelle sei der Vollständigkeit halber auch Support Vector Regression (SVR) genannt, welche in Rechenaufwand und Geschwindigkeit mit GPR vergleichbar ist. Auf die Darstellung wird jedoch in dieser Arbeit verzichtet, um den Umfang der Arbeit in Grenzen zu halten. Zu SVR sei deshalb auf die Literatur verwiesen: [8]

In diesem Zusammenhang soll erwähnt werden, dass die Modellierung von Systemen durch nicht parametrisierte Regression eine Form des Supervised Learning darstellt. Die Ergebnisse werden in der Regel bei der so genannten Computed Torque Control verwendet. Dabei werden durch Regressionen die inversen Systemmodelle erstellt. Mithilfe dieser Modelle können, basierend auf den gewünschten Positionen, Geschwindigkeiten und Beschleunigungen, die notwendigen Steuerkommandos erzeugt werden. Zur Regelung reicht dann, bei einem genauen Modell, oft bereits ein einfacher PD-Controller (vgl. [9]). Auf diese Weise können jedoch auch Modelle gelernt und als eine Art Simulator verwendet werden, um mittels Reinforcement Learning einen Controller zu optimieren, welcher spezielle Aufgaben lernen soll.

3.2.1 Gaussian Process Regression (GPR)

Ein Gaußscher Prozess ist vollkommen definiert durch seine Mittelwertfunktion und seine Kovarianzfunktion. Im Gegensatz zur Gaußschen Verteilung, bei der die Verteilung über einen Vektor erfolgt, erfolgt sie beim Gaußschen Prozess über Funktionen. Durch einsetzen von diskreten Eingangswerten, jeweils in Mittelwert- und Kovarianzfunktion, kann der Prozess als Verteilung dargestellt werden.

Um zu verhindern, dass mit einer unendlichen Anzahl von Funktionen gearbeitet werden muss, wird nur eine endliche Anzahl an Eingangswerten betrachtet, für die jeweils die Mittelwertfunktion und die Kovarianz berechnet werden. Die geeignete Auswahl der Eingangswerte ist dabei verantwortlich für eine ausreichende Repräsentation des abzudeckenden Bereichs. Dabei muss immer bedacht werden, dass mit der Anzahl an Werten der Rechenaufwand stark ansteigt, da jeder zusätzliche Wert eine zusätzliche Zeile und eine zusätzliche Spalte in der Kovarianzmatrix bedeutet.

Mittelwert- und Kovarianzfunktion sind beim Lernen von Systemen in der Regel nicht vorab bekannt. Aus diesem Grund müssen diese Funktionen, abhängig von den jeweiligen Daten, beim Trainieren des Gaußschen Prozesses ausgewählt werden. Bei vagen Informationen über das System wird in der Regel ein hierarchisches Vorwissen

verwendet, bei welchem Mittelwert- und Kovarianzfunktion durch Hyperparameter parametrisiert sind. Diese Parametrisierung erlaubt es, bei vagem Wissen über das System, dieses zu spezifizieren, ohne es jedoch zu stark einzuschränken. Es kann beispielsweise angenommen werden, dass sich das System ähnlich wie ein Polynom zweiter Ordnung verhält ohne sich festlegen zu müssen, welches Polynom genau verwendet wird und wie ähnlich es sich verhält.

Die Hyperparameter werden dann im Zuge des Trainierens bestimmt, indem die Wahrscheinlichkeit der Daten in Abhängigkeit von den Hyperparametern betrachtet wird. Diese Wahrscheinlichkeit wird als *log marginal likelihood* bezeichnet. Mit Hilfe der partiellen Ableitung können die Hyperparameter bestimmt werden, für welche die *log marginal likelihood* optimiert wird. Auf diese Weise erhält man die Mittelwert- und Kovarianzfunktion aus der jeweiligen Funkionsschar, mit welchen sich die Daten am besten beschreiben lassen.

Nachdem ein Gaußscher Prozess definiert wurde, welcher die Abbildung von Eingangs- auf Zustandsgröße modelliert, können für beliebige Werte aus dem Raum der Eingangsgrößen die dazugehörigen Zustandsgrößen vorhergesagt werden. Der Gaußsche Prozess ist dabei nicht unveränderbar, sondern kann durch neue Trainingswerte fortlaufend erweitert werden, was jedoch auf Kosten der Rechendauer geht.

Für tiefergehende Informationen zu GPR sei auf die Arbeit von C.E. Rasmussen [10] verwiesen, welche als Grundlage für den hier gegebenen Überblick diente. In dieser Arbeit wird unter anderem auch auf die Behandlung von überlagertem Rauschen auf den Trainingsdaten eingegangen.

Eine Anwendung von GPR zur Modellbildung, bei der das gelernte Modell als Simulator für einen Reinforcement-Learning-Algorithmus dient, ist in [11] gegeben. Dabei wird ein angepasster GPR Algorithmus verwendet, um die Steuerung eines Luftschiffs am Simulator mittels Reinforcement Learning zu lernen, bevor ein Test in der Realität erfolgt.

Des Weiteren wird in [12] das einfache Beispiel eines Autos im Tal gegeben, welches einen Berg hinauf fahren will. Dabei werden aufgezeichnete Simulationsdaten mittels GPR angenähert und zur Vorhersage verwendet. Der RL-Algorithmus verwendet dabei einen Gaußschen Prozess zur Darstellung der Wertfunktionen.

3.2.2 Locally Weighted Projection Regression (LWPR)

Dieses Verfahren ist charakterisiert durch die Anwendung von nicht parametrisierter Regression auf eine Vielzahl von lokalen, einfachen Modellen. Jedes lokale Modell führt dazu eine geringe Anzahl von eindimensionalen Regressionen in ausgewählte Richtungen (Principle Components) des Eingangsraumes, gemäß der Idee des partiellen Least Squares Verfahrens, durch. Auf diese Weise kann der Rechenaufwand für das gesamte System reduziert werden. Da eine Vielzahl von lokalen Modellen verwendet wird, können jeweils sehr einfache Modelle verwendet werden, wie beispielsweise Polynome niedriger Ordnung (lineares Modell).

Ein wichtiger Punkt bei der Betrachtung der lokalen Modelle des LWPR ist die Dimensionsreduktion. Da der Raum der Eingangsgrößen durch lokale Modelle abgedeckt werden soll, nimmt die Anzahl der benötigten lokalen Modelle exponentiell mit den Dimensionen zu. Um diesem Problem entgegen zu wirken wird eine Projektionsregression (PR) verwendet, welche die Regression höherdimensionaler Eingänge durch eine Überlagerung eindimensionaler Regressionen von wenigen, ausgewählten Projektionen aus dem Eingangsraum ersetzt. Diese Reduktion von Dimensionen ist möglich, da die verschiedenen Dimensionen häufig sehr stark redundant, beziehungsweise teilweise irrelevant sind (vgl. [13]). Für die lokale Zerlegung in die so genannten Principle Components existieren mehrere Verfahren, welche in [13] vorgestellt und bewertet werden.

Beim Lernen der Daten müssen verschiedene Parameter bestimmt werden:

- Die Anzahl der lokalen Modelle
- Die Parameter der Hyperebene in jedem lokalen Modell
- Der Gültigkeitsbereich des jeweiligen Modells (Gaußscher Kernel)

Abbildung 3.6 gibt einen schematischen Überblick über die Funktionsweise des Algorithmus.

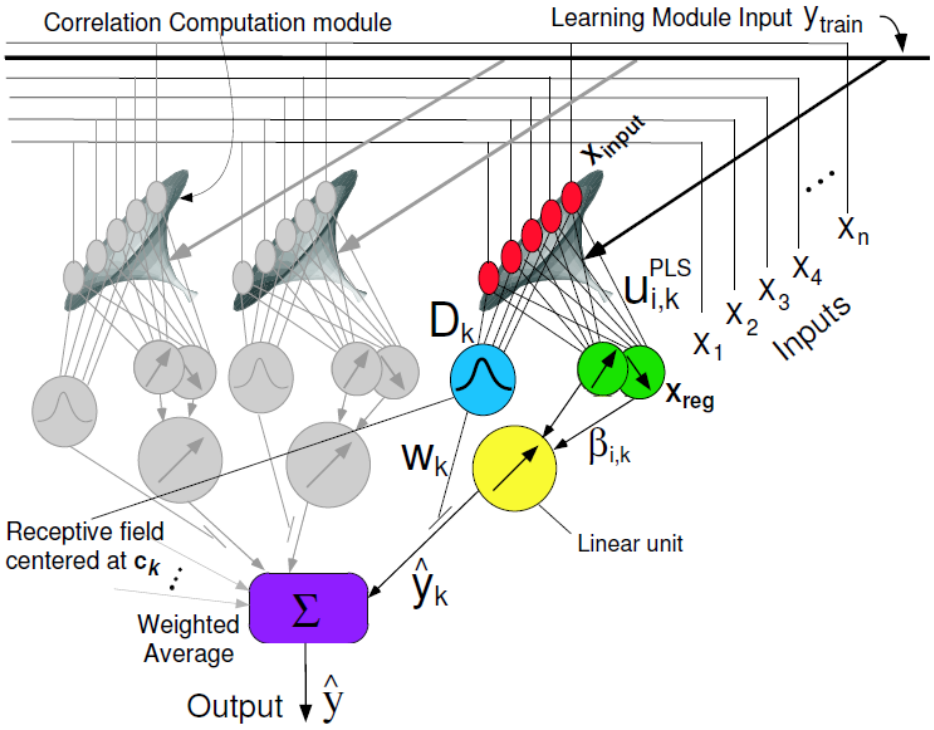

Abbildung 3.6: Informationsverarbeitungseinheit des LWPR [13]

Der hier gegebene Überblick zum Thema LWPR basiert auf der Arbeit von S. Vijayakumar et al. [13]. Bei tiefergehendem Informationsbedarf zu LWPR sei auf diese Arbeit verwiesen. Als wichtigste Eigenschaften des Algorithmus werden folgende Aspekte genannt:

- LWPR lernt schnell mit Lernmethoden zweiter Ordnung, welche auf inkrementellem Training basieren.
- LWPR verwendet beim Lernen eine so genannte Cross-Validation, weshalb kein Abspeichern der Trainingsdaten erforderlich ist.
- Die Anpassung des Gewichtungskernel erfolgt nur unter Berücksichtigung der lokalen Daten.
- Der Rechenaufwand verhält sich linear zur Anzahl der Eingänge.
- LWPR kann eine große Anzahl von Eingängen, auch redundante, verarbeiten.

Eine Anwendung von LWPR zur Modellbildung, welche zum Lernen eines Controllers mittels Reinforcement Learning verwendet wird, ist in [14] gegeben. Dabei werden, auf Basis eines aus Daten einer Expertendemonstration gewonnenen LWPR Modells, Flugmanöver eines Helikopters gelernt und autonom ausgeführt. In der Arbeit wird gezeigt, dass auf diese Weise sehr schwierige Manöver gelernt und geflogen werden können.

3.2.3 Local Gaussian Process Regression (LGP)

Die Einführung zu LGP orientiert sich an den Arbeiten von D.Nguyen-Tuong, J.Peters und M. Seeger [15][16][17]. Diese Arbeiten behandeln das Thema mit Hilfe der gleichen Beispiele, aber mit unterschiedlichen Schwerpunkten.

LGP kombiniert die Eigenschaften der GPR mit der lokalen Aufteilung der LWPR. Diese Kombination führt zu einem Algorithmus, welcher in Genauigkeit und Rechendauer jeweils zwischen GPR und LWPR liegt. Ein wichtiger Aspekt ist hier, ähnlich wie bei der LWPR, dass der Rechenaufwand soweit reduziert werden kann, dass Online-Lernen möglich ist. Auf die Vor- und Nachteile, sowie die Anwendungsgebiete der verschiedenen Algorithmen im Allgemeinen, wird in den folgenden Kapiteln näher eingegangen.

Die lange Rechenzeit des GPR ist auf eine Matrixinversion zurückzuführen, welche mit $O(n^3)$ von der Anzahl der Trainingswerte abhängt. Die Reduktion des Rechenaufwands wird bei LGP dadurch erreicht, dass der Eingangsraum mit Hilfe eines Zuweisungsnetzwerks (gating network) in kleinere Unterräume unterteilt wird. In diesem Unterraum wird wiederum ein Gaußscher Prozess, z.B. ein lokales gaußsches Modell, trainiert. Die Rechendauer hängt dabei stark von der Anzahl der Modelle ab. Der Rechenaufwand des Algorithmus, welcher in [15][16][17] verwendet wird, ist in etwa $O(n^3/M)$ beim Training und $O(n^2d)$ zum Anpassen der Parameter des Zuweisungsnetzwerks, wobei M die Anzahl der Modelle, und d die Dimension des Eingangsvektors ist.

Das Zuweisungsnetzwerk führt dabei eine Art Clustering mit Gaußschen Kernels durch, wobei die Eingangsdaten möglichst sinnvoll auf die Modelle verteilt werden. Die Vorhersage für einen Eingangswert erfolgt dann durch gewichtete Mittelung der Vorhersagen der lokalen Modelle in der Nachbarschaft.

Der Algorithmus lässt sich somit in zwei Stufen unterteilen. In der ersten Stufe werden die Eingangsdaten den entsprechenden lokalen Modellen zugewiesen und die Modelle gelernt. In der zweiten Stufe können dann Vorhersagen für beliebige Punkte aus dem Eingangsraum getroffen werden.

Das Lernen der Clusterzuordnung und der Generation von neuen Clustern erfolgt dabei durch kontinuierliches Hinzufügen der einzelnen Daten, ohne dass alle Trainingsdaten im Vorhinein bekannt sind. Das bedeutet, dass die Weiten und Zentren der Kernels regelmäßig ein Update benötigen, wenn neue Werte ins Modell eingefügt werden. Die Generation von neuen Clustern erfolgt, wenn die Abstände eines neuen Wertes zu den Zentren der existierenden Kernels oberhalb eines gewissen Grenzwerts liegen. Der neue Punkt bildet dann das Zentrum des neuen Kernels. Die Komplexität der einzelnen Modelle kann dabei beschränkt werden, indem eine Obergrenze für die Anzahl an Werten in einem Modell festgelegt wird. Wird diese Grenze überschritten, muss ein alter Datenpunkt gelöscht werden, um einen neuen hinzuzufügen.

Dieser Algorithmus liefert eine sehr flexible Möglichkeit der Modellbildung, da bei zunehmender Komplexität der Trajektorien die Anzahl der Modelle beliebig erweitert werden kann. Speziell wenn sich im Betrieb gewisse Verläufe von Trajektorien ändern, können diese online im Modell angepasst werden, da durch die Beschränkung der lokalen Modellkomplexität der Rechenaufwand begrenzt ist.

Die Anwendung dieses Algorithmus erfolgte bislang in erster Linie in Zusammenhang mit Computed Torque Control ([15][16][17]). Seine Darstellung in dieser Arbeit lässt sich jedoch dadurch rechtfertigen, dass er, wie im folgenden Kapitel beschrieben, genauer ist wie LWPR und schneller als GPR, für welche jeweils Beispielanwendungen in Zusammenhang mit Reinforcement Learning existieren.

3.2.4 Gegenüberstellung der Algorithmen

Allen hier angeführten Algorithmen ist gemeinsam, dass deren Anwendung nur bei nicht linearen Systemen sinnvoll ist. Für den Fall, dass die Abbildungsfunktion bekannter weise lineare Zusammenhänge aufweist, ist die Verwendung gewöhnlicher, linearer Regressionsmethoden vorzuziehen, da ansonsten unnötige Rechenresourcen verbraucht werden. Ferner ist es notwendig, dass eine Vielzahl von Daten, bestehend aus Paaren von Eingangs- und Zustandsgrößen zur Verfügung stehen, welche vom realen System aufgezeichnet wurden und zum Training zur Verfügung stehen. Zusätzlich müssen bei jedem Algorithmus problemspezifisch externe Anpassungen getroffen werden. Bei GPR sind das beispielsweise die Grundform der Mittelwert- und Kovarianzfunktion, bei LGP und LWPR die Parameter zur Partitionierung der Eingangswerte. Die Anzahl dieser offenen Parameter ist dabei gemäß [18] bei LWPR größer als bei GPR.

Die rechenaufwendigen Algorithmen, wie GPR, oder auch die nicht näher behandelte SVR, eignen sich in erster Linie fürs Batch- Lernen. Das bedeutet, dass sämtliche Trainingsdaten vor dem Lernen vorhanden sein müssen und alle Daten auf einmal gelernt werden. Grund dafür ist, dass die Berechnung aufgrund der notwendigen Rechenkapazitäten und -dauer nur offline erfolgen kann. Diese Methoden der

Regression sind deshalb vorrangig für Systeme geeignet, deren Verhalten zeitunabhängig immer gleich ist.

Viele Systeme verändern jedoch während des Betriebes ihre Eigenschaften, beispielsweise aufgrund von thermischen Effekten. Für diese Systeme werden Modelle benötigt, mit der Möglichkeit der Onlineanpassung.

Bei LWPR und LGP, den Algorithmen mit lokalen Regressionen, können Datenpunkte einzeln hinzugefügt werden ohne jeweils die gesamte Abbildungsfunktion neu berechnen zu müssen. Vielmehr haben neue Datenpunkte nur Einfluss auf die lokalen Modelle und können deshalb mit begrenztem Rechenaufwand in die Abbildungsfunktion des Modells integriert werden. Das hat den Vorteil, dass diese Algorithmen beim Eintreffen neuer Daten schnell und unkompliziert angepasst werden können. Über einstellbare Parameter können im Bedarfsfall alte Daten entfernt werden, so dass inzwischen vermeintlich falsche Daten keinen Beitrag mehr leisten.

Der Vorteil von GPR und SVR gegenüber lokalen Modellen liegt in der Genauigkeit mit der die Datensätze abgebildet werden, da alle verfügbaren Werte ein globales Modell bilden. Speziell gegenüber LWPR hat GPR den weiteren Vorteil, dass bereits eine geringere Anzahl an Daten ausreicht, um ein brauchbares Modell zu erstellen. Bei LWPR sind mehr Daten notwendig, um die Anzahl der Projektionsrichtungen der lokalen Modelle zu bestimmen. In [9] werden speziell LWPR und GPR gegenübergestellt. Dort wird festgestellt, dass GPR leichter auf Lernprobleme angewandt werden kann.

In [17] werden die Algorithmen GPR, SVR, LWPR und LGP an verschiedenen Systemen getestet. Zusätzlich wird hier der Approximatiosfehler der linearen Regression eines analytischen Modells bestimmt. Bei einem System mit sieben Freitheitsgraden (Roboterarm), zeigt sich dabei gemäß [17] deutlich, dass der Approximationsfehler des analytischen Modells um ein vielfaches höher ist, als der bei den nicht parametrisierten Algorithmen. Die Genauigkeit von GPR und SVR ist dabei vergleichbar. Etwas ungenauer, aber besser als LWPR, verhält sich in den dargestellten Daten LGP.

Joint Nr.	nMSE [%]				
	Lin. Regr.	LWPR	ν-SVR	GPR	Local GPR
1	19.2	2.4	1.2	0.6	1.1
2	182.5	2.6	0.5	0.3	0.7
3	19.8	1.5	0.4	0.2	0.5
4	65.6	0.9	0.3	0.2	0.7
5	295.5	5.3	1.4	1.5	2.8
6	27.3	1.7	0.7	0.5	1.2
7	31.9	0.8	0.5	0.3	0.6

Abbildung 3.7: Approximationsfehler des jeweiligen DOFs [17]

Bei Betrachtung der Rechenzeit für die Vorhersage aller sieben Freiheitsgrade (DOF) eines gesuchten Punktes (vgl. Abbildung 3.7) ist LWPR besser als LGP. GPR und SVR verhalten sich ähnlich, sie benötigen nochmals länger für die Vorhersage.

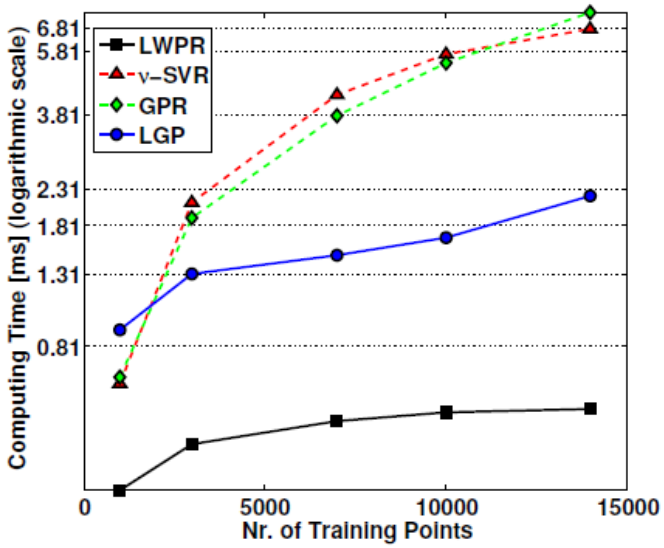

Abbildung 3.8: Durchschnittliche Zeit zur Vorhersage der 7 DOF eines Punktes [17]

Diese Aussagen über Genauigkeit und Geschwindigkeit sind jedoch mit Vorsicht zu betrachten, da diese über die Einstellungen der jeweiligen Algorithmen beeinflusst werden können. So spielt es beispielsweise bei LGP und LWPR eine wichtige Rolle, wie die Parameter gesetzt werden, die festlegen, wie die Eingangswerte partitioniert und somit den jeweiligen Modellen zugeordnet werden.

3.3 Gegenüberstellung

Die Modellierungen mittels Neuronale Netzen und Regressionsalgorithmen weißen sehr viele Ähnlichkeiten auf. Obwohl Neuronale Netze auf Beobachtungen in der Biologie beruhen und Regressionsalgorithmen rein mathematisch entwickelte Verfahren sind, leisten beide Verfahren eine Annäherung an die den Trainingsdaten zu grundliegende Funktion.

Ist ein komplexes System zu simulieren, so wird sowohl bei Neuronalen Netzen, als auch bei Regressionsalgorithmen mit globaler Regression (z.B. GPR), viel Rechenleistung benötigt. Das Neuronale Netz benötigt bei zunehmender Problemkomplexität immer mehr Neuronen im Hidden-Layer, was eine Vervielfachung der zur Simulation benötigten Rechenoperationen nach sich zieht. Beim GPR-Verfahren ist das Problem ähnlich, da für komplexe Zusammenhänge mehr Daten und somit eine größere Kovarianzmatrix benötigt werden, was ebenfalls eine Vervielfachung der zur Simulation benötigten Rechenoperationen nach sich zieht.

Es existieren Verfahren die weniger Rechenzeit benötigen. Auf Seiten der Regressionsverfahren wären das LWPR und LGPR, bei den Neuronalen Netzen wären das die RBN. Bei den Regressionsalgorithmen wird das globale Modell durch viele lokal gültige Modelle repräsentiert. Der Vorteil ist, dass diese Modelle wesentlich schneller zu berechnen sind als ein großes Modell. Bei den RBN haben die Gaußfunktionen nur eine lokale Gültigkeit (vergl. Abbildung 3.5). Unserer Meinung nach könnte dieser Umstand

dazu genutzt werden, nur die Neuronen zu berechnen die das Ergebnis beeinflussen. Dadurch könnte bei großen Netzwerken Rechenzeit eingespart werden.

Außerdem ist anzunehmen, dass Neuronale Netze besser aus verrauschten Signalen lernen können als Regressionsalgorithmen. Die Lernstrategie des Neuronale Netzwerks beruht auf Fehleroptimierung was bedeutet, dass die Neuronen die Konfiguration einnehmen, welche den geringsten Fehler hervorruft. Dies zieht einen Generalisierungseffekt nach sich, der im besten Fall das Rauschen ausmittelt. Diese Art der Rauschunterdrückung ist bei den Regressionsalgorithmen nur bedingt gegeben.

Die Frage nach dem besseren Verfahren kann nicht eindeutig beantwortet werden. Die einzelnen Literaturstellen geben nur an, dass jedes Verfahren zur Modellierung verwendet wurde. Nicht ersichtlich war allerdings die Performanz der einzelnen Verfahren im direkten Vergleich.

Für beide Approximationsverfahren ist die Auswahl der Zustandsgrößen von zentraler Bedeutung. Wird beispielsweise bei einer bewegten Masse nur die zurückgelegte Strecke nicht aber die aktuelle Geschwindigkeit berücksichtigt, ist es unmöglich ein genaues Modell dieses Systems zu erstellen. Für eine erfolgreiche Modellierung müssen also nach Möglichkeit alle relevanten Zustandsgrößen messtechnisch erfasst werden und nicht nur die zu simulierenden Größen.

Die messtechnische Erfassung aller Zustandsgrößen kann bei komplexen Systemen sehr aufwendig werden. Durch Integration oder Differentiation können zum Beispiel aus der gemessenen Geschwindigkeit die zurückgelegte Strecke oder die Auftretende Beschleunigung berechnet werden. Bei der numerischen Integration mit einem der drei gängigen Verfahren (Forward-Euler, Backward-Euler und Trapez) ist darauf zu achten, dass bei der Signalquelle ein Offsetabgleich durchgeführt wurde. Ohne diesen würde der Offset aufintegriert werden, was zu einem falschen Ergebnis führt. Bei der numerischen Differentiation sollten keine Differenzenquotienten verwendet werden, um die Ableitung des Signals zu bestimmen, da das Rauschen bei diesem Verfahren verstärkt wird. So ist mit einem schlechten Nutzsignal-Rausch-Verhältnis zu rechen. Ein Besseres Verfahren zur Ableitung wäre mit Hilfe einer Ausgleichsgerade (vgl. [19]). Hierbei wird das Rauschen unterdrückt in dem eine Ausgleichsgerade durch eine bestimmte Anzahl von Messwerten gelegt wird. Die Steigung dieser Gerade ergibt die Ableitung. Die unten stehende Formel stellt diesen Algorithmus da.(3.1 Durch die Fensterlänge w wird die Anzahl der verwendeten Messwerte festgelegt. Wobei m(k) der k'te Messwert aus dem Datensatz ist und T der Wert der Abtastzeit ist.

$$\dot{y}(k) = \frac{6}{Tw(w^2 - 1)} \sum_{i=1}^{w} \big((w - 2i + 1)m(k - i + 1)\big) \qquad (3.2)$$

Statt numerischer Differentiation könnte auch der Approximationsalgorithmus selbst das Ableitungsverfahren entwickeln, indem das Signal mehrfach zeitverzögert als Eingang verwendet wird.

$$in = \begin{pmatrix} x \\ x' \\ x'' \end{pmatrix} \qquad (3.3)$$

Je mehr Werte aus der Vergangenheit verwendet werden, desto besser kann das Rauschen unterdrückt werden, was das Simulationsergebnis verbessert. Allerdings erhöht dies auch deutlich die Komplexität des Modells. Bei rechenzeitoptimalen Modellen müssen deswegen Differentiationsalgorithmen verwendet werden, da diese zum Lösen dieser Aufgabe weniger Rechenoperationen benötigen.

4 Diskussion

4.1 Gegenüberstellung parametrisierter und nicht parametrisierter Modelle

Eine globale Aussage darüber, welche Form der Modellierung am besten geeignet ist, kann in dieser Arbeit nicht getroffen werden. Grund hierfür ist, dass abhängig von den jeweiligen Rahmenbedingungen der Problemstellung, die Verfahren ihre Vor- und Nachteile haben. Die offensichtlichsten Eigenschaften der Modellierungsverfahren werden deshalb im Folgenden dargestellt.

Parametrisierte Verfahren eignen sich besonders, wenn die dynamischen Eigenschaften des Systems möglichst genau bekannt sind und wenn gewünscht ist, dass das gebildete Modell eine physikalische Bedeutung besitzt. Dies kann beispielsweise der Fall sein, wenn mit Hilfe der angenäherten Parameter Rückschlüsse auf Systemeigenschaften gezogen werden sollen. Ferner hat die Verwendung eines Simulationstools den Vorteil, dass das Modell visuell dargestellt, und das Problem relativ leicht in seine Subsysteme unterteilt werden kann. Auf diese Weise können die einzelnen Teilkomponenten analysiert, und, im Falle eines Austausches von Komponenten, relativ leicht im Modell angepasst werden.

Die Stärke von nicht parametrisierten Modellierungsverfahren liegt in der Modellierung unbekannter Systeme. Zur Bildung eines Modells sind weder die physikalischen, noch die dynamischen Zusammenhänge notwendig. Aus diesem Grund eignet sich dieser Ansatz speziell für Systeme mit vielen unbekannten, beziehungsweise ungenau bekannten Zusammenhängen. Da der Modellierung kaum beschränkende Annahmen zugrunde liegen, welche an die Messdaten angepasst werden sollen, kann für die gegebenen Messdaten eine vergleichsweise genaue Modellbildung erfolgen.

Als wichtigste Voraussetzung für nicht parametrisierte Modellierung ist zu erwähnen, dass eine Vielzahl an Messdaten notwendig ist, welche den gesamten Arbeitsbereich abdecken müssen. Das bedeutet, dass möglichst viele Kombinationen von Zuständen und Aktionen in den Messwerten enthalten sein sollten. Mit zunehmender Komplexität des Systems sind deshalb mehr Messdaten erforderlich. Es ist dabei explizit darauf zu achten, dass bei der Aufzeichnung der Messwerte alle relevanten Zustände miteinbezogen werden, da deren Modellierung sonst nicht möglich ist.

Im Gegensatz hierzu können parametrisierte Modelle theoretisch bereits mit vergleichsweise wenigen Messwerten gelernt werden, da deren Modell für den betrachteten Arbeitsbereich generalisierbar ist. Dabei muss jedoch unbedingt berücksichtigt werden, dass die Genauigkeit der Parameterbestimmung mit der Anzahl

an Messdaten zunimmt, da beispielsweise bei zu wenigen Daten unter Umständen nur lokal optimale Parameter bestimmt werden.

Als Problem bei parametrisierten Verfahren muss genannt werden, dass das verwendete DGL-System in der Regel das System nur zu einem gewissen Maße beschreibt und deshalb viele kleine Einflussfaktoren vernachlässigt werden. Als Beispiel sei die Annahme der homogenen Masseverteilung innerhalb eines festen Körpers genannt, welche oft nicht der Realität entspricht. Gewisse Vereinfachungen sind bei dieser Art der Modellierung auch schlichtweg notwendig, um die damit verbundene Rechenintensität zu begrenzen. Unter dem Gesichtspunkt der Genauigkeit sind deshalb die nicht parametrisierten Verfahren, vorausgesetzt eine ausreichende Anzahl an Daten steht zur Verfügung, in der Regel etwas besser.

4.2 Autonomes Lernen mit vielen Freiheitsgraden

Wie bereits erwähnt, nimmt der Aufwand zur Modellierung eines Systems mit der Anzahl an Freiheitsgraden stark zu. Bei Systemen wie beispielsweise humanoiden Robotern, mit 30-50 Freiheitsgraden und einem ungenauen mechanischen und dynamischen Modell, ist der Zustands-Aktions-Raum so groß, dass auch das Maschinelle Lernen einer Steuerstrategie, wenn überhaupt, nur mit sehr viel Aufwand realisierbar ist. Diese Tatsache ist unabhängig von der Art der Modellierung. Um trotzdem Steuerstrategien für diese Systeme lernen zu können, befasst sich die Wissenschaft mit einer makroskopischeren Repräsentation von Steuerstrategien. Hierzu zählen beispielsweise die so genannten Movement Primitives. Movement Primitives sind parametrisierte Steuerstrategien, welche eine komplette Bewegung ausführen können. Die Erstellung dieser Primitiven kann beispielsweise durch Supervised Learning oder mit Hilfe von Reinforcement Learning erfolgen. Dabei besteht eine Primitive jeweils aus einem kanonischen, und einem transformierenden DGL-System, welche jeweils angepasst werden müssen. Für tiefergehende Information zu Movement Primitives, bzw. Dynamic Movement Primitives (DMP) sei auf [20] verwiesen.

Die Erwähnung von Movement Primitives in dieser Arbeit basiert darauf, dass gemäß [20] die Existenz von Movement Primitives die einzig erdenkliche Möglichkeit ist, wie autonome Systeme (mit sehr vielen Freiheitsgraden) mit der Komplexität von Motor-Steuerung und Motor-Lernen zurechtkommen können. Es ist deshalb bei derartigen Problemstellungen zu überdenken, ob es sinnvoll ist, ein Zustands-Aktions-Modell des gesamten Systems zu erstellen, oder ob vorzugsweise Movement Primitives gelernt werden sollen. Optimal wäre vermutlich eine Kombination aus beidem. Auf diese Weise könnten die Kombination und Parametrisierung der Movement Primitives am Simulator gelernt und überprüft werden.

4.3 Hybride Systeme

Um die Genauigkeit der Simulationsmodelle zu erhöhen, liegt der Gedanke nahe, eine Kombination aus parametrisiertem und nicht parametrisiertem Modell einzusetzen. Ein

solches System würde die Vorteile beider Ansätze kombinieren. Es würde vermutlich ausreichen, für den parametrisierten Teil der Simulation nur die wichtigsten DGLs aufzustellen. Ein Beispiel hierfür wäre eine sich bewegende Masse welche einfach durch DGL`s zu beschreiben ist. Diese Bewegungsgleichungen können mit Parameteroptimierung angepasst werden. Durch einen nachgeschalteten, nicht parametrisierten Teil könnten anschließend schwerer zu modellierende Effekte, wie z.B. Reibung oder eine nichthomogene Gewichtsverteilung berücksichtigt werden. In der Ausarbeitung "Learning to Drive and Simulate Autonomous Robots with Reinforcement Learning" [21] wird eine solche Kombination verwendet, um die Genauigkeit der Simulationsumgebung zu erhöhen. Dies geschieht indem zusätzlich ein Neuronales Netz zur Berücksichtigung von Effekten, wie Traktion und Todzeiten, eingesetzt wird.

4.4 Online Anpassung

Ein weiterer wichtiger Aspekt ist der Einsatz im Feld. So ist in autonomen Robotersystemen mit einer begrenzten Rechenzeit und Speicherkapazität zu rechnen. Ein mögliches Schema zur Online Anpassung ist in Abbildung 4.1 dargestellt.

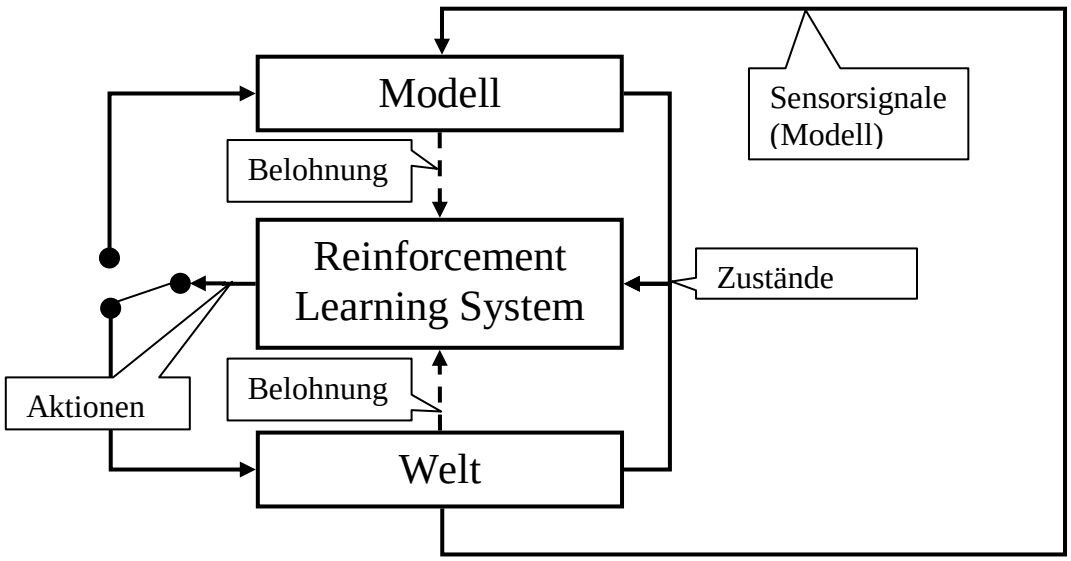

Abbildung 4.1: Schematischer Ablauf der Online Anpassung

Der Grundgedanke dieses Systems ist, dass das RL-System sowohl Aktionen in der Welt, als auch in der Simulation ausführt, um eine Regelungsstrategie zu entwickeln. Außerdem wird basierend auf den Messwerten, welche durch die real ausgeführten Aktionen gewonnen werden, das Simulationsmodell verbessert. Da starke Einschränkung der Rechenleistung und der Speicherkapazität vorliegen, müssen Algorithmen eingesetzt werden, welche hinsichtlich dieser Kriterien optimal sind. Eine Möglichkeit wäre der Einsatz eines Radial Bases Netzes. Bei diesem Netztyp ist es möglich, sowohl die Lern- als auch die Simulationsphase effizient zu gestalten (vgl. [6]). Auch der Einsatz der lokalen Regressionsalgorithmen ist denkbar, während der Rechenaufwand zur Onlineoptimierung von komplexen Parametermodellen zu groß erscheint.

5 Ausblick

In dieser Arbeit wurden im Grunde vier verschiedene Modellierungsverfahren vorgestellt. Dabei wurde deutlich, dass sowohl parametrisierte, als auch nicht parametrisierte Verfahren abhängig von der jeweiligen Problemstellung ihre Berechtigung haben. Unter der Voraussetzung, dass genügend Messdaten des Systems erfasst werden können, welche den gesamten Arbeitsraum hinreichend abdecken, sind wir der Meinung, dass nicht parametrisierte Modellierungsmethoden zu bevorzugen sind. Der Grund hierfür ist der geringere Aufwand, da das physikalische Modellieren entfällt, und Nichtlinearitäten, welche bei einer Beschreibung durch vereinfachte DGLs oft nicht berücksichtigt werden, automatisch miteinbezogen werden.

Eine mögliche Folgearbeit könnte sich deswegen damit beschäftigen, diese These zu bestätigen. Hierzu würden wir vorschlagen die einzelnen Modellierungsverfahren, beispielsweise mit dem am Lehrstuhl existenten Pencil Balancer zu evaluieren. Auf diese Weise könnte ein empirisch gestützter, direkter Vergleich durchgeführt werden.

Abbildungsverzeichnis

Literaturverzeichnis

[1] H. Hultgren and H. Jonasson, *Automatic calibration of vehicle models*, 2007.

[2] P. Abbeel, a Coates, and a Y. Ng, "Autonomous Helicopter Aerobatics through Apprenticeship Learning," *The International Journal of Robotics Research*, Jun. 2010.

[3] J. Bédorf and S. Korzec, *Finding the Optimal movement patterns for continuous virtual creatures using discrete reinforcement learning algorithms*, 2007.

[4] J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the theory of neural computation*, Westview press, 1991.

[5] C.M. Bishop, *Neural networks for pattern recognition*, Oxford University Press, USA, 1995.

[6] W. Shi, J. Yang, and T. Tang, "RBF NN Based Marine Diesel Engine Generator Modeling," *Proceedings of the American Control Conference*, 2005, p. 2745.

[7] Y. Tajima and T. Onisawa, "Model-based reinforcement learning with model error and its application," *SICE, 2007 Annual Conference*, IEEE, 2008, p. 1337–1340.

[8] Y. Zhang, "Support Vector Regression for Basis Selection in Laplacian Noise Environment," *IEEE Signal Processing Letters*, vol. 34, Nov. 2007, pp. 783-874.

[9] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Computed torque control with nonparametric regression models," *2008 American Control Conference*, Jun. 2008, pp. 212-217.

[10] C.E. Rasmussen, "Gaussian processes for machine learning.," *International Journal of Neural Systems*, vol. 14, Apr. 2006, pp. 69-106.

[11] J. Ko, D.J. Klein, D. Fox, and D. Haehnel, "Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp," *Proceedings 2007 IEEE International Conference on Robotics and Automation*, Apr. 2007, pp. 742-747.

[12] C.E. Rasmussen and M. Kuss, “Gaussian Processes in Reinforcement Learning,” *Advances in Neural Information Processing Systems*, vol. 16, Apr. 2004, pp. 751-759.

[13] S.S. Sethu Vijayakumar, Aaron D’Souza, “Incremental Online Learning in High Dimensions,” 2005.

[14] A.Y. Ng, H.J. Kim, M.I. Jordan, and S. Sastry, “Autonomous helicopter flight via Reinforcement Learning.”

[15] D. Nguyen-Tuong, M. Seeger, and J. Peters, “Model Learning with Local Gaussian Process Regression,” *Advanced Robotics*, vol. 23, Oct. 2009, pp. 2015-2034.

[16] D. Nguyen-Tuong, M. Seeger, and J. Peters, “Real-time local gp model learning,” *From Motor Learning to Interaction Learning in Robots*, 2010, p. 193–207.

[17] D. Nguyen-Tuong and J. Peters, “Local gaussian process regression for real-time model-based robot control,” *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, IEEE, 2008, p. 380–385.

[18] D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schölkopf, “Learning inverse dynamics: a comparison,” *Proc. of the Euro. Symp. on Artificial Neural Networks*, Citeseer, 2007.

[19] J. Prock, “Prozessdatenverarbeitung / Systeme und Methoden der Prozessautomatisierung,” 2008, pp. 67-72.

[20] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, “Learning movement primitives,” *Robotics Research*, 2005, p. 561–572.

[21] T. Hester, M. Quinlan, and P. Stone, “Generalized model learning for reinforcement learning on a humanoid robot,” *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, IEEE, 2010, p. 2369–2374.