Hochschule für Angewandte Wissenschaften Hamburg

Hamburg University of Applied Sciences

Bachelor Thesis

Phillip Gesien

Bewertendes Lernen optimaler Bremspunkte in Kurvenfahrten

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Phillip Gesien

Bewertendes Lernen optimaler Bremspunkte in Kurvenfahrten

im Studiengang Technische Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Stephan Pareigis
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Abgegeben am 20. Oktober 2013

Phillip Gesien

Thema der Bachelorarbeit

Bewertendes Lernen optimaler Bremspunkte in Kurvenfahrten

Stichworte

C++, Bestärkendes Lernen, Reinforcement Learning, Q-Learning, autonomes Fahren, FAUST, TORCS

Kurzzusammenfassung

Diese Arbeit zeigt den Weg zur Umsetzung eines Agenten, der selbständig lernen soll, wie ein vorgegebener Parcours möglichst schnell bewältigt werden kann. Um die Notwendigkeit des Lernverfahrens aufzuzeigen, wird zuerst die Physik in Kurvenfahrten und ihre Komplexität erklärt. Danach wird in einem kontinuierlichen Zustands- und Aktionsraum der Reinforcement Learning Algorithmus modelliert und umgesetzt. Während der ganzen Arbeit wird auf die Anforderungen der FAUST-Plattform hingewiesen.

Phillip Gesien

Title of Bachelor Thesis

Reinforcement lerning of breaking points for driving turns

Keywords

C++, Reinforcement Learning, autonomous driving, Q-Learning, FAUST, TORCS

Abstract

This work shows a way to implement an agent, capable to learn to drive a given track with a high speed. To illustrate the need of a machine learning technique, the physics and the complexity of driving a turn are shown. After this there is a description of the design and implementation for a a reinforcement learning algorithm with a continuous state-action-space. There are several references to the requirements of the FAUST platform.

Inhaltsverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

1 Einleitung

Ein Mensch, insbesondere ein professioneller Rennfahrer, kann vor einer Kurve sehr gut bis perfekt einschätzen kann wann er bremsen muss. Der Rennfahrer optimiert seine jeweilige Rundenzeit über die Erfahrung im Laufe seiner Karriere. So muss er die Strecke, auf der er fahren wird, perfekt auswendig kennen und wissen wann er bremsen und beschleunigen muss. Dieses Wissen erlangt er über ein langes Training. Mit Hilfe von Simulatoren und häufigen Testfahrten weiß der Fahrer an welcher Stelle er genau bremsen muss, damit er möglichst schnell um die Kurve fahren kann. Ein intelligentes, autonomes System hat mit solch einer Einschätzung des Bremspunktes in vielerlei Hinsicht Probleme. Viele Faktoren beeinflussen den optimalen Bremspunkt und der Mensch kann diese anhand von Erfahrung verarbeiten. Diese Faktoren müssen im autonomen System rechtzeitig erkannt werden und korrekt interpretiert werden. Weil aber eine große Menge an Daten (die des eigenen Fahrzeugs und der Umwelt) verarbeitet werden muss, werden die Daten sinnvoll reduziert und mit Hilfe von lernenden Algorithmen ausgewertet. Dabei benötigt auch der Agent viel Training und gute Algorithmen, die ihm dazu verhelfen ein guter Rennfahrer zu werden.

1.1 Motivation

Das Studienprojekt „FAUST - Carolo Cup“ beschäftigt sich mit einem autonomen Fahrzeug. Dort steht ein Modellfahrzeug im Maßstab 1:10 mit einem Pico-PC, einem Mikrokontroller, einer Kamera und diversen Sensoren zur Verfügung. Im Folgenden wird dieses Auto „Carolo-Cup Fahrzeug„genannt, weil es an dem gleichnamigen Wettbewerb in Braunschweig teilnehmen soll. Die Software auf dem Computer ist vollständig von Studenten der HAW Hamburg in C++ entwickelt. Als Betriebssystem läuft ein Debian-Linux zur Unterstützung im Bereich der Treiber für die Kamera und für die Netzwerkschnittstelle. Darauf wurde der sogenannte „FAUSTcore“ aufgebaut, der mit Hilfe einer Plugin Architektur alle Aufgaben zur Steuerung des Fahrzeugs regelt. Das System steuert mittlerweile sehr stabil durch jeden beliebigen Kurs. Dennoch ist es wichtig, dass die Geschwindigkeit nicht zu hoch ist, weil das Fahrzeug sonst über- bzw. untersteuert und die Spur verlässt. Weiterhin ist zu beachten, dass die Streckenbedingungen nicht dauerhaft die selben sind. Es sammelt sich viel Staub und Dreck auf

der Strecke, wenn die Strecke schon länger nicht geputzt wurde. Damit nimmt die Bodenhaftung stark ab und das Fahrzeug kann nicht wie zuvor die Geschwindigkeit halten. Deswegen ist ein adaptierender Algorithmus hier ganz wichtig. Dennoch ließe sich die optimale Geschwindigkeit mit Hilfe von vielen genauen Messungen analytisch bestimmen. In der Praxis fehlen aber so viele Größen, wie zum Beispiel die Haftreibung der Reifen, die genaue Gewichtsverteilung oder die korrekte Bremskraft, sodass ein Lernalgorithmus die richtige Wahl ist.

1.2 Ziel der Arbeit

Ziel der Arbeit ist einen Algorithmus zu entwickeln, welcher lernen soll, ein Fahrzeug so schnell wie möglich über einen festen Parcours zu steuern. Dabei ist die zu fahrende Spur vorgegeben und das Fahrzeug wird von externen Modulen gelenkt. Mit Hilfe von Reinforcement Learning soll das System die optimale Geschwindigkeit an verschiedenen Streckenpunkten lernen und darüber Aussagen machen können, an welchen Punkten gebremst und beschleunigt werden soll. Diese Problemstellung ist aus dem Projekt „FAUST - Carolo Cup“ [FAUST-HAW-HH, 2013] übernommen.

Aufgrund der vorhandenen Architektur wird die vorliegende Bachelorarbeit in C++ implementiert und versucht die Simulationsumgebung ähnlich wie auf dem Fahrzeug zu gestalten.

Um das Konzept zu prüfen, wird die Arbeit nicht mit dem Modellfahrzeug getestet, sondern mit Hilfe des Fahrzeugsimulators „The Open Racing Car Simulator“ (kurz TORCS) [Wymann und Espié, 2013] realisiert. Der Simulator bietet eine reale Simulationsumgebung mit guter Fahrzeugphysik. Das Programm kann damit schneller getestet und auf vielen verschiedenen Strecken ausprobiert werden. Weil TORCS ein Open-Source Projekt ist, wird dieses kontinuierlich weiterentwickelt und hat eine große Community, die bei Problemen schnell und gut helfen kann. Des Weiteren wird eine Schnittstelle für „Robots“, also autonome Fahrer, angeboten. Diese Schnittstelle ähnelt sehr der Architektur des Carolo-Cup Fahrzeugs, weil in jedem Schedulingzyklus jeweils eine selbst erstellte Funktion ausgeführt wird und alle Daten zum Fahrzeug und der Strecke bereitgestellt werden. Aufgrund dieser Tatsache kann das endgültige Programm leicht auf die Plattform des Carolo-Cup Fahrzeugs portiert wer-

den. Lediglich die Sensoren müssen mit dem umfangreichen Datenbestand des Programms simuliert und mit einem Wrapper für das Lernproblem vorbereitet werden.

1.3 Gliederung der Arbeit

Zunächst werden im zweiten Kapitel die physikalischen Grundlagen für die analytische Berechnung der Geschwindigkeit erläutert. Dabei werden immer die verfügbaren Mittel beachtet. So können nur begrenzt rechenintensive Verfahren oder genaue Messinstrumente genutzt werden. Hier wird deswegen zuerst von einfacher Mechanik ausgegangen und diese dann mit weiteren Faktoren erweitert. Zuletzt werden dort die Probleme aufgezeigt und somit eine Grundlage für das dritte Kapitel geschaffen.

Dort folgt dann die Vorstellung des zugrundeliegenden Lernalgorithmus. Es wird zuerst überlegt wie das Lernen vonstatten geht und welche Parameter notwendig sind, damit das Lernen erfolgreich wird. Das Ziel wird dort mit Hilfe von Belohnung endgültig festgelegt und zuletzt wird der genaue Algorithmus beschrieben.

Die Implementierung wird im vierten Kapitel mit Hilfe von UML-Diagrammen und Pseudocode erklärt. Weiterhin werden aufgetretene Probleme und deren Lösungen vorgestellt.

Im fünften und letzten Kapitel findet eine Auswertung statt. Diese zeigt anhand von Diagrammen ein Ergebnis und kann verschiedene aufgetretene Probleme und Phänomene anschaulich erläutern. Ein Fazit fasst zum Schluss alles noch einmal zusammen.

2 Physikalische Grundlagen

Viele Probleme im Forschungsbereich der künstlichen Intelligenz lassen sich auch mit einem analytischen Verfahren lösen. Die genaue Berechnung der Aktion wäre dann perfekt und der Agent könnte sich optimal im Zustandsraum bewegen. Dennoch sind einige Probleme viel zu komplex, als dass diese schnell und fehlerfrei berechnet werden könnten. Meist ist der Zustandsraum viel zu groß, bzw. es gibt viele unbekannte Variabeln oder die Umwelt ist schlicht nicht deterministisch. In solchen Fällen kann das Lernverfahren eine gute Näherung der Lösung darstellen. Um aufzuzeigen, welche Probleme das analytische Verfahren beim Finden von optimalen Bremspunkten haben kann, werden im Folgenden die physikalischen Grundlagen zum Kurvenfahren und zum Bremsen erläutert. Danach werden anhand der vorgestellten Formeln die kritischen Punkte herausgehoben und erläutert.

2.1 Kräftegleichgewicht

Wirken auf einem Körper Kräfte, dann müssen alle wirkenden Kräfte sich gegenseitig aufheben, damit der Körper stabil bleibt.

$$\sum \vec{F_i} = \vec{0} \tag{1}$$

In diesem Fall heißt stabil, dass das Fahrzeug auf der Straße fährt und nicht aus der Kurve heraus geschleudert wird. Sollte das Fahrzeug die Strecke verlassen, dann wirkt eine zu hohe Zentrifugalkraft F_Z auf das Fahrzeug und die gegensätzliche Reibungskraft F_R ist nicht hoch genug um das Fahrzeug stabil zu halten. Damit das nicht passiert, muss die Reibungskraft größer, bzw. gleich so groß sein wie die Zentrifugalkraft. Allgemein ergibt sich folgende Näherung:

$$F_Z = F_R \tag{2}$$

$$\frac{mv^2}{r} = mg\mu \tag{3}$$

Dabei ist m die Masse des Fahrzeugs, v die Geschwindigkeit, r der Kurvenradius, μ der Reibungskoeffizient der Straße und g die Erdschwerebeschleunigung von $\approx 9,81\text{m/s}^2$. Diese Näherung wäre ausreichend, um sicher durch den Kurs zu fahren. Formel 3 lässt sich umformen zu

$$v_{max} = \sqrt{rg\mu} \tag{4}$$

Damit diese maximale Kurvengeschwindigkeit v_{max} rechtzeitig vor der Kurve erreicht werden kann, muss gebremst werden. Mit der Bremsung wird die aktuelle kinetische Energie E_{akt} des Fahrzeugs abgebaut, bis das Fahrzeug nur noch die erforderliche maximale Energie E_{max} für die Kurvenfahrt besitzt. Dieser Vorgang lässt sich beschreiben mit

$$E_{akt} - E_{max} = E_{Brems} \tag{5}$$

$$\frac{m{v_{akt}}^2}{2} - \frac{m{v_{max}}^2}{2} = mg\mu s \tag{6}$$

wobei E_{Brems} die von den Bremsen abgebaute Energie über die Strecke s ist. Es kann also gesagt werden, dass

$$s = \frac{{v_{akt}}^2 - {v_{max}}^2}{2\mu g} \tag{7}$$

Meter vor der Kurve gebremst werden muss.

Jedoch gibt es hier noch weitere Kräfte zu beachten. Die aerodynamischen Teile des Fahrzeugs helfen mit ihrem Abtrieb der Bremskraft, weil die Reibungskraft dadurch höher wird. Weiterhin bremst der Luftwiderstand das Fahrzeug zusätzlich ab. Somit ergibt sich

$$E_{akt} - E_{max} = E_{Brems} + E_{Abtrieb} + E_{Luftwiderstand} \tag{8}$$

Weil der Abtrieb und der Luftwiderstand des Fahrzeugs mit höherer Geschwindigkeit größer

werden, müssen die Energien $E_{Abtrieb}$ und $E_{Luftwiderstand}$ aufwendiger berechnet werden. Es ergibt sich eine Differentialgleichung, die gelöst werden muss. Für die Berechnung der entstehenden Gleichung wird der Widerstandsbeiwert des Fahrzeugs benötigt.

2.2 Probleme der analytischen Lösung

Für die in Kapitel 2.1 vorgestellten Formeln werden einige Variablen benutzt, die nur schwer bestimmbar sind.

In Gleichung 2 wird angenommen, dass die Reibungskraft der Reifen perfekt gleich verteilt ist und an dem Punkt wirkt, an dem auch die Zentrifugalkraft angreift. Eine Bestimmung der Reibungskraft ist bei einem Modellfahrzeug sehr schwierig und kann sich schnell ändern, da schon ein wenig Staub die Reibung stark verschlechtern kann. Dagegen kann die Masse und die Geschwindigkeit mit geringem Aufwand sehr genau bestimmt werden. Dennoch muss zum Berechnen des genauen Angriffspunkt der Zentrifugalkraft der Schwerpunkt des Fahrzeugs errechnet werden. Erschwert wird dies durch die hohen Aufbauten und selbst gebauten Sensorbefestigungen. Wegen der nicht immer genau gleichen Position der Akkus ändert sich der Schwerpunkt zusätzlich minimal und kann in der Folge dafür sorgen, dass die zulässige Geschwindigkeit falsch berechnet wird oder das Fahrzeug umkippt. Weiterhin muss für die Berechnung der Zentrifugalkraft der Kurvenradius bestimmt werden. Dieser ist in der Regel bis auf wenige Zentimeter genau bestimmbar. Jedoch soll das Fahrzeug verschiedene Kurse abfahren. Vor dem Start müsste jede Kurve vermessen und dem System mitgeteilt werden. Teilweise ist das Bestimmen der Kurvenradien nur schwer möglich (klothoidische Kurven, Schikanen etc.).

Die Bremsenergie aus Gleichung 6 lässt sich ebenso schwer bestimmen. Der Reibungskoeffizient der Bremsen ist, durch die Bremsung des Fahrreglers, auch nicht genau ermittelbar. Mit sinkendem Ladestand der Akkus kann die Bremsleistung zusätzlich nachlassen.

Mit Gleichung 8 werden noch weitere unbekannte Faktoren eingeführt. Bei einem Modellfahrzeug ist es sehr aufwendig und schwierig die erforderlichen Variablen zu bestimmen. Der Widerstandsbeiwert ist durch die vielen Sensoren und Aufbauten schwer zu ermitteln. Eine

Messung im Windkanal ist auf Grund der hohen Kosten für das Studienprojekt nicht zweckmäßig. Weiterhin wird der Luftwiderstand gering sein, so dass die Messungen möglichst genau sein müssen, damit es nicht zu sehr großen Toleranzen bei dem Ergebnis kommt. Der Abtrieb ist ähnlich schwer zu messen. Durch das vergleichsweise hohe Gewicht (ca. 2500g) für ein 1:10 Modellfahrzeug ist der Abtrieb der Karosserie eventuell auch nicht mehr Messbar oder ausschlaggebend.

Darüber hinaus müssen noch weitere fahrdynamische Aspekte betrachtet werden. Das Fahrzeug besitzt sowohl an der Hinter-, als auch an der Vorderachse, ein Differential, welches die Kraft dynamisch auf die Reifen verteilt. Ein weiterer Faktor ist das hohe Gewicht, welches die mechanischen Teile des Modellfahrzeugs an ihre Grenzen bringt. Zum Beispiel sind die Stoßdämpfer trotzt harter Federn fast am Limit und können deswegen teilweise nicht wie vorgesehen arbeiten.

Auf Grund dieser Reihe von Problemen wird im Folgenden versucht, die Kurvenfahrt zu optimieren. Dabei werden alle zuvor erläuterte Variablen uninteressant, da diese dann Teil der Umwelt sind, die der Agent zu erforschen versucht. Alle Einflüsse sind Teil des Problems und werden somit beachtet. Der Vorteil dieser Methode ist, dass es auf unterschiedlichen Strecken mit unterschiedlichen Eigenschaften eingesetzt werden kann. So kann zum Beispiel bei einem Wettbewerb das Programm auf die dortigen Bedingungen eingelernt werden und seine Fahrgeschwindigkeit verbessern. Mit Vorwissen, also bekannte minimale und maximale Geschwindigkeiten, wird der Agent schnell lernen, schnell fahren zu können.

3 Bestärkendes Lernen und Modellierung

Lernverfahren bedienen sich verschiedener Herangehensweisen. Es gibt das überwachte, unüberwachte und das bestärkende Lernen. Bei dem überwachten Lernen wird versucht, dass der Algorithmus lernt, wie gegebene Paare von Ein- und Ausgaben zusammengehören. Dabei muss ein Lehrer die Daten zuvor klassifizieren. Das unüberwachte Lernen versucht in gegebenen Daten Muster zu erkennen und ordnet sie. Damit können Vorhersagen gemacht werden. Mit dem bestärkenden Lernen wird dem Algorithmus durch Belohnung oder Bestrafung der Erfolg der Aktion mitgeteilt. Daraufhin kann der Algorithmus entscheiden, ob diese Aktion zielführend ist und ob diese erneut genutzt werden sollte. Dieses Kapitel beschreibt zuerst die Theorie des bestärkenden Lernens und geht danach auf die konkrete Modellierung der Daten für das gegebene Problem ein.

3.1 Bestärkendes Lernen

Der Agent steuert in dieser Aufgabe die Geschwindigkeit des Fahrzeugs. Dadurch agiert er fortwährend mit der Umwelt, also wie sich das Fahrzeug auf der Strecke bewegt. Mit Hilfe von Sensoren weiß der Agent dann was passiert ist und kann die Situation bewerten. Weiterhin kann der Agent einen Erfolg messen und diesen mit in seine Entscheidungen einfließen lassen.

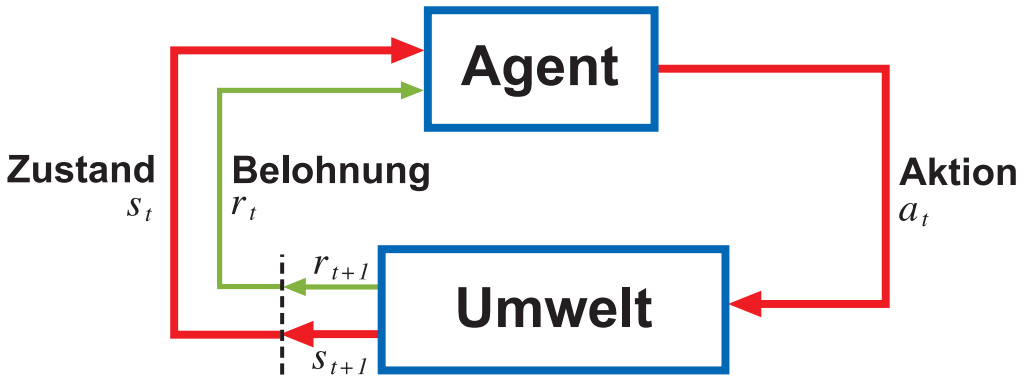

Abbildung 1: Interaktion zwischen Agent und Umwelt.

Die Abbildung 1 zeigt diesen Prozess. Der Agent wählt eine Aktion a_t und führt diese aus. Daraufhin bekommt dieser von der Umwelt die neue Situation s_{t+1} in der er sich befindet. Weiterhin erhält der Agent eine Rückmeldung bzw. Belohnung, r_{t+1}. Mit Hilfe dieser Informationen entscheidet der Agent im nächsten Zeitschritt wieder welche Aktion die Beste wäre und der Kreislauf setzt sich fort. Nun stellt sich die Frage nach dem Umfang der Situation oder der Umwelt. Agiert zum Beispiel ein Laufroboter in der realen Welt, gibt es eine sehr ausführliche Umwelt mit einer großen Menge an messbaren Parametern. Dennoch ist es für den Roboter vermutlich nicht wichtig wie hoch der Luftdruck ist oder welches Wetter gerade herrscht. Die Umwelt muss mit Sensoren auf das Nötigste erfasst werden und somit den Zustand mit den notwendigen Informationen abbilden.

3.2 Belohnung

Als nächstes stellt sich die Frage was denn belohnt werden soll. Der Agent kann nur sinnvoll lernen, wenn er weiß welche Aktion zielführend ist und welche er unterlassen soll. Welches Ziel der Agent verfolgen soll, wird mit der Belohnungsfunktion $r(s)$ beschrieben. Als Ziel wird für den, in dieser Arbeit zu entwickelnden Agenten, definiert, dass das Fahrzeug so schnell wie möglich um den Kurs fahren soll. Dabei darf er die Streckenmarkierung nicht verlassen und soll sich möglichst mittig auf der Strecke halten. Um dieses Ziel zu erreichen gibt es verschiedene Ansätze.

Eine Möglichkeit wäre, die Rundenzeit als Indikator dafür zu nutzen wie schnell der Agent war und daraus eine Belohnung abzuleiten. Dazu muss dann jeweils noch eine Strafzeit für ein mögliches Verlassen der Strecke addiert werden. Der Nachteil dieser Variante ist, dass die tatsächliche Belohnung immer erst viel später bekannt wird und dadurch einzelnen Aktionen nicht zugeordnet werden können, ob diese gut waren oder nicht. Zwar werden alle ausgeführten Aktionen zusammen neu bewertet, aber welche Aktion genau (un-)günstig für die Gesamtbelohnung war lässt sich schwer nachvollziehen. Gerade auf langen Strecken mit vielen Kurven sind viele Bremspunkte vorhanden und somit viele eventuelle Strafzeiten durch Fahrfehler möglich. An welcher Kurve der Agent zu schnell war lässt sich dann mit einer Gesamtzeit nicht nachvollziehen. Damit das möglich ist, müsste die gesamte Strecke

in viele Abschnitte unterteilt werden. Mit einer Kurve pro Abschnitt kann dann die Strecke gut erlernt werden und die Zeiten pro Abschnitt können als Belohnung genutzt werden. Auch hier besteht ein ähnliches Problem wie zuvor. Bei der Kurvenein- bzw. Kurvenausfahrt können auch viele Fehler passieren und diese sind dann nicht nachvollziehbar. Eine verspätete Belohnung erscheint also nicht zweckmäßig.

Besser ist, ein direktes Belohnungssystem einzuführen. Wie in [Abdullahi und Lucas, 2011] beschrieben, ist es sinnvoll, die Belohnungsfunktion mit Hilfe der gefahrenen Strecke der letzten Zeitschritte d_{sum} zu formulieren. Zusätzlich wird der Abstand zur Mitte d_{Mitte} als zusätzliches Kriterium herangezogen, weil das Fahrzeug sich auf einer festen Spur bewegen soll. So ergibt sich

$$r = -\frac{c_1}{d_{sum}} - d_{Mitte} * c_2 \tag{9}$$

wobei c_1 und c_2 Faktoren zur Anpassung der Größenordnung dienen. Ein weiterer Vorteil mit dieser Belohnungsfunktion ist, dass der Agent mit Hilfe des d_{sum} schon früh merkt, dass er zu schnell ist bevor er tatsächlich die Strecke verlässt. Auf der anderen Seite muss c_2 sinnvoll gewählt werden. Ist der Anteil der Abstandsbelohnung vergleichsweise hoch, wird der Agent vermutlich langsam in der Mitte der Strecke fahren als leicht zu schnell durch die Kurven zu fahren und dabei ein wenig rutschen. Je geringer die Geschwindigkeit ist, desto kleiner wird die Belohnung bzw. desto größer die Bestrafung. Bei einer sehr schnellen Fahrt auf der Mittellinie wird die Belohnung dicht an der Null im negativen Bereich sein. Verlässt der Agent die Strecke, wird dieser sofort mit einem sehr hohen negativen Wert bestraft. Die Gesamtbelohnung pro Runde ist dann die Summe aller einzelnen Belohnungen der besuchten Zustände. Mit dieser Belohnungsfunktion ist sichergestellt, dass der Agent sofort seine vergangenen und folgenden Aktionen anders bewerten muss. Diese Funktion modelliert die Aufgabe von einer episodischen Aufgabe zu einer kontinuierlichen Aufgabe um. Dadurch müssen keine Zielzustände definiert werden, das Ziel wird von einer optimalen Rundenzeit zu sehr schnellem Fahren geändert. Aus schnellem Fahren folgt zwar, dass die Rundenzeit besser wird, jedoch hat der Agent die Rundenzeit nicht mehr als Ziel, sondern versucht immer in der Nähe der Mittellinie schnell zu fahren.

Im Rundenzeitmodell wäre eine Möglichkeit, dass der Agent mit Abweichungen von der

Mittellinie in Kurven, aus Unter-/Übersteuern, schneller durch Schikanen kommt und somit betrügt. Weiterhin müssten noch Zielzustände definiert werden, die mit dem Carolo Cup-Fahrzeug auf einer realen Strecke auch berechnet und erkannt werden müssten. Die sofortige Belohnung benötigt keinen Zusatzaufwand, weil die nötigen Messwerte bekannt sind. Aus diesen Gründen wird die sofortige Belohnung in der Implementierung bevorzugt.

3.3 Zustands- und Aktionsraum

Damit der Agent weiß, welche Aktionen er ausführen kann und auf welcher Basis die Beurteilung von Situationen gemacht werden kann, muss der Zustandsraum und der Aktionsraum festgelegt werden. Der Zustands-, bzw. Aktionsraum kann mit Hilfe von mehreren Komponenten beschrieben werden. Sind zu viele Variablen bei der Zustands- bzw. Aktionsbeschreibung erforderlich, wird das Lernen verlangsamt. Es müssen viele unterschiedliche Kombinationen der Zustandswerte ausprobiert werden, damit der Agent optimal lernen kann. Deswegen ist ein Ziel, die Dimensionen von Zustands- und Aktionsraum nicht zu groß zu gestalten.

3.3.1 Aktionsraum

Soll ein humanoider Laufroboter das Laufen erlernen, muss er beide Beine gleichzeitig bewegen. Dass bedeutet, dass der Aktionsraum mehrere Dimension benötigt, damit die gleichzeitige Beinbewegung beschrieben werden kann. Im vorliegenden Fall ist jedoch eine Dimension ausreichend, weil nur die Geschwindigkeit kontrolliert werden soll. Weil es unnütz ist, gleichzeitig zu Bremsen und zu Beschleunigen, wird der Aktionsraum $A(s)$ folgendermaßen festgesetzt:

$$A(s) = \{acc\} \; \forall s \in S \tag{10}$$

Damit ist rein die Pedalstellung gemeint, wobei acc im Bereich $[-1,1]$ definiert ist. Dabei gilt für

-1 $\rightarrow$ Bremspedal auf Maximalstellung und Gaspedal auf Neutralstellung,

0 $\rightarrow$ Gas- und Bremspedal auf Neutralstellung und

$+1$ $\rightarrow$ Gaspedal auf Maximalstellung und Bremspedal auf Neutralstellung.

Es muss hier darauf hingewiesen werden, dass damit keine genaue Geschwindigkeitsvorgabe seitens des Agenten gemacht werden kann. Die Geschwindigkeit kann trotz gleicher Pedalstellung unterschiedlich sein, weil ein anderer Gang genutzt wird. Weil zwischen den Werten -1 und $+1$ theoretisch unendlich viele Zahlen liegen, muss dieser Aktionsraum diskretisiert werden. Ausreichend ist hier eine gleichmäßige Abtastung mit $0,1$ Abstand. Damit ist sichergestellt, dass der volle Aktionsraum ausgeschöpft ist. Insgesamt entstehen so 21 verschiedene Aktionen.

3.3.2 Zustandssraum

Der Zustandsraum ist umfangreicher. Es werden alle Daten benötigt, um eine ausreichende Aussage über die tatsächliche Situation zu machen. Dabei muss beachtet werden, dass nur relevante Daten mit einbezogen werden. Weiterhin ist wichtig, dass die Zustandsbeschreibung den Zustand eindeutig werden lässt (Markov-Eigenschaft). Die Markov-Eigenschaft beschreibt, dass alle für die Zukunft wichtigen Daten in einem Zustand zusammengefasst sind. Wenn kein weiterer Informationsgewinn aus der Hinzuziehung von älteren Informationen erfolgt, dann unterliegt der Zustand der Markov-Eigenschaft. So reicht es nicht aus nur die Geschwindigkeit im Zustand anzugeben. Dabei wird nicht ersichtlich, ob das Fahrzeug im Beschleunigungs- oder Bremsvorgang ist. Aus diesem Grund muss die Geschwindigkeit anders modelliert werden.

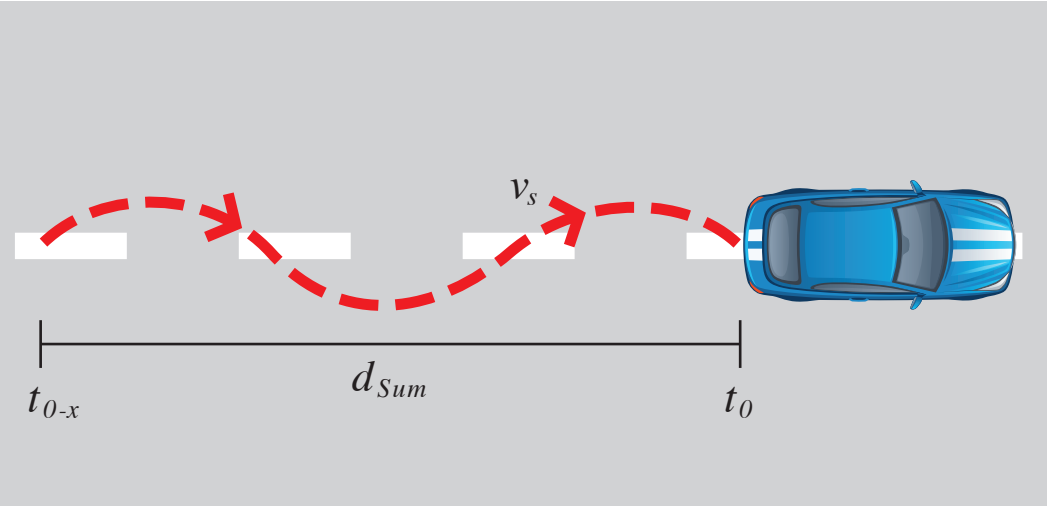

Abbildung 2: Visualisierung der Zustandsvariable d_{Sum}. Auch wenn das Fahrzeug Schlangenlinien mit einer hohen Geschwindigkeit v_s fährt, hat es den gleichen d_{Sum} Wert wie ein Fahrzeug das auf der Mittellinie mit langsamerer Geschwindigkeit fährt.

Geschwindigkeit

Eine Möglichkeit ist die Vergangenheit, also die gefahrene Strecke in einem Wert zu bündeln. Es kann daher die Geschwindigkeit mit der zurückgelegten Strecke in Fahrtrichtung $d(t)$ der letzten Zeitschritte gemessen werden.

$$d_{sum}(t) = \int_{t_{0-x}}^{t_0} d(t)dt \tag{11}$$

Dabei ist t_0 der aktuelle Zeitpunkt und x die Anzahl der Zeitschritte, über denen dieser Wert berechnet werden soll. Wichtig ist, dass nur die Distanz auf der Mittellinie der Spur berechnet wird. Dadurch unterstützt dieses Zustandssignal die Anforderung mittig zu fahren, weil Schlangenlinien mit höherer Geschwindigkeit genauso schnell sind wie eine langsamere Fahrt an der Mittellinie. Das zeigt Abbildung 2. Wenn das Fahrzeug nun eine bestimmte Geschwindigkeit erreicht, kann mit Hilfe des Wertes d_{sum} eine Aussage getroffen werden, ob das Fahrzeug beschleunigt oder bremst. Bei einem Vergleich von zwei verschiedenen d_{sum} mit einer gleichen Endgeschwindigkeit, fällt auf, dass der größere Wert einen Bremsvorgang

beschreibt, weil die Geschwindigkeit vorher höher war und mehr Strecke zurückgelegt wurde. Dieses Wissen ist wichtig, damit Bremsvorgänge erkannt werden können.

Lokalisierung

Ein wichtiger Punkt ist die örtliche Situation. Weil Bremspunkte gefunden werden sollen, muss der Agent wissen wo er sich bewegt. Dazu gehört, dass er weiß,wie weit es noch bis zur Kurve ist und welche Form die Kurve hat. Nur mit diesem Wissen kann dann die Pedalstellung richtig kontrolliert werden. Eine Haarnadelkurve muss mit einer geringeren Geschwindigkeit angefahren werden als eine leichte 45 Grad-Steilkurve. Möglich wäre es, die Strecke mit absoluten Koordinaten zu beschreiben. Der Agent weiß dann nach mehrmaligen abfahren, dass nach einer bestimmten Position eine Kurve folgt und kann dann die Aktion entsprechend anpassen. Dennoch sind absolute Koordinaten an dieser Stelle ungeeignet, da dieser Zustand dann für jede Runde einzigartig ist. Einzigartige Zustände erschweren allgemein das Lernen stark, weil diese innerhalb einer großen Zustandsmenge nur sehr selten auftreten. Auf einem ovalen Rundkurs existieren jedoch zwei gleiche Kurven, die beide die gleiche Kurvengeschwindigkeit haben sollten. Somit würden die gleichen Kurven mit den gleichen Bedingungen unterschiedlich behandelt.

Damit Zustände häufiger auftreten bietet es sich an, die vorausliegende Strecke mit einer Funktion zu interpolieren. Diese Variante ist, längerfristig gedacht, von Vorteil, weil das Carolo-Cup Fahrzeug zur Streckenbeschreibung ein Polynom nutzt und die Portierungsarbeit wegen der Ähnlichkeit geringer wird. Daher wird hier ein Polynom dritter Ordnung verwendet. Ein Polynom erlaubt, Schikanen mit wechselnder Kurvenrichtung zu beschreiben und ist dabei nicht sehr aufwändig zu berechnen.

$$f(x) = ax^3 + bx^2 + cx + d \tag{12}$$

Dieses Polynom hat den Vorteil, dass es vier Parameter a, b, c, d benutzt, die zur Beschreibung des Zustands genutzt werden können. Ähnliche Streckenführungen ergeben ähnliche Polynome und somit auch ähnliche Parameter a, b, c, d. Der Agent kann dann wegen der Ähnlichkeit schon wissen, dass er bremsen oder beschleunigen muss. Damit das Polynom

immer ähnlich ist muss der Koordinatenursprung relativ zum Fahrzeug modelliert werden. Deswegen gilt das Fahrzeug als Ursprung. Die DIN 8855 [DIN8855, 2012] beschreibt unter anderem ein Fahrzeugkoordinatenmodell. Mit einer Darstellung, in der die Y-Achse parallel zu den Achsen des Fahrzeug verläuft und das Fahrzeug sich in Richtung der X-Achse bewegt, wird ein mögliches Modell beschrieben. Dennoch hat dieses Modell kleine Probleme. Aus den folgenden Gründen wird für die Modellierung nicht auf das Fahrzeugkoordinatenmodell der DIN 8855 [DIN8855, 2012] zurückgegriffen, bei der die Y-Achse achsenparallel verläuft. Es können zum Beispiel keine engen Haarnadelkurven mit nur einem Polynom beschrieben werden (siehe Abb. 3). Eine solche Kurve würde keine lineare Abbildung darstellen und wäre somit nicht zu berechnen. Deswegen verläuft bei dem gewählten Modell die Y-Achse, vom Fahrzeug aus gesehen, in Fahrtrichtung und die X-Achse senkrecht dazu, also Achsenparallel. Natürlich ist es nicht möglich die gesamte Strecke mit nur einem Polynom zu approximieren. Deswegen muss überlegt werden, wie weit der Agent voraus schauen soll. In der Simulation könnte der Agent die komplette Strecke im voraus errechnen und auswerten. Dagegen wird der Agent in der Realität unter anderem eine Kamera zur Wegfindung nutzen müssen. Die Kamera kann natürlich nur in Fahrtrichtung und nur begrenzt in die Kurve hinein schauen. Bilder vom dem Carolo Cup-Fahrzeug haben gezeigt, dass diese nur ca. 3m weit blicken und die Kurven nicht vollständig sehen kann. Dementsprechend muss bestimmt werden wie weit der Agent nach vorne Blicken darf, um ein realistisches Verhalten zu erzielen. Ein professioneller Rennfahrer sollte die Strecke, auf der er unterwegs ist, perfekt auswendig kennen. Somit weiß er aus Erfahrung wie eine Kurve aussieht und mit welcher Geschwindigkeit er diese anfahren kann. Dagegen kann ein Verkehrsteilnehmer im normalen Straßenverkehr die Straße nur wenige hundert Meter weit überblicken und einschätzen. Dabei sind sehr scharfe Kurven eventuell gar nicht einsehbar. Weil der Agent so schnell wie möglich fahren soll, muss er die Strecke ähnlich gut wie ein Rennfahrer einschätzen können. Dabei soll er aber nicht zu viele Informationen bekommen, so dass der Agent wiederum ein wenig menschlich wirkt. Das Einschränken der Informationen kommt dann zusätzlich dem Ziel der Einfachheit des Zustandes zu gute. Exemplarische Messungen an einer Haarnadelkurve in TORCS haben gezeigt, wie unterschiedlich sich verschiedene Sichtweiten auf die Polynome auswirken. Die Abbildungen 3 mit einer Sichtweite von 120m und Abbildung 4 mit einer Sichtweite von 200m zeigen wie verschiedene Sichtweiten in der selben Kurve das Polynom beeinflussen.

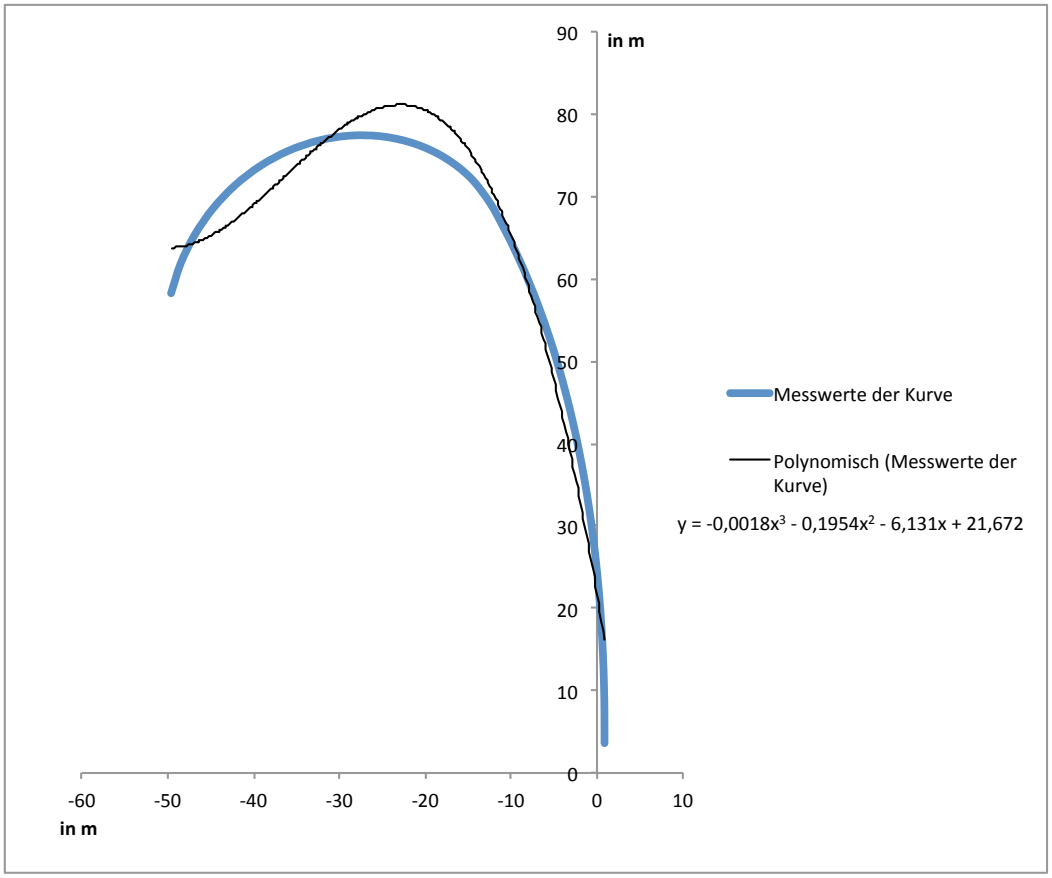

Abbildung 3: Polynom zu Messpunkten mit 120m Voraussicht. Die Hoch- und Querachse beschreibt den Abstand zum Fahrzeug in Metern.

Bei 120m Sichtweite fällt auf, dass das Polynom bis zum Hochpunkt dichter an der tatsächlichen Kurve liegt. Dabei wird jedoch der Kurvenausgang zu dem Zeitpunkt schlecht getroffen und die Kurve verläuft ab -50 wieder in Richtung $+\infty$. Die Interpolation mit einer Sichtweite von 200m hat zur Folge, dass mehr Punkte am Kurvenausgang verfügbar sind. Dadurch wird das Polynom dort stärker gekrümmt. Jedoch kommt es dabei zu einem viel engeren Kurvenradius am Scheitelpunkt der Kurve. Ein engerer Scheitelpunkt muss mit geringeren Geschwindigkeiten angefahren werden. Wegen dieses engen Radius, könnte der Lernalgorithmus sehr enge Kurven eventuell nicht mehr gut voneinander abgrenzen. Die Kurve ist mit 120m Sichtweite, wegen des größeren Kurvenradius und somit auch der genaueren Kurveneinfahrt, ausreichend beschrieben. Abweichungen am Ende der Kurve sind bei dieser Aufgabe uninteressant, weil das Ziel die Anfahrt der Kurve ist. Daher ist es wesentlich wich-

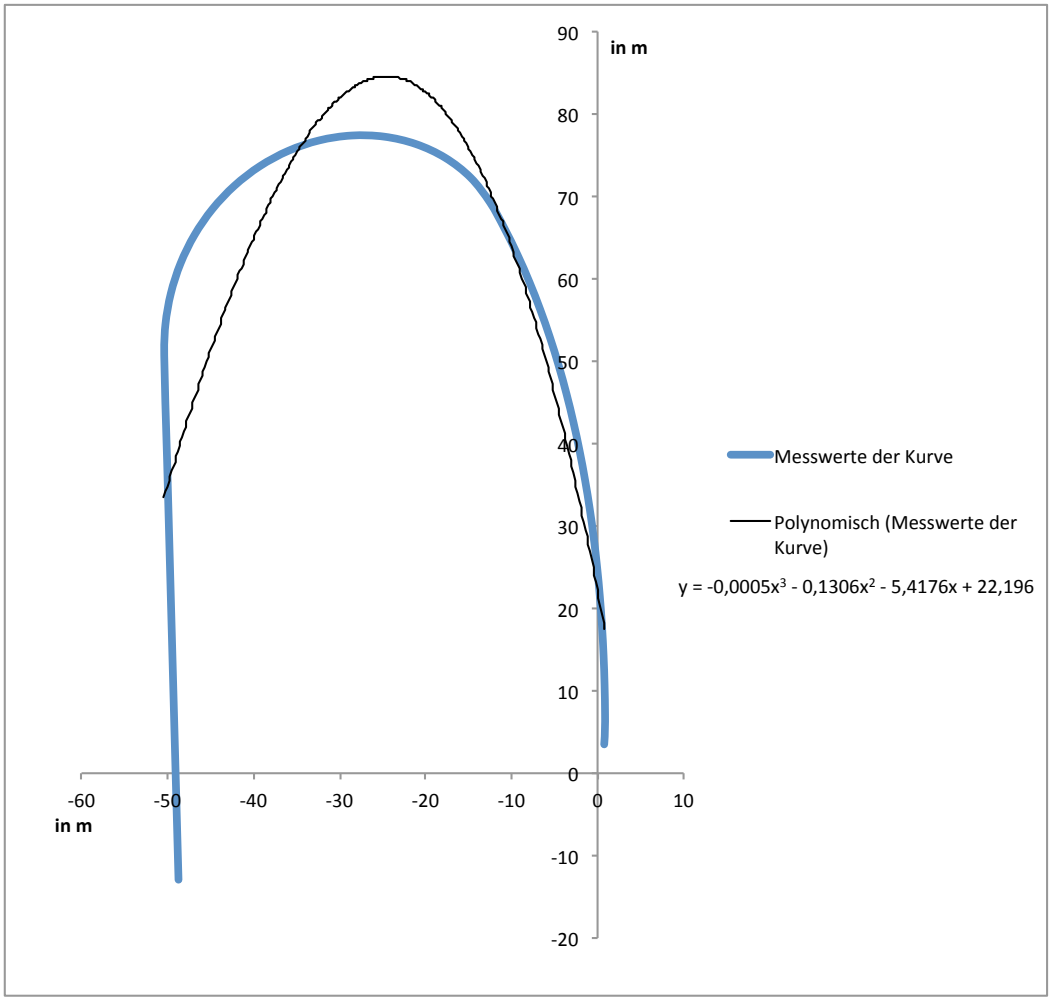

Abbildung 4: Polynom zu Messpunkten mit 200m Voraussicht. Die Hoch- und Querachse beschreibt den Abstand zum Fahrzeug in Metern.

tiger die Charakteristika der Kurve gut abzubilden und dafür wird der Schwerpunkt auf den Kurveneingang gelegt.

Ein weiterer Vorteil der Modellierung mit einem Polynom ist, dass Links-, sowie Rechtskurven nur unterschiedliche Vorzeichen in den Parametern a und c haben. Dadurch können beide Richtungen auf eine Funktion abgebildet werden. Somit treten einige Kurven mehrmals auf und führen so zu einer Reduktion der möglichen Zustände.

Dennoch hat die Darstellung mit Polynomen auch einige Probleme. Der Wertebereich der verschiedenen Parameter ist stark abhängig von der zu interpolierenden Kurve. Gerade Stre-

ckenabschnitte werden durch einen sehr großen Parameter a beschrieben, weil das Polynom dann sehr stark und schnell ansteigen muss. Wohingegen dieser Parameter bei Kurven sehr klein werden kann. Das führt dazu, dass die Werte in der Implementierung angepasst werden müssen. So können sehr große Werte angepasst werden, während Werte nahe Null zu Null werden können. Daher werden sehr kleine Parameter ($|p| < 10^{-5}$) ignoriert und sehr große Parameter ($|p| > 8000$) mit einem Faktor verkleinert. Dennoch können die Parameter Wertebereiche in der Größenordnung bis 500000 haben.

Abstand zur Mitte

Ein weitere Variable zur Beschreibung des Zustandes, ist der Abstand zur Mitte der Strecke d_{Mitte}, weil das Fahrzeug sich innerhalb der Streckenmarkierung und möglichst mittig auf der Fahrbahn bewegen soll. Verschiedene Positionen auf der Strecke erlauben unterschiedliche Geschwindigkeiten in der Kurve. So kann ein Fahrzeug, welches durch eine Schikane fährt mehrere Wege fahren.

Abhängig davon welche Streckenführung vorher vorhanden ist und welche Geschwindigkeit das Fahrzeug fährt, hat es eine andere Ausgangsposition auf der Strecke. Mit der Position A in Abbildung 5 kann der Agent mit einer sehr hohen Geschwindigkeit durch die Rechtskurve fahren, weil er dann nur wenig nach links steuern muss, um die Linkskurve zu fahren. Dabei verlässt der Agent jedoch die angestrebte Mittellinie und muss bestraft werden. Fährt der Agent auf Spur B, dann muss er zwar langsamer fahren als vorher, wird aber nicht wegen Verlassen der Spur bestraft. Dies ist dann das gewünschte Verhalten. Spur C hingegen ist sowohl extrem schlecht für die maximal mögliche Geschwindigkeit, als auch von der Bestrafung, weil die Mittellinie lange Zeit verlassen wird und sehr starke Kurven mit geringer Geschwindigkeit gefahren werden müssen. Um das Verhalten von Spur A und C zu vermeiden, muss der Agent über den Abstand zur Mitte Bescheid wissen.

Damit ist der Zustandsraum vollständig zusammengefasst. Die Position, die Geschwindigkeit und der Abstand zur Mitte beschreiben den Zustand ausreichend, um die Situation einzu-

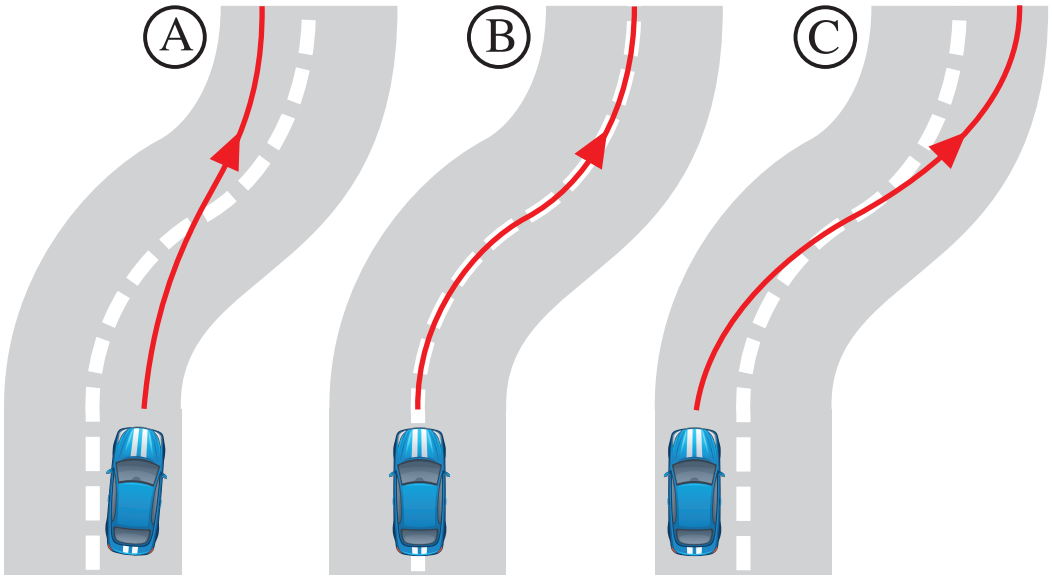

Abbildung 5: Mögliche Trajektorien für eine Schikane. Für die höchste Geschwindigkeit ist die Ausgangsposition des Fahrzeugs entscheidend. Jedoch wird Möglichkeit B am geringsten bestraft.

schätzen. Zusammengefasst ist der Zustandsraum folgender:

$$S = \{d_{sum}, d_{Mitte}, Polynomparameter\} \tag{13}$$

Damit hat der Zustandsraum eine Dimension von 6. Mit dieser Größe ist der Zustandsraum so klein wie möglich gehalten. Die Größe des Zustandsraumes ist unter Anderem ein Faktor für die Lerngeschwindigkeit. Ist der Zustandsraum zu groß, wird das Lernen wegen zu vieler möglichen Zustände verlangsamt.

3.4 Q-Learning

Weil der Agent kein Modell seiner Umwelt besitzt und somit nicht weiß in welchen Zustand ihn eine Aktion führt, muss eine Strategie, basierend auf der Bewertung von möglichen Folgezuständen, genutzt werden. Für die Strategie π wird nun Q-Lernen angewandt. Die beste

Aktion wird mit Hilfe der Q-Funktion ausgewählt:

$$\pi^* = \max_a Q(s,a) \tag{14}$$

Dabei ist π^* die optimale Strategie. Die Q-Funktion setzt sich zusammen aus allen zukünftigen Belohnungen, die jedoch mit zunehmender Entfernung immer weiter abgeschwächt werden.

$$Q(s_t,a_t) = \max_{a_{t+1},a_{t+2},\ldots} \left(r(s_t,a_t) + \gamma r(s_{t+1},a_{t+1}) + \gamma^2 r(s_{t+1},a_{t+1}) + \ldots\right) \tag{15}$$

$$Q(s_t,a_t) = r(s_t,a_t) + \gamma \max_{a_{t+1},a_{t+2},\ldots} \left(r(s_{t+1},a_{t+1}) + \gamma^2 r(s_{t+1},a_{t+1}) + \ldots\right) \tag{16}$$

$$Q(s_t,a_t) = r(s_t,a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) \tag{17}$$

Um nun ein Q-Wert nach der Aktionswahl zu errechnen, müssen alle weiteren Belohnungen bekannt sein. Weil das unendlich viele Werte sind, kann der Q-Wert schrittweise mit Hilfe der rekursiven Form angepasst werden. Dabei ist γ der Diskontierungsparamter. Dieser bewirkt, dass der aktuelle Q-Wert höher gewichtet wird, als die in Zukunft liegenden Werte. Ist $\gamma = 1$, sind alle gleich gewichtet. Liegt der Wert bei 0, dann wird nur die aktuelle Belohnung betrachtet und immer die Aktion mit dem maximalen Erfolg gewählt. Für einen möglichst vorausschauenden Agenten sollte ein hoher Wert genutzt werden, weil damit die zukünftigen Aktionen mit berücksichtigt werden. Dazu müssen alle Zustandsübergänge nur oft genug besucht werden. Es wird mit dem schrittweisen Vorgehen also keine optimale Funktion, sondern nur eine (gute) Näherung erzielt. Zum Aktualisieren eines Q-Werts muss eine Transition

$$(s_t, a_t, r_{t+1}, s_{t+1}) \tag{18}$$

aufgezeichnet werden.

Im Gegensatz dazu, nutzt die SARSA-Methode die Transition

$$(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}) \tag{19}$$

mit der Aktualisierungsformel

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right]. \tag{20}$$

Damit der Q-Wert nun berechnet werden kann, wird noch die dazugehörige Aktion a_{t+1} benötigt. Diese Aktion ist die nächste auszuführende Aktion. Sie wird mit Hilfe einer bestimmten Strategie ermittelt. Die Gleichung 20 ist für den nicht-deterministischen Fall. Mit der Ergänzung der Gleichung wird die Aktualisierung stabilisiert. Tritt bei dem gleichen Zustand mit der gleichen Aktion eine unterschiedliche Belohnung auf, wird das mit dem letzten Summanden kompensiert. So wird sichergestellt, dass Unterschiedliche Ergebnisse sich gegenseitig nicht zu stark behindern. Ein weiterer Faktor der Gleichung 20 ist das α. Hiermit wird die Lernrate, also die Intensität des Lernens, beschrieben. Ein hoher Wert für α bewirkt, dass neue Erfahrung stark in die Bewertung und die Aktualisierung mit einfließen. Dabei können alte Erfahrungen leicht wieder vergessen werden, da die neuen Erkenntnisse genauso stark in die Beurteilung der Situation mit einfließen. Ist der Wert hingegen zu gering, wird das neu gewonnene Wissen evtl. zu sehr missachtet und hat kaum einen Anteil zum Lernerfolg. Laut [Wolter, 2008] sollte α hoch beginnen und mit voranschreitender Zeit immer geringer werden, jedoch niemals 0 erreichen. Wird $\alpha = 1$ gesetzt, ergibt sich Gleichung 17 und somit die Gleichung zum deterministischen Fall umgeformt. Bei der ursprünglichen Q-Iteration wird die Aktion nach der Maximalmethode ausgesucht. Dahingegen wird der Q-Wert ohne Wissen des nächsten Zustandes aktualisiert. Der Vorteil der SARSA-Methode ist, dass der Q-Wert mit Hilfe der gleichen Strategie aktualisiert wird, wie die nächste Aktion ausgesucht wird. Weil die SARSA-Methode Schritt für Schritt lernt, wird eine für den Moment gute Aktion, die sich später als ungünstig herausstellt, voraussichtlich nicht gewählt.

3.5 Exploration und Exploitation

Wie schon in der Einleitung bemerkt, kann jeder Mensch und Agent nur durch Erfahrung lernen. Es entsteht daher die Frage, wie der Agent diese Erfahrung sammeln kann. Zu diesem Stichwort gibt es die Begriffe Exploration und Exploitation. Exploration bedeutet so viel wie Erkundung, wohingegen Exploitation das Ausnutzen einer gewissen Aktion anhaftet. Der Agent muss also explorieren, erkunden welche Aktionen er ausführen kann und welche er wohl unterlassen sollte. Bei der Exploitation nutzt der Agent das bisher gelernte aus und erhält dadurch eine höhere Belohnung. Jedoch ist und bleibt es dem Agenten unbekannt, ob es eine bessere Aktion geben könnte. Dementsprechend muss bei der Modellierung darauf

geachtet werden, dass der Agent zu Beginn, wenn kein Wissen vorhanden ist, viel erkunden kann. Im späteren Verlauf sollte er jedoch sich auf das Gelernte verlassen können und nur noch wenig erkunden. Dennoch darf die Erkundung, bei vielen möglichen Zuständen, bis zuletzt nicht aufgegeben werden. Für den vorliegenden Fall gibt es meherere Möglichkeiten wie die Exploration aussehen kann. Zum einen kann versucht werden, dass der Agent zuerst mit vollständig zufälligen Aktionen beginnt zu lernen und dann von Runde zu Runde mehr das Gelernte ausnutzt. Es muss dem Agenten bis zum Ende jedoch die Möglichkeit gegeben werden neue Aktionen auszuprobieren, um unbekannte Zustände zu finden. Eine andere Methode benötigt Vorwissen. Ist durch Berechnung oder vorherige Erfahrungen bekannt, wie schnell der Agent mindestens fahren kann, kann die Möglichkeit der Erkundung so weit eingeschränkt werden, dass nur leichte Abweichungen von dem Vorwissen erlaubt sind. Dadurch kann der Agent die bereits gute Fahrt verbessern. Jedoch wird es damit schwer möglich sein, dass der Agent eine ganz andere Lösung findet, weil sein Aktionsraum für die Exploration beschränkt ist.

4 Implementierung

Die vorhergehenden Überlegungen müssen nun sinnvoll implementiert werden. Als Programmiersprache wird C++ gewählt, weil die TORCS-Schnittstelle in C/C++ vorliegt und die objektorientierte Programmierung die Implementierung erleichtert. Das Programm wird dadurch übersichtlicher und die Module lassen sich leichter voneinander abgrenzen. So kann das Verfahren zum Speichern der Q-Werte als Objekt modelliert und bei Bedarf leicht getauscht werden. Zuerst beschreibt das Kapitel die genutzten Komponenten und verschafft einen ersten Überblick über das Programm. Danach folgt eine detaillierte Beschreibung der Implementierung mit Hilfe eines Klassendiagrammes. Zuletzt werden noch entscheidende Feinheiten und Probleme der Implementierung hervorgehoben.

4.1 Komponentenübersicht

Das Programm teilt sich in zwei Komponenten auf. Einmal die Komponente, die das Fahrzeug steuert und die Komponente, die für das Lernen verantwortlich ist. Abbildung 6 zeigt diese Aufteilung. Weiterhin ist dort die Komponente TORCS abgebildet. TORCS nutzt die vom Driver, also den Fahrer, bereitgestellte Schnittstelle und tauscht darüber die nötigen Information zur Simulation aus. Der Fahrer soll also die geforderten Funktionen zur Initiali-

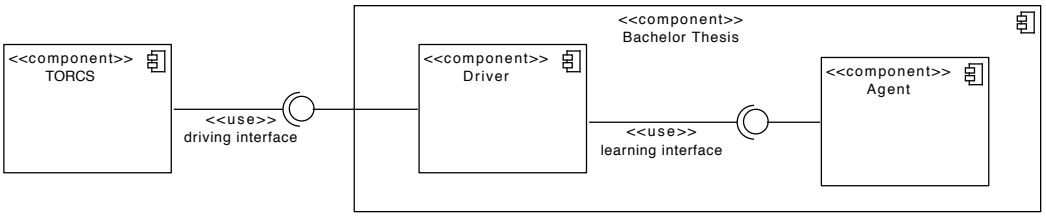

Abbildung 6: UML-Komponentendiagramm zur Darstellung des Gesamtsystems. Dabei stellt der Driver die Schnittstelle für TORCS bereit.

sierung, zum regelmäßigen Abfragen des Zustands und zum Herunterfahren des externen Programms (also der implementierten Arbeit) bereitstellen. Weiterhin ist es die Aufgabe dieser Komponente die Spur zu halten und immer möglichst in der Mitte zu fahren. Dazu wird der einfache Spurhaltealgorithmus aus [Wymann] genutzt. Dieser stellt die Reifen zuerst par-

allel zur Strecke ein und korrigiert diesen Winkel, falls sich das Fahrzeug nicht in der Mitte befindet. Weiterhin werden die einfachen Funktionen zur Geschwindigkeitsberechnung und Pedalstellung benutzt, um Vergleichswerte zu erhalten. Diese einfache Bremspunktberechnung nutzt die Formel 7, um zu wissen wie weit vor der Kurve gebremst werden muss.

Alle Werte der Umgebung sollen hier gekapselt und sinnvoll aufbereitet werden. Die Fahrer-Komponente nutzt dann den lernenden Agenten zur Bestimmung der Pedalstellung. Der Agent kann somit bestimmen, welche Aktion ausgeführt werden. Über die verfügbaren Informationen wird der Agent selbstständig entscheiden können, welche Aktion die Beste ist, um das gegebene Ziel zu erreichen.

4.2 Klassendiagramm

Das Klassendiagramm aus Abbildung 7 zeigt die nötigen Klassen mit den zugehörigen Attributen und Methoden. Hier sind auch die Helferklassen mit aufgezeigt. Damit das Programm überhaupt fahren und steuern kann wird die Klasse `Driver` benötigt, die zur Darstellung der nötigen Umgebungsvariablen die Klasse `Car` und `Polynomial` nutzt. Zum Lernen wird die Klasse `Agent` genutzt. Dieses Objekt wird zur Laufzeit vom Driver mit den notwendigen Informationen versorgt und versucht dann mit Lernverfahren das Ziel schnell zu fahren zu erreichen. Der Agent nutzt die Klasse `RBF` (kurz für radiale Basisfunktion) zur Speicherung der Q-Werte.

Driver

Ganz oben steht die Klasse `Driver`. Diese Klasse repräsentiert die Komponente Driver aus Abbildung 6 und ist das Bindeglied zwischen TORCS und der Lernumgebung. Die Methoden `initTrack()`, `newRace()`, `drive()`, `pitCommand()` und `endRace()` werden von TORCS eingefordert und daher vom `Driver` bereit gestellt. Dabei ist die Methode `drive()` sehr wichtig. TORCS ruft diese Funktion in jedem Zeitschritt einmal auf und gibt dem Programm damit die Möglichkeit die Sensoren auszuwerten und zu agieren. Tritt während zwei Zeitschritten ein wichtiges Ereignis auf, kann TORCS nicht durch Interrupts

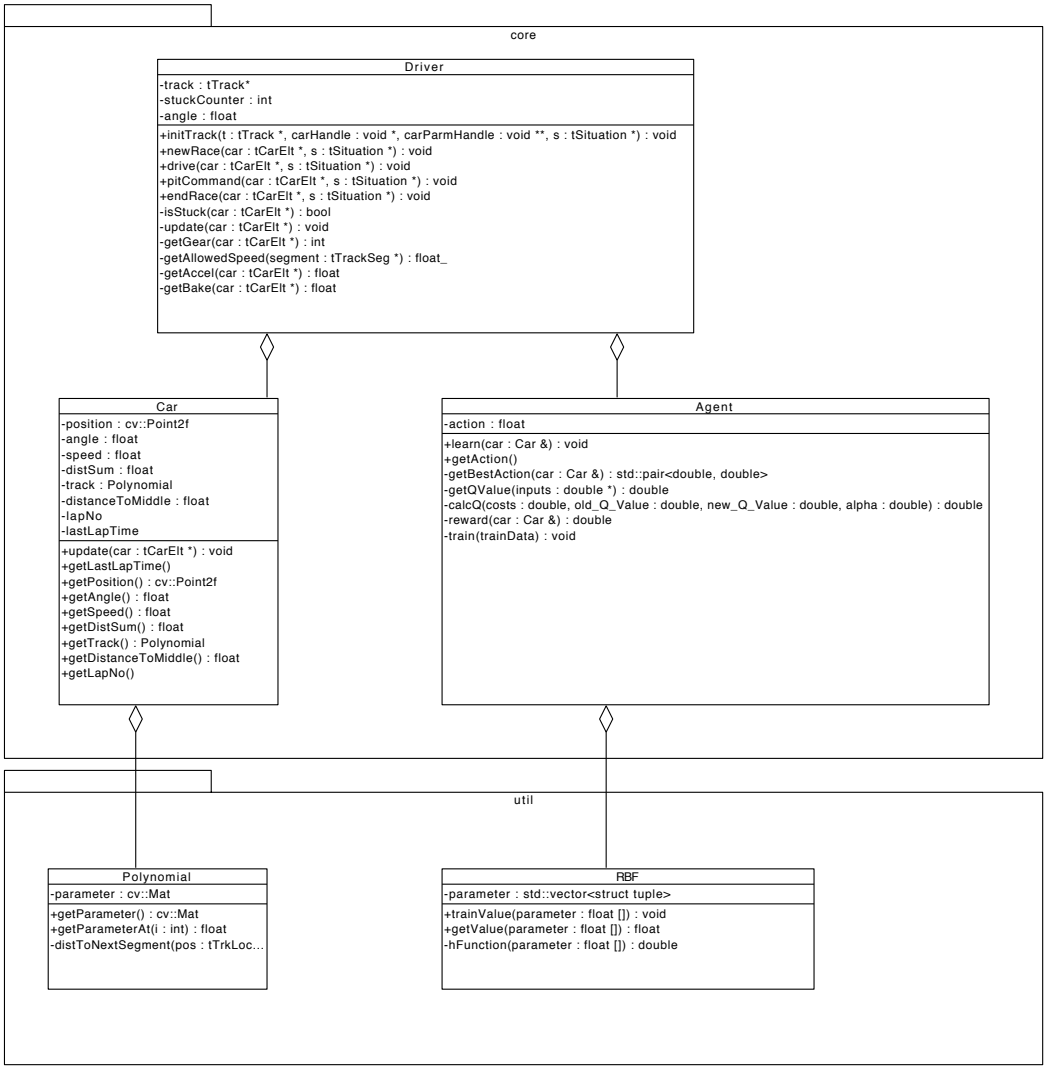

Abbildung 7: UML-Klassendiagramm zur Darstellung des Gesamtsystems. Die Komponenten Driver und Agent aus dem Komponentendiagramm werden hier genauer beschrieben.

o.ä. unterbrochen werden. Diese Art des Schedulings ist dem Scheduling in der FAUST-Architektur sehr ähnlich. Weiterhin wird diese Funktion mit einem 50Hz Takt aufgerufen. Das erfordert bei begrenzter Rechenkapazität eine zügige Abarbeitung des Lernenverfahrens. In der Simulation ist das Überschreiten der verfügbaren Rechenzeit jedoch kein Problem. Das Fahrzeug fährt dann trotzdem weiter und nur die Anzeige verlangsamt sich und beginnt zu ruckeln. Die Methoden `initTrack(), newRace()` und `endRace()` werden benötigt um das Programm zu starten, zu initialisieren und wieder zu beenden. Soll das Fahrzeug in die Box zum Auftanken oder zum Reparieren fahren, wird die Methode `pitCommand()` aufgerufen. Für diese Arbeit wird das Schadensmodell und der Benzinverbrauch deaktiviert. Daher wird diese Funktion nicht genutzt und ist nur vollständigkeitshalber implementiert. Es kann vorkommen, dass das Fahrzeug in der Simulation von der Strecke abkommt und mit einer Vorwärtsfahrt nicht auf die Strecke zurückfindet. Weil der Agent nur für das Vorwärts fahren die Aktionen bestimmen kann, muss ein Mechanismus vorhanden sein, der das Fahrzeug aus dieser Situation befreien kann. Dazu prüft die Methode `isStuck()` ob das Fahrzeug sich bewegt und ob es eventuell in die falsche Fahrtrichtung fährt. Sollte einer der Fälle eintreten, dass ein Eingriff von Nöten ist, greift der `Driver` in der Methode `drive()` ein und übernimmt kurzzeitig die Kontrolle. Dieser Fall wird vom Agenten sofort bemerkt und stark bestraft. Weil es kein Automatikgetriebe in TORCS gibt, muss die Schaltung auch vom `Driver` übernommen werden. Der optimale Gang wird anhand der Drehzahl berechnet und direkt eingestellt. Zur weiteren Informationsdarstellung bzw. zum Lernen, hält der `Driver` ein Objekt der Klasse `Car` und `Agent` bereit.

Car

Damit die zahlreichen Informationen der Umgebung sinnvoll bereitgestellt werden können, kapselt die Klasse `Car` diese. Dabei wird mit Hilfe der Methode `update(tCarElt* car)` dem Objekt die aktuelle Struktur für das Fahrzeug von TORCS übergeben. Dann werden die benötigten Informationen zur Lage des Fahrzeugs, Geschwindigkeit etc. in die jeweiligen Attribute geschrieben. Die Streckendarstellung wird, wie in Kapitel 3.3.2 beschrieben, mit Hilfe der Klasse `Polynomial` modelliert. Sonst stellt die Klasse weitere Getter zur Verfügung, um alle Attribute abzurufen.

Agent

Die Klasse `Agent` bildet die andere Komponente aus der Abbildung 6 ab. Hier wird mit Hilfe der Methode `learn(Car& car)` der Lernvorgang voran getrieben. In dieser Funktion wertet der Agent den aktuellen Zustand mit Hilfe des `Car`-Objekts aus und ruft die privaten Helfermethoden auf, um die in Kapitel 3.4 vorgestellten Gleichungen umzusetzen. Der Algorithmus ist in Algorithmus 1 in Pseudocode skizziert.

Algorithmus 1 Methode learn(Car& car)

```
1: bestAction = getBestAction()
2: newQ = getQValue(newSituation,bestAction)
3: updatedQ = calcQ(reward(),oldQ,newQ)
4: train(oldSituation,updatedQval)
5: oldQ = newQ
6: oldSituation=newSituation
```

Die Methode `learn()` nutzt alle privaten Methoden der Klasse `Agent`, um den Lernvorgang abzubilden. Zuerst wird in Zeile 1 die auszuführende Aktion gemäß einer bestimmten Strategie gewählt. Die Strategie ist dadurch leicht austauschbar. Danach muss der aktuelle Q-Wert, also $Q(s_{t+1}, a_{t+1})$, errechnet werden. Weil nicht in die Zukunft geschaut werden kann, wird der Q-Wert immer einen Zeitschritt verzögert aktualisiert. Deswegen wird der aktuelle Q-Wert aus Zeile 2 in Zeile 5 zum „alten“ Q-Wert. Das gleiche wird mit der Situation in Zeile 6 getan. Zeile 3 errechnet den Q-Wert des vorherigen Zeitschritts, also $Q(s_t, a_t)$, und teilt in der darauf folgenden Anweisung der jeweiligen Speicherungsart diesen Wert mit (siehe Kapitel 4.3.1). Damit der Agent möglichst unterschiedliche Aktionen und damit unterschiedliche Zustände erfährt, muss mit einer bestimmten Wahrscheinlichkeit eine andere Aktion als die momentan optimal erscheinende Aktion ausgewählt werden. Mehr dazu im Kapitel 3.5 „Exploration und Exploitation“.

4.3 Besonderheiten

Die Implementation gestaltete sich in einigen Bereichen etwas schwieriger als angenommen. So war es von Bedeutung, dass eine Möglichkeit gefunden wird, die in C++ schnell und einfach die vielen Q-Werte zuverlässig speichert. Weiterhin muss beachtet werden, dass auf einer vergleichsweise kurzen Strecke mit durchschnittlichen Rundenzeiten von 45 Sekunden insgesamt $45s * 50Hz = 2250$ Zeitschritte und dem entsprechend viele Daten anfallen. Versuche zeigten auch, dass TORCS das Rennen beendet, wenn das Fahrzeug zu langsam ist. Bei Strecken mit Rundenzeiten von ca. 100 Sekunden beendet TORCS das Rennen, wenn in einer Runde diese nicht innerhalb von ca. 3500 Sekunden beendet wird. Das Ende äußert sich so, dass das Fahrzeug plötzlich schwebt und neben die Strecke gesetzt wird. Um diese Anforderungen und die Datenmenge zu bewältigen und die richtigen Schlüsse daraus zu ziehen müssen diverse Annahmen und Anpassungen gemacht werden, die im Folgenden erläutert werden.

4.3.1 Speicherung der Q-Werte

Zur Speicherung der Q-Werte gibt es mehrere Möglichkeiten. Eine dieser Möglichkeiten ist, ein künstliches neuronales Netz zu verwenden. Ein neuronales Netz bildet, nachdem es trainiert wurde, eine Approximation einer Funktion ab. Der Vorteil von neuronalen Netze ist, dass sie sehr gut generalisieren. So wird sich die Belohnung auf gerader Strecke kaum unterscheiden, wenn das Fahrzeug 200 km/h oder 220 km/h fährt. Wenn die Geschwindigkeit ähnlich ist, werden sich dann auch die Zustände nur minimal unterscheiden. Wegen der Generalisierung wird dann bei beiden Zuständen ein sehr ähnliches Ergebnis vom neuronalen Netz zurückgegeben. Damit kann schon früh ein Erfolg erzielt werden, da wegen der Generalisierung nicht mehr alle möglichen Zustände besucht werden müssen. Weil das SARSA-Verfahren dauerhaft die Q-Werte aktualisiert, wird ein inkrementelles Lernverfahren genutzt. [Wilson und Martinez, 2003] behauptet sogar, dass das inkrementelle Lernverfahren schneller zum Ziel kommt als Batch-Lernverfahren. Weiterhin gibt es für neuronale Netze viele frei verfügbaren Bibliotheken für C++. Da ist z. B. die Shark machine learning library [Fischer u. a.], FANN (Fast Artificial Neural Network) [Nissen] und Lightweight Neural

Network++ [Masetti u. a., 2004]. Alle genannten Bibliotheken wurden ausprobiert und es ergaben sich dabei einige Probleme. Shark ist sehr umfangreich und der Einstieg gestaltete sich als schwierig. Weiterhin nutzt Shark die Boost-Bibliothek. Diese muss zusätzlich installiert und in den Buildvorgang eingebunden werden. FANN ist ist sehr einfach zu nutzen und zu integrieren und nutzt eigene Datentypen zur Darstellung der Ein- und Ausgänge. Diese sind durch einfache „includes“ zu ersetzen und werden nach Bedarf auf float oder double abgebildet. Weil FANN ursprünglich nur eine C Bibliothek war, gibt es mittlerweile für die C++ Anbindung einen Wrapper. Dieser kopiert bei jedem Aufruf die übergebenen Variablen und alloziert neuen Speicher, was die Bibliothek etwas ausbremst. Die Bibliothek Lightweight Neural Network++ war auch einfach zu integrieren, baut aber intern auf FANN auf und ist fast nur eine abgespeckte Version davon. Leider trat bei allen Bibliotheken das Problem der Skalierung auf. Mit Matlab wurde nach einigen Testrunden die Größenordnung der einzelnen Werte untersucht und damit die Skalierung gestaltet. Dabei fiel auf, dass einige Werte zwar einen großen Zahlenbereich haben, aber viele Werte sich nur in einem Teilbereich davon befinden. Diese Werte werden mit Skalierung und der Darstellung mit float/double evtl. nicht mehr unterscheidbar. Weiterhin konnte nicht sichergestellt werden, dass die untersuchten Größenordnungen wirklich eingehalten werden. Dabei wurde eine Schwäche der neuronalen Netze deutlich. Sind einige Eingangs- oder Ausgangswerte nicht in dem Bereich zwischen -1 und $+1$, haben die Bibliotheken große Probleme das neuronale Netz sicher zu trainieren.

Aus diesem Grund wurde ein anderes Verfahren, das der radialen Basisfunktionen, versucht. Hier wird mit mehreren Teilfunktionen $h_i(\vec{x})$, die jeweils radialsymetrisch sind, eine reelle Funktion approximiert. Die einzelnen Teilfunktionen bilden jeweils ein Trainingsdatum ab und erzeugen so eine umfangreiche Abbildung.

Abbildung 8 zeigt eine Teilfunktion einer radialen Basisfunktion vom Gauß-Typ. Um die Funktion nun zusammenzusetzen, müssen die Teilfunktion gewichtet mit einfließen.

$$y(\vec{x}) = \frac{\sum_i [y_i * h_i(\vec{x})]}{\sum_i h_i(\vec{x})} \tag{21}$$

Gleichung 21 zeigt wie diese endgültig approximierte Funktion berechnet wird. Die Klasse

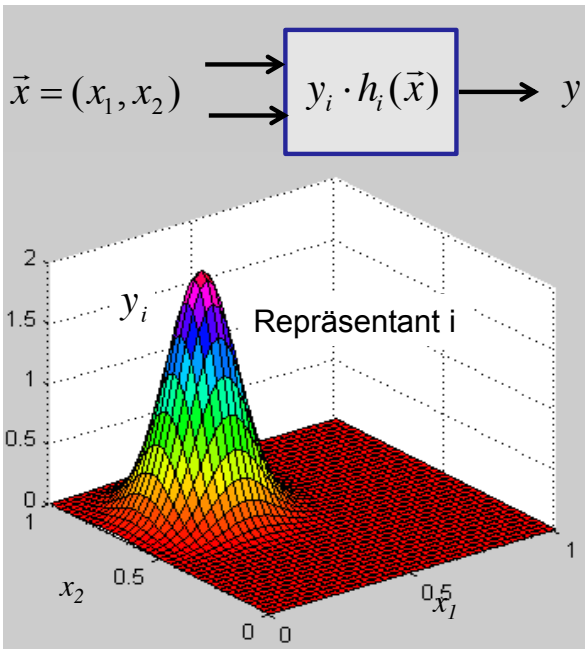

Abbildung 8: Teilfunktion einer radialen Basisfunktion vom Gauß-Typ. Dabei ist $h_i(\vec{x}) = e^{\frac{(\vec{x}-\vec{x}_i)^2}{2\sigma^2}}$. Quelle: [Meisel, 2012].

`RBF` aus Abbildung 7 setzt dieses Verfahren um. Der Nachteil dieser Variante ist offensichtlich. Es müssen alle Trainingsmuster gespeichert und bei jeder Berechnung eines Wertes mit einbezogen werden. Das kostet extrem viel Speicher und Zeit. Diese Parameter sind bei eingebetteten Systemen nicht immer vorhanden. Auf der Carolo-Cup Plattform ist dies auch ein Problem, weswegen dafür in Zukunft eine Lösung gesucht werden muss.

4.3.2 Datenmenge

Mit ca. 2250 Zeitschritten auf einer 45 Sekunden langen Runde fallen vergleichsweise viele Daten an. Diese müssen bei Benutzung der radialen Basisfunktion alle gespeichert werden. Wird dann versucht ein Ergebnis von der radialen Basisfunktion abzufragen, muss eine Berechnung mit allen diesen Daten angestellt werden. Auf schnellen Computern stellt diese Berechnung erst bei sehr vielen Daten eine Hürde dar. Bei eingebetteten Systemen ist diese Hürde jedoch schnell erreicht.

Weiterhin sind die Auswirkungen von einzelnen Aktionen innerhalb der Zeitschritte, die mit einen Abstand von 0,02 Sekunden ausgeführt werden, nicht detektierbar. Wird zum Beispiel eine Pedalstellung von +1, danach für einen Zeitschritt -1 und dann wieder +1 angelegt, wird der kurzzeitige Bremsvorgang nicht auffallen und nicht als schlecht festgestellt werden. Das Fahrzeug besitzt in der Realität und in der Simulation eine gewisse Trägheit, so dass Beschleunigungs- und Bremsvorgänge erst im Laufe der Zeit auffallen.

Aus diesen beiden Gründen wurde versucht, eine Aktion immer 20 Zeitschritte zu halten und das Ergebnis dann auszuwerten. Dadurch wird erhofft, dass ein konstanteres Ergebnis heraus kommt und der Agent nicht ständig versucht, abwechselnd zu beschleunigen oder zu bremsen. Des Weiteren reduziert sich damit auch die Datenmenge um den Faktor 20 auf ca. 112 Datenpaare pro Runde.

4.3.3 Verhindern des Rennabbruchs wegen zu langsamer Fahrt

Dass das Fahrzeug plötzlich zu Schweben beginnt und sich dann neben die Strecke bewegt, ist ein Zeichen dafür, dass es zu lange gedauert hat die aktuelle Runde zu beenden. Dieses Phänomen ist schwer im Internet wieder zu finden und eine Lösung, außer schnell genug zu fahren, wurde nicht gefunden um das Problem zu beheben.. Damit TORCS das Rennen nicht abbricht, muss die zufällige Verteilung der Aktionen in der Explorationsphase angepasst werden. Eine Gleichverteilung aller möglicher Aktionen ist nicht sinnvoll, weil die Hälfte der Aktionen das Fahrzeug zum Stehen bringen. Gelöst wurde dies, indem zuerst eine Aktion im Intervall $[0, +1]$ zufällig gewählt wird. Danach besteht eine Chance von 23%, dass die Aktion als Bremsung interpretiert wird. Mit dieser Lösung werden mehr Aktionen ausgeführt, die das Fahrzeug beschleunigen. Dadurch fährt der Agent schnell genug, um die Zeitgrenze nicht zu erreichen. Der Wert zur Auswahl ob gebremst wird, muss aber für unterschiedliche Strecken angepasst werden. Es hat sich gezeigt, dass 23% auf vielen kurzen Strecken ein guter Wert ist. Auf Strecken mit vielen Kurven sollte der Agent eher langsamer fahren, damit dieser die Strecke nicht zu häufig verlässt und deswegen viel Zeit verliert. Schnelle Kurse erfordern dagegen mehr beschleunigende Aktionen, da der Agent sonst zu langsam fahren würde und dadurch an die Zeitgrenze kommt.

5 Ergebnis

In diesem Kapitel folgt die Auswertung des genutzten und implementierten Verfahren. Das Ziel wurde schon früh, in Kapitel 1.2, festgelegt. Der Agent soll das Fahrzeug so schnell wie möglich um den Parcours steuern. Es stellt sich nun die Frage danach, wie „so schnell wie möglich“ definiert wird. Ein Ansatz ist die maximale Geschwindigkeit pro Abschnitt zu messen und daraus Schlüsse zu ziehen. Jedoch ist es eventuell nicht ratsam hohe Geschwindigkeiten zu fahren und dann stark zu bremsen, sondern eher eine geringere Geschwindigkeit und damit auch weniger starke Bremsungen. Jedes Mal wenn das Fahrzeug bremst, kann es ausbrechen bzw. der Agent kann die Kontrolle verlieren. Als einfacher, aussagekräftiger Wert eignet sich da die Rundenzeit. Die Rundenzeit beinhaltet, mit Wissen über die Streckenlänge, eine durchschnittliche Geschwindigkeit und ist sehr einfach vergleichbar. Für eine Referenz kann ein Programm mit Hilfe der Formel 7 den Bremspunkt und die Geschwindigkeit errechnen und über den Parcours fahren. Danach lässt sich leicht ein Vergleich anhand der Rundenzeit anstellen. Theoretisch könnte auch ein Mensch das Fahrzeug über die Strecke steuern. Auch diese Zeit ließe sich dann vergleichen. Leider reicht aber die Rundenzeit alleine nicht aus. Fährt das Fahrzeug nicht auf der Mittellinie, sondern die optimale Trajektorie, also so, dass die Kurven schneller angefahren werden können und das Fahrzeug früher beschleunigen kann, wird es viel schneller fahren können. Dabei wird aber das Ziel, auf der Mittellinie zu fahren, missachtet. Deswegen muss die Belohnung/Bestrafung pro Runde mit in die Überlegung, welche Zeit besser ist, einbezogen werden.

5.1 Test auf einer ovalen Strecke mit Schikanen

Der Agent wurde auf einer zum Mittelpunkt punktsymmetrischen Strecke mit Schikanen auf den langen Geraden getestet. Die Strecke ist in Abbildung 9 sikzziert. In der Mitte im unteren Bereich befindet sich der Startpunkt.

Ein großer Vorteil solcher symmetrischen Strecken ist, dass es alle Kurven häufiger gibt und der Agent somit schneller das Anfahren dieser Kurven erlernen kann. Dadurch wird der Lernvorgang erheblich beschleunigt. Diese Strecke hat die Besonderheit, dass der Agent

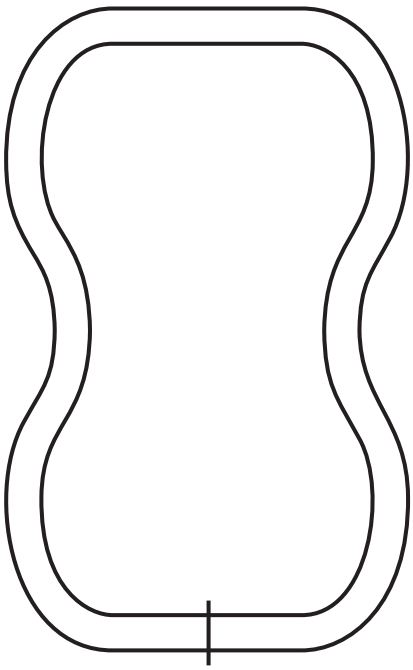

Abbildung 9: Ovale Teststrecke mit Schikanen auf den langen Geraden. Länge: 1621,73 m. Breite: 20 m.

nicht zu schnell durch die Schikane fahren darf. Falls er das tut, wird er bei einer der Kurven die Mittellinie verlassen müssen, weil das Fahrzeug ab einer bestimmten Geschwindigkeit ausbricht. Ein Versuch mit dem rechnenden Fahrer nach Gleichung 7 zeigt, dass dieser in der Schikane beschleunigt und dann neben der Mittellinie fährt. Dadurch ist die Geschwindigkeit in dieser Schikane zwar hoch, aber das Ziel auf der Mittellinie zu fahren wird verfehlt.

Abbildung 10 zeigt die Rundenzeiten und die summierten Belohnungen pro Runde. Auffällig ist, dass die Rundenzeiten zu Beginn sehr schwanken und sich nach ungefähr 30 Runden sehr ähneln. Dabei ist zu beobachten, dass der Agent kontinuierlich schneller wird. Das ist damit zu erklären, dass zu Beginn der Agent die Umwelt noch sehr stark erkundet. Zu einem späteren Zeitpunkt verlässt dieser sich dann mehr und mehr auf seine Erfahrung und kann daher gleichmäßige Zeiten fahren. Gegen Ende steigt die Bestrafung wieder ein wenig an, weil der Agent vermutlich in den Schikanen nicht perfekt mittig fahren kann. Eine weitere Auffälligkeit ist, dass die Bestrafung der Referenzfahrt genauso hoch ist, wie die des lernenden Agenten. Dabei ist jedoch die Rundenzeit sehr unterschiedlich. Das lässt sich damit erklären, dass der lernende Agent vermutlich langsamer in die Kurven hineinfährt und dadurch besser die Mittellinie halten kann, während der Referenzagent vermutlich beim Bremsen die Mitte

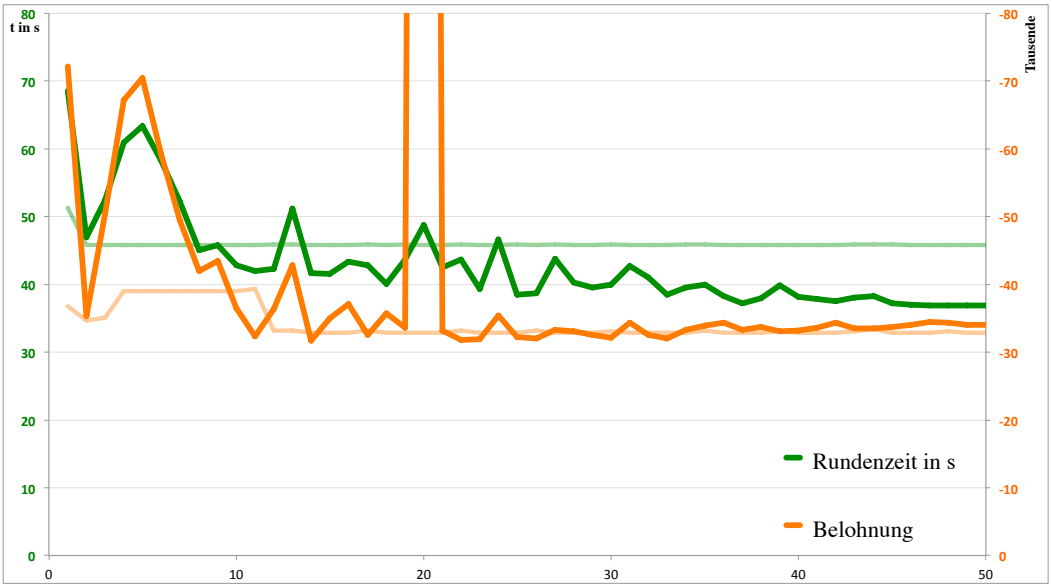

Abbildung 10: Diagramm über die Rundenzeit in grün (linke Skala) und der summierten Belohnung pro Runde in orangener Farbe (rechte Skala) auf der Strecke E-Track-5. Die jeweiligen hellen Kurven zeigen einen Referenzwert, gefahren mit Hilfe von Gleichung 7. Je höher die orange farbene Kurve ist, desto größer ist die Bestrafung.

nicht halten kann und bei anschließender Beschleunigung diese dann signifikant verlässt. Das erklärt auch warum die beste Rundenzeit nicht die geringste Bestrafung gemeinsam hat. Allgemein kann behauptet werden, dass der Agent erfolgreich gelernt hat, weil dieser die Rundenzeit bei fast gleicher Bestrafung um mehr als 8,9 Sekunden bei einer Referenzzeit von 45,83 Sekunden verbessert hat.

Ein weiteres Phänomen zeigt Grafik 11. Hier ist der Lernvorgang ähnlich abgelaufen wie in Diagramm 10, jedoch ist das Endergebnis ein Anderes. Es zeigt sich, dass der Agent die Bestrafung sehr weit, um ca. 2000 Punkte unter der Referenz, reduzieren konnte. Leider ist die Rundenzeit nicht so gut wie in dem anderen Versuch. Eine Erklärung ist, dass der Agent nur die Pedalstellung, nicht aber die Geschwindigkeit direkt, kontrolliert. Lernt der Agent nun, dass er das Pedal 50% betätigen soll, dann ist damit nicht der Gang erwähnt. Ein Fahrzeug wird natürlich im sechsten Gang bei 50% Pedalstellung wesentlich schneller fahren als im dritten Gang bei gleicher Pedalstellung. Nun wird das Fahrzeug im dritten Gang nicht so beschleunigen, dass der Agent es nötig hat, hoch zu schalten und fährt im dritten Gang mit geringerer Geschwindigkeit weiter. Dadurch wird dieser besser die Spur halten können und

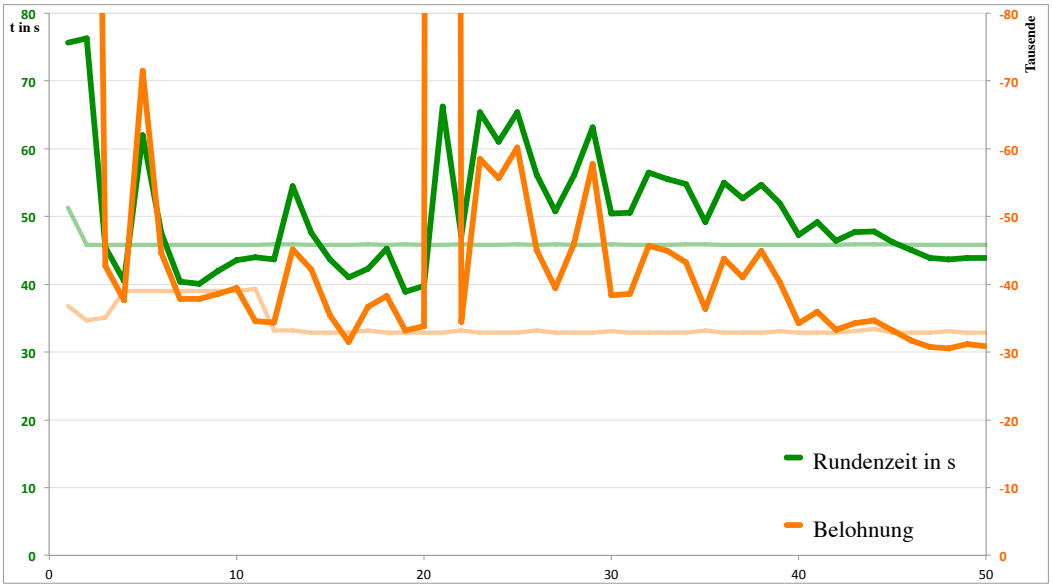

Abbildung 11: Diagramm über die Rundenzeit in grün (linke Skala) und der summierten Belohnung pro Runde in orangener Farbe (rechte Skala) auf der Strecke E-Track-5. Die jeweiligen hellen Kurven zeigen einen Referenzwert, gefahren mit Hilfe von Gleichung 7. Je höher die orange farbene Kurve ist, desto größer ist die Bestrafung. Hier ist der Lernvorgang auch erfolgreich gewesen, jedoch fällt hier die vergleichsweise langsamere Rundenzeit auf.

wird weniger bestraft. Diese These unterstützt auch die Abbildung 12. Dort fährt der Agent auf einer kurzen, symmetrischen Strecke mit jeweils zwei 90-Grad Kurven nach den langen Geraden. In Runde 44 fährt der Agent zu schnell und verursacht einen Unfall. Dabei verlässt er die Strecke und benötigt ein wenig Zeit bis dieser dann wieder normal fährt. Dafür wird er stark bestraft. Nun ist die Geschwindigkeit, die das Fahrzeug vorher hatte, verloren und der Agent fährt in einem geringeren Gang. Stellt er dort nun die gelernte Pedalstellung ein, fährt das Fahrzeug langsamer als erwartet.

5.2 Test auf rutschiger Strecke

Ein Versuch den Agenten auf einer sehr rutschigen Strecke lernen zu lassen, fand auf der Strecke „Dirt-5“ statt. Dirt-5 ist eine sandige Strecke mit leichten Kurven. Auf Grund der schlechten Bodenhaftung muss der Agent sehr frühzeitig bremsen und sanft beschleunigen, damit das Fahrzeug nicht ausbricht. Leider ist es hier sehr schwer die richtige Balance

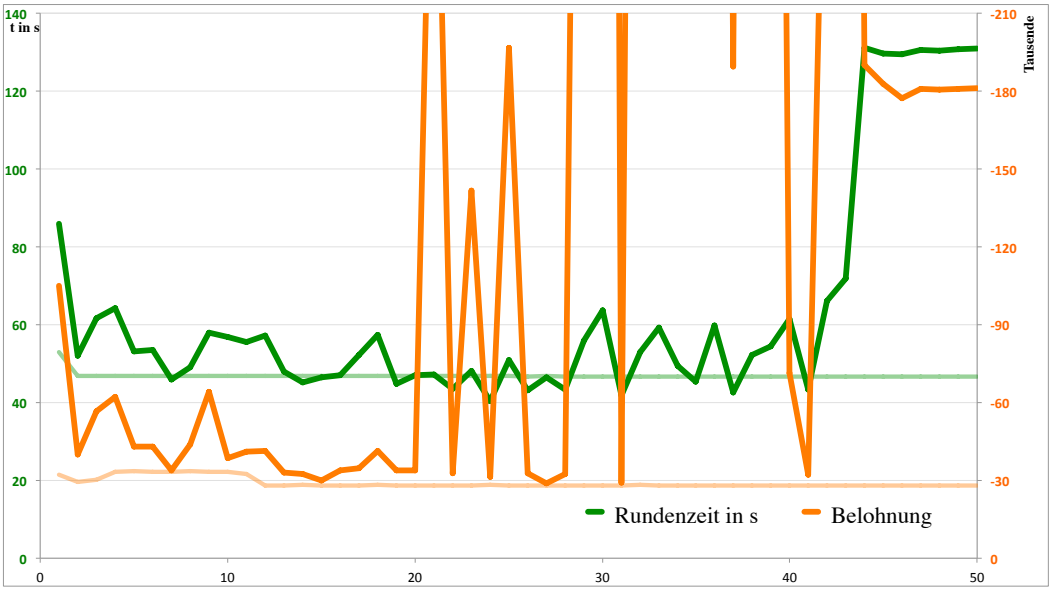

Abbildung 12: Diagramm über die Rundenzeit in grün (linke Skala) und der summierten Belohnung pro Runde in orangener Farbe (rechte Skala) auf der Strecke A-Speedway. Die jeweiligen hellen Kurven zeigen einen Referenzwert, gefahren mit Hilfe von Gleichung 7. Je höher die orange farbene Kurve ist, desto größer ist die Bestrafung. Hier ist der Lernvorgang auch erfolgreich gewesen, jedoch fällt hier die vergleichsweise langsamere Rundenzeit auf.

vom Bremsen und Beschleunigen (siehe dazu Kapitel 4.3.3) der zufälligen Aktion zu finden. Abbildung 13 zeigt das Ergebnis des Lernprozesses auf dieser Strecke. Es lässt sich ein Lernfortschritt nur schwer erahnen. Die vielen Unfälle machen es dem Agenten sehr schwer Erfolge zu messen. Sehr häufig gibt es hohe Bestrafungen für das Verlassen der Mittellinie und der Nutzung der Unstuck-Funktion der Klasse `Driver` aus Kapitel 4.2. Weiterhin ergibt sich auch hier das Problem des Trainings der Pedalstellung. Beobachtungen haben dort auch gezeigt, dass der Agent eine bestimmte Pedalstellung als erfolgreich erachtet, das Fahrzeug im zweiten Gang damit aber nicht so schnell wie in einigen Runden zuvor fährt. Ein weiteres Problem kann eine falsche Methode zur Exploration der Zustande gewesen sein. Auf sehr rutschigen Strecken ist die rein zufällige Aktionswahl wegen der ständigen Wechsel zwischen Bremsen und Beschleunigen nicht zu empfehlen. Hier bietet sich viel eher an, aus der Erfahrung der Referenz zu profitieren und diese zu verbessern.

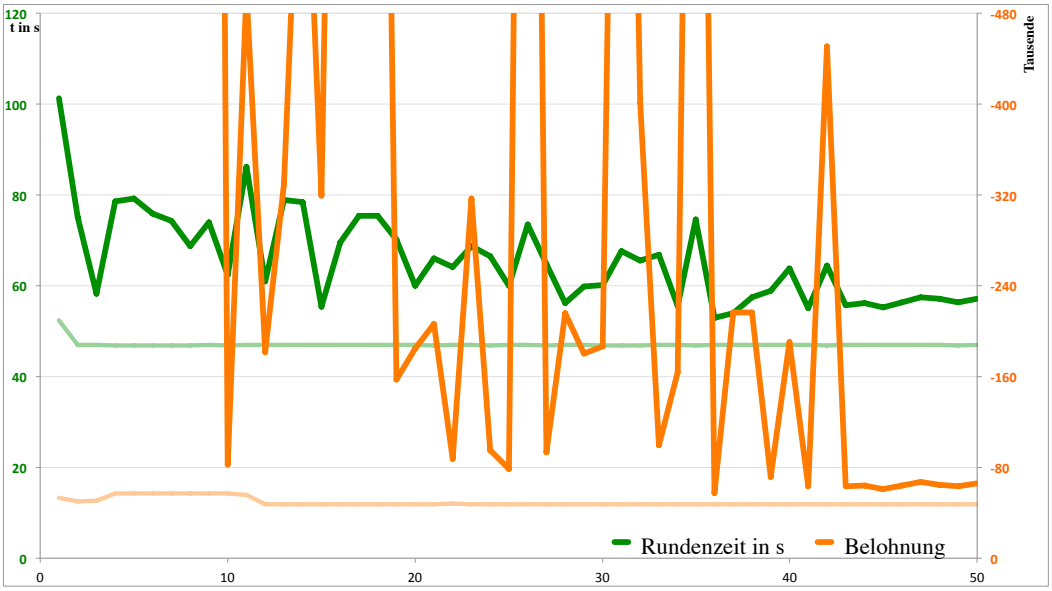

Abbildung 13: Diagramm über die Rundenzeit in grün (linke Skala) und der summierten Belohnung pro Runde in orangener Farbe (rechte Skala) auf der Strecke Dirt-5. Die jeweiligen hellen Kurven zeigen einen Referenzwert, gefahren mit Hilfe von Gleichung 7. Je höher die orange farbene Kurve ist, desto größer ist die Bestrafung. Hier ist der Lernvorgang auch erfolgreich gewesen, jedoch fällt hier die im Vergleich zur Referenz langsamere Rundenzeit und höhere Bestrafung auf. Der Agent hatte wegen der sehr rutschigen Strecke Probleme unfallfrei zu fahren.

5.3 Test unterschiedlicher Parameter

Der Versuch den Parameter α oder γ aus Gleichung 20 zu ändern, Anpassungen bei den zufälligen Aktionen oder Vorwissen über die Referenzzeit mit einzubringen, war wenig erfolgreich. Geringe Anpassungen dieser Parameter hatten keinen erkennbaren Einfluss auf die Lerngeschwindigkeit und das Ergebnis. Vermutlich kann die Art der Exploration eine große Änderung bewirken, aber angesichts des positiven Ergebnisses ist das für das Simulationsergebnis nicht zwingend nötig. Wird der Algorithmus jedoch auf der Carolo-Cup-Plattform eingesetzt, wäre eine solche Änderung sinnvoll, damit das Fahrzeug nicht zu häufig die Strecke verlässt oder zu starken mechanischen Belastungen durch das Beschleunigen und Bremsen ausgesetzt wird.

5.4 Bestehende Probleme, Ausblick

In dieser Arbeit wurde in mehreren Kapiteln versucht ein Programm mit künstlicher Intelligenz zu schaffen, welches selbstständig lernen kann, auf einer Strecke so schnell wie möglich zu fahren. Zu Beginn wurde festgestellt, dass das Problem der Kurvenfahrten berechnet werden könnte. Jedoch ist es von Nöten, dass diese Berechnung viele Faktoren beinhalten, damit z.B. ein Modellfahrzeug schnell und präzise Fahren kann. Viele dieser notwendigen Variablen lassen sich nur schwer bestimmen. Aus diesem Grund ist es eine einfache und kostengünstige Methode dieses Problem mit einem Lernverfahren, speziell das des bewertendem Lernen, zu lösen. Im darauffolgenden Kapitel ist das genutzte Verfahren erläutert. Dabei wird auch genau auf die Umsetzung eingegangen. Die Zustände für das Problem setzen sich aus der vorliegenden Strecke, beschrieben mit Hilfe von Polynomen, der Geschwindigkeit und dem Abstand zur Mitte zusammen. Insgesamt hat der Zustand eine Dimension von 6. Als Aktion reicht es die Pedalstellung zu wählen. Diese liegt im Intervall $[-1, +1]$. Dabei stellen die negativen Werte eine Betätigung des Bremspedals dar. Während der Implementierung traten kleine Probleme auf, die jedoch alle gelöst werden konnten. Unter anderem waren Probleme bei der Nutzung von neuronalen Netzen vorhanden. Am Ende wurde dieses Problem gelöst, indem ein anderes Verfahren, nämlich das der radialen Basisfunktionen, genutzt wurde. Leider benötigt dieses Verfahren bei vielen, bzw. bei langen Runden, viel Rechenzeit und Speicher, so dass es für eingebettete Systeme nur mäßig brauchbar ist. Das Ergebnis zum Schluss zeigt, dass es gelungen ist, dem Agenten das schnelle Fahren lernen zu lassen. Der Agent fährt auf einem einfachen Rundkurs knapp 9 Sekunden schneller als die Referenzzeit, welche mit einer einfachen Berechnung des Bremspunktes aufgestellt wurde. Dennoch offenbarten sich dabei weitere Verbesserungen. Mit der Pedalstellung als einzige Aktion ist nicht sichergestellt welche Geschwindigkeit gefahren wird. Zur Pedalstellung gehört zusätzlich noch der aktuelle Gang, damit die Geschwindigkeit einschätzbar wird. Eine Möglichkeit wäre es, einen Tempomaten mit einer Sollgeschwindigkeit zu nutzen. Das vergrößert den Aktionsraum nicht in der Dimension. Die Geschwindigkeitskontrolle im Carolo-Cup Fahrzeug hat so einen Tempomaten. Dadurch wird dieses Problem dort nicht auftreten. Um auch auf sehr rutschigen Strecken einen sinnvollen Lernfortschritt zu erkennen, benötigt es eine starke Anpassung der Parameter. Wahrscheinlich muss das Expolorationsverhalten für solche Strecken geändert werden, weil die Referenzfahrt dort sehr gut und vorsichtig herangeht. Je-

doch sind solche sehr rutschigen Strecken nicht das Hauptziel. Soll der Agent Strecken mit wechselnden Bedingungen lernen, wird vermutlich kein Lernfortschritt zu sehen sein. Wenn mehrmals eine ähnliche Kurve, die jedes Mal aber ein sehr unterschiedlichen Reibungskoeffizienten aufweist, vorkommt, wird der lernende Algorithmus die Erfahrung der vorherigen Kurve immer wieder überschreiben. Die Zustandsbeschreibung, die nur eine grobe Beschreibung der Streckenführung und der aktuellen Position enthält, ist nicht ausführlich genug, um solche Bedingungen mit aufzunehmen. Beide Kurven sehen dann gleich aus und der Agent kann die Erfahrung nicht differenzieren.

Um den Titel der Arbeit noch einmal aufzugreifen, soll noch einmal darauf eingegangen werden, ob denn nun Bremspunkte gefunden werden. Weil das Ziel schon zu Beginn zu einer schnellen Fahrt geändert werden musste, werden keine Bremspunkte direkt ersichtlich. Wird das Verhalten des Agenten jedoch beobachtet, können Bremspunkte erkannt werden. An den Punkten, an den der Agent die Geschwindigkeit drosselt und der d_{Sum} Wert stark abnimmt wird gebremst. Da aber die Umsetzung des Ziels dem Agenten überlassen wird, kann es auch vorkommen, dass dieser lieber dauerhaft langsam fährt und eine Geschwindigkeit hält, anstatt häufig zu beschleunigen und zu bremsen.

Literatur

[Abdullahi und Lucas 2011] ABDULLAHI, Aisha A. ; LUCAS, Simon M.: Temporal difference learning with interpolated n-tuples: Initial results from a simulated car racing environment. In: *Computational Intelligence and Games (CIG), 2011 IEEE Conference on* IEEE (Veranst.), 2011, S. 321–328

[Büchler 2012] BÜCHLER, Dieter: *Optimale Trajektorien mit Reinforcement Learning*. 2012

[DIN8855 2012] DIN8855: *DIN ISO 8855:2012-04, Straßenfahrzeuge - Fahrzeugdynamik und Fahrverhalten - Begriffe*. asd: Deutsches Institut für Normung e. V., Berlin (Veranst.), 4 2012

[Ertel 2009] ERTEL, Wolfgang: *Grundkurs künstliche Intelligenz: eine praxisorientierte Einführung*. Springer DE, 2009

[FAUST-HAW-HH 2013] FAUST-HAW-HH: *Faust*. Internet www.faust.informatik.haw-hamburg.de. 7 2013

[Fischer u. a.] FISCHER, Asja ; GLASMACHERS, Tobias ; HANSEN, Kasper N. ; IGEL, Christian ; KRAUSE, Oswin ; LI, Bill ; TUAN, Trinh X. ; TUMA, Matthias ; VOSS, Thomas: *Shark*. Internet http://shark-project.sourceforge.net

[Harmon und Harmon 1997] HARMON, Mance E. ; HARMON, Stephanie S.: Reinforcement Learning: A Tutorial. / DTIC Document. 1997. – Forschungsbericht

[Hering u. a. 2012] HERING, E. ; MARTIN, R. ; STOHRER, M.: *Physik für Ingenieure*. Springer, 2012

[Huang u. a. 2005] HUANG, Bing-Qiang ; CAO, Guang-Yi ; GUO, Min: Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. In: *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on* Bd. 1 IEEE (Veranst.), 2005, S. 85–89

[Masetti u. a. 2004] MASETTI, Lorenzo ; CINTI, Luca ; ROSSUM, Peter van: *Lightweight Neural Network++*. Internet http://lwneuralnetplus.sourceforge.net. 5 2004

[Meisel 2012] MEISEL, Andreas: *Musterklassifikation und Neuronale Netze*. Vorlesung. 2012

[Nissen] NISSEN, Steffen: *FANN*. Internet http://leenissen.dk

[Noglik 2012] NOGLIK, Anastasia: *Effizientes Reinforcement-Learning für autonome Navigation*, Universität Duisburg-Essen, Fakultät für Ingenieurwissenschaften» Ingenieurwissenschaften-Campus Duisburg» Abteilung Informatik und Angewandte Kognitionswissenschaft» Intelligente Systeme, Dissertation, 2012

[Riedmiller u. a. 2007] RIEDMILLER, Martin ; MONTEMERLO, Michael ; DAHLKAMP, Hendrik: Learning to drive a real car in 20 minutes. In: *Frontiers in the Convergence of Bioscience and Information Technologies, 2007. FBIT 2007* IEEE (Veranst.), 2007, S. 645–650

[Russell u. a. 1995] RUSSELL, Stuart J. ; NORVIG, Peter ; CANNY, John F. ; MALIK, Jitendra M. ; EDWARDS, Douglas D.: *Artificial intelligence: a modern approach*. Bd. 74. Prentice hall Englewood Cliffs, 1995

[Stanley und Miikkulainen 1996] STANLEY, Kenneth O. ; MIIKKULAINEN, Risto: Efficient reinforcement learning through evolving neural network topologies. In: *Network (Phenotype)* 1 (1996), Nr. 2, S. 3

[Sutton und Barto 1998] SUTTON, Richard S. ; BARTO, Andrew G.: *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998

[Wilson und Martinez 2003] WILSON, D R. ; MARTINEZ, Tony R.: The general inefficiency of batch training for gradient descent learning. In: *Neural Networks* 16 (2003), Nr. 10, S. 1429–1451

[Wolter 2008] WOLTER, A.: *Reinforcement Learning in der Roboter-Navigation: Grundlagen, Methoden, Realisierung und Experimente*. VDM Publishing, Saarbrücken, 2008

[Wymann] WYMANN, Bernhard: *T.O.R.C.S. Manual installation and Robot tutorial*. URL www.berniw.org/aboutme/publications/torcs.pdf:

[Wymann und Espié 2013] WYMANN, Bernhard ; ESPIÉ, Eric: *TORCS*. Internet http://torcs.sf.net. 7 2013

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 20. Oktober 2013 Phillip Gesien