Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Ivo Nikolov

Maschinelles Lernen zur Optimierung einer autonomen Fahrspurführung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Ivo Nikolov

Maschinelles Lernen zur Optimierung einer autonomen Fahrspurführung

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik
am Department 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 14. November 2011

Ivo Nikolov

Thema der Masterarbeit
Maschinelles Lernen zur Optimierung einer autonomen Fahrspurführung

Stichworte
FAUST, autonomes Fahren, Spurführung, Pure Pursuit, Q-Learning, Neural Fitted Q Iteration

Kurzzusammenfassung
In dieser Arbeit wurde ein modellfreies Spurführungsverfahren entwickelt. Dieses Verfahren berücksichtigt die Fahrzeugkinematik, den Verlauf der Fahrspurgeometrie und die Aktionen während der Lenkungstotzeit, um die optimale Lenkungsaktion für den aktuellen Fahrzeugzustand zu bestimmen. Das prädiktive NFQ-Spurführungsverfahren setzt den Neural Fitted Q Iteration Algorithmus zum Trainieren eines neuronalen Netzes ein. Zur Ermittlung von der besten Lenkungsaktion bezüglich des Fahrzeugzustandes, liefert das trainierte neuronale Netz eine Approximation der Q-Funktion.

Ivo Nikolov

Title of the paper
Machine Learning for optimizing an autonomous lane tracking

Keywords
FAUST, autonomous driving, path tracking, Pure Pursuit, Q-learning, Neural Fitted Q Iteration

Abstract
In this work a model-free path tracking method was developed. The method considers the vehicle kinematics, the path geometry and the delayed control actions in order to determine the optimal control action. The predictive path tracking method applies the Neural Fitted Q Iteration algorithm for training a neural network. The trained neural network provides an approximation of the Q-function so that the best control action for the current vehicle state can be determined.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Im Forschungsprojekt FAUST aus dem Department Informatik der Hochschule für Angewandte Wissenschaften Hamburg werden Technologien für Fahrerassistenz- und Autonome Systeme entwickelt und entworfen [FAUST 2011]. Autonomes Fahren wurde auf Autobahnen und Landstraßen seit Anfang der neunziger Jahre in verschiedenen Projekten erforscht. Am Markt verfügbare Fahrerassistenzsysteme greifen bereits heute aktiv in die Fahrzeuglängs- und Querführung ein. Für die Zukunft ist die Automatisierung in der Fahrzeugführung, von der reinen Assistenz hin zu automatischen Fahrfunktionen, denkbar [Weiser u. a. 2009].

Volkswagen setzt auf innovativste Entwicklungsprozesse und Technologien, um die aktive Sicherheit, und damit das Vermeiden von Unfällen, zu optimieren. Ein Beispiel ist der selbstfahrender Golf GTI 53+1, der mit Pylonen markierte Strecken vollautomatisch und bei maximaler Leistung abfährt. Dieses Fahrzeug erreicht dabei ähnlich schnelle Rundenzeiten wie versierte Profis [Volkswagen 2006].

Die DARPA-Challenge, welche die Entwicklung autonomer Fahrzeuge fördert, fand im November 2007 zum dritten Mal statt. Diesmal war das Ziel der Roboterfahrzeuge eine 96 km lange Route in bebautem Gebiet einer verlassenen Kaserne zu bewältigen. Insgesamt sechs Teams haben die Route erfolgreich abgeschlossen. Das Fahrzeug Boss des Gewinnerteams Tartan Racing war mit einer durchschnittlichen Geschwindigkeit von ca. 23 km/h unterwegs [DARPA]. Die TU Braunschweig hat mit dem Team CarOLO erfolgreich teilgenommen und ist als eines der 11 Teams von 89 Bewerbern im Finale angetreten [Wille u. a. 2010].

1.1 Motivation

Der Hochschulwettbewerb Carolo-Cup, einer der Schwerpunkte innerhalb des Projektes FAUST, bietet Studententeams die Möglichkeit an, sich mit der Entwicklung und Umsetzung von autonomen Modellfahrzeugen auseinander zu setzen [Carolo-Cup]. Beim Wettbewerb müssen bestimmte Fahraufgaben autonom, möglichst schnell und fehlerfrei bewältigt werden und die erarbeiteten Konzepte in Präsentationen erläutert werden.

Laut [Carolo-Cup-Regelwerk 2011] sollen die Fahrzeuge bei der dynamischen Disziplin *Rundstrecke* autonom drei Minuten lang auf einem unbekannten Rundkurs so weit wie möglich fah-

ren. Bei der Straße handelt es sich um die Nachbildung einer Landstraße, die aus langen Geraden, schnellen Kurven, engen Serpentinen und Kreuzungen besteht. Die Straße ist konstant 820 mm breit und der minimale Kurvenradius beträgt 1000 mm (vgl. Abbildung 1.1). Das Verlassen der eigenen, rechten Fahrspur mit mehr als einem Rad wird mit fünf Strafmetern geahndet.

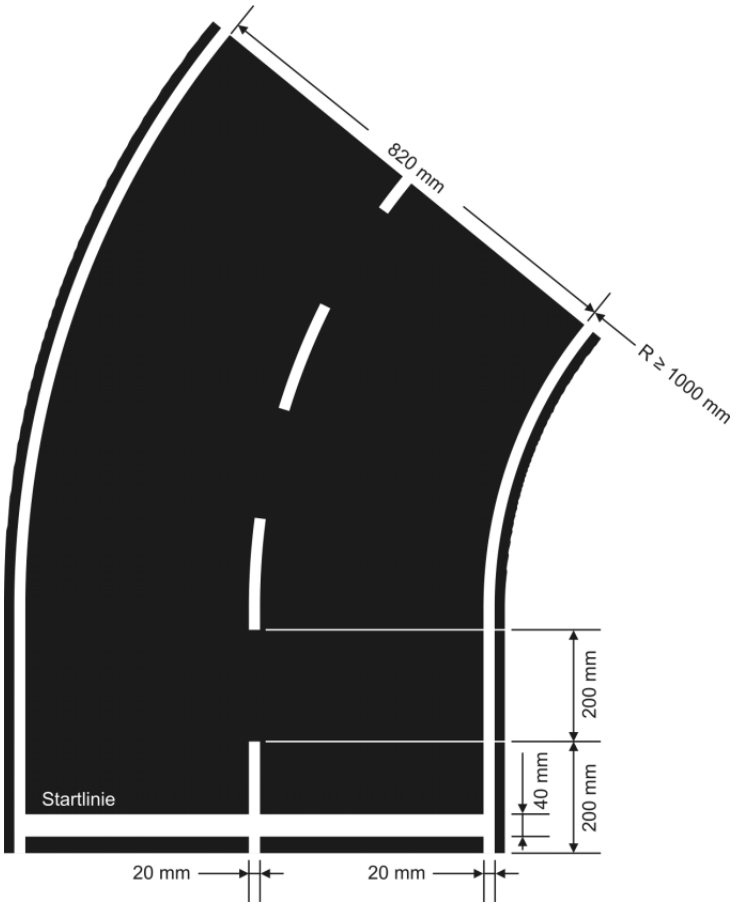


Abbildung 1.1: Fahrspurgeometrie im [Carolo-Cup-Regelwerk 2011]

Das im Carolo-Cup 2010 eingesetzte Verfahren zur Spurführung [Nikolov 2009] lokalisiert kontinuierlich mit Hilfe einer polynombasierten Spurerkennung [Jenning 2009] ein Ziel, dass das Fahrzeug verfolgen soll. Zur Berechnung des optimalen Lenkwinkels in Bezug auf das Ziel wird der Pure Pursuit Algorithmus [Coulter 1992] verwendet. Beim Wettbewerb konnte das Modellfahrzeug Onyx die Fahrspur bei ca. 1,5 m/s verfolgen. Ziel dieser Arbeit ist dieses Verfahren zu optimieren, um eine Fahrbahnverfolgung bei höheren Geschwindigkeiten zu erreichen.

1.2 Problemstellung

Die Verfolgung einer Sollspur wird in der Robotik als *Path Tracking* bezeichnet. Methoden zur Path Tracking haben die Aufgabe die Abweichung zur Sollspur zu minimieren und ein stabiles Lenkungsverhalten zu leisten, wodurch eine Verfolgung bei höheren Geschwindigkeiten erreichbar ist.

Viele Methoden basieren auf Zusammenhängen zwischen dem geometrischen Farhzeugmodell und der Sollspur. Zur Vereinfachung der Geometrie eines Fahrzeugs mit Ackermann-Lenkung wird häufig das geometrische Fahrradmodell verwendet (vgl. Abbildung 1.2).

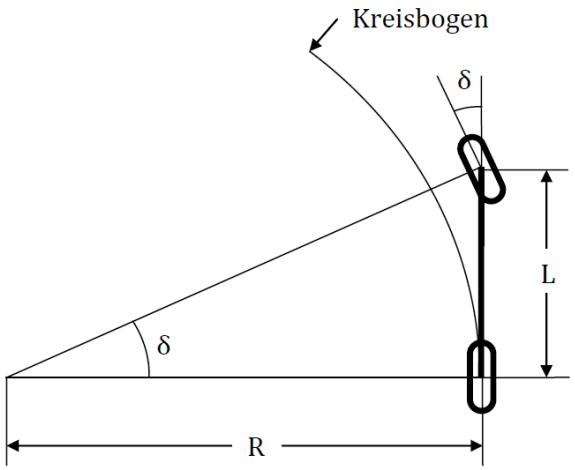


Abbildung 1.2: Geometrisches Fahrradmodell

Aus diesem Modell lässt sich der Zusammenhang ermitteln:

$$\tan(\delta) = \frac{L}{R}. \tag{1.1}$$

Daraus folgt, dass bei einem eingeschlagenen Lenkwinkel δ ein Fahrzeug mit Radstand L eine Kurve mit Radius R fährt. Die meisten geometrischen Path Tracker bestimmen den Lenkwinkel bezüglich eines Ziels g, welches in einer gewissen Entfernung D vor dem Fahrzeug liegt.

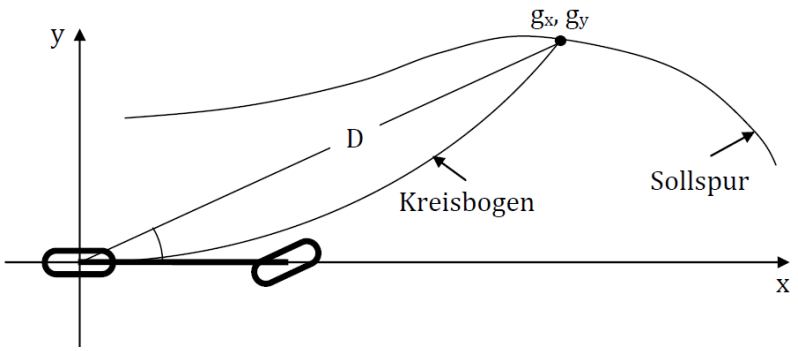


Abbildung 1.3: Pure Pursuit Geometrie

Der einfachste Algorithmus ist sicherlich Follow-the-carrot [Lundgren 2003]. Der Lenkungsstellwert y_{ftc} wird mittels einer linearen Funktion des Winkels zwischen Fahrzeugausrichtung und Ziel wie folgt berechnet:

$$y_{ftc} = K_p \arctan\left(\frac{g_y}{g_x}\right). \tag{1.2}$$

Der Pure Pursuit Algorithmus [Coulter 1992] basiert auf der Krümmung γ des Kreisbogens, welcher das Fahrzeug mit dem Ziel verbindet:

$$\gamma = \frac{2g_y}{D^2}. \tag{1.3}$$

Um eine Fahrt dieser Krümmung zu realisieren wird der Lenkwinkel mittels Gleichung 1.1 ermittelt, wobei $\gamma = 1/R$. Follow-the-carrot und Pure Pursuit sind beide robust, weil diese Algorithmen bei vielen unterschiedlichen Fahrmanövern einsetzbar sind.

Trotz der leichten Implementierung und Verständlichkeit haben Follow-the-carrot und Pure Pursuit einige Nachteile. Das Fahrzeug orientiert sich kontinuierlich an einem Ziel, welches in einer gewissen Entfernung vom Fahrzeug liegt. Somit wird die entsprechende Fahrspurgeometrie zwischen Fahrzeug und Ziel nur teilweise berücksichtigt. Zur Folge schneidet das Fahrzeug die Kurven. Die Entfernung zwischen Fahrzeug und Ziel ist für Präzision und Stabilität der Fahrbahnverfolgung entscheidend. Um eine präzise Fahrt zu erreichen, muss das Ziel nah am Fahrzeug liegen, damit die Fahrspurgeometrie optimal berücksichtigt wird. Das wiederum beeinträchtigt die Stabilität so erheblich, dass höhere Geschwindigkeiten nicht erreichbar sind, weil die Verzögerung des Lenkungssystems ausschlaggebende Schwingungen beim Fahren verursacht. Um diese Schwingungen zu reduzieren, soll die Entfernung erhöht werden. Auf diese Weise wird Stabilität erreicht, was die Fahrbahnverfolgung bei höheren Geschwindigkeiten ermöglicht. Das verschlechtert gleichzeitig aber die Präzision der Fahrt und ab einer

bestimmten Entfernung kann der Algorithmus nicht mehr leisten, dass das Fahrzeug dauerhaft auf der zu verfolgenden Fahrbahn bleibt. Da sowohl Präzision als auch Stabilität wichtig sind, muss eine passende Entfernung hinsichtlich der gestellten Anforderungen der Fahrmanöver bestimmt werden. Häufig wird die Entfernung bezüglich der aktuellen Geschwindigkeit ermittelt.

Die Stabilität von geometrischen Methoden wird außerdem stark von der Lenkungstotzeit T_t gestört. T_t beträgt die Zeit von der Ausführung eines Steuerbefehls bis zu einer Lenkbewegung hinsichtlich dieses Befehls. Zur Analyse wurde das Verhalten von Pure Pursuit in einer Simulationsumgebung bei unterschiedlichen Totzeiten untersucht. Die Ergebnisse sind im Abschnitt 6.2 vorgestellt. Im Vergleich zu den Fahrten bei $T_t = 0,24\,s$ sind fast doppelt so schnellen Geschwindigkeiten bei der gefahrenen Teststrecke erreichbar, wenn die Totzeit im Mikrosekundenbereich ist. Einige Verfahren [Murphy 1994; Kelly 1994] nutzen ein Systemmodell, um die Auswirkung der Totzeit zu modellieren [Rankin 1997].

Weitere Methoden zum Path Tracking beruhen auf einem kinematischen oder dynamischen Fahrzeugmodell. Die kinematischen Methoden befassen sich mit den Fahrzeugbewegungen ohne Ursachenbetrachtung der Kräfte. In [Luca u. a. 1997] wird ein solcher Path Tracker vorgestellt. Wenn auf dem Fahrzeug wirkende Kräfte betrachtet werden, spricht man von dynamischen Methoden. Das Roboterfahrzeug Stanley, welches DARPA Grand Challenge 2005 gewonnen hat, profitierte von einer dynamischen Methode. Es handelt sich dabei um die so genannte Stanley-Methode [Hoffmann u. a. 2007].

In [Snider 2009] wurden geometrische, kinematische und dynamische Methoden verglichen. In der Simulationsumgebung CarSim wurden Pure Pursuit, eine vereinfachte kinematische Version der Stanley-Methode und eine Linear Quadratic Regulator (LQR) basierte Methode implementiert und analysiert. Pure Pursuit hat die beste Robustheit geleistet, aber aufgrund der beschriebenen Nachteile schwingt das Fahrzeug bei höheren Geschwindigkeiten. Die Stanley-Methode erreicht eine stabilere Fahrt, aber liefert nicht die Robustheit von Pure Pursuit, weil dieser Path Tracker nur die aktuelle Sollspurgeometrie und keine vor dem Fahrzeug liegende Ziele berücksichtigt, was zu Überschwingungen führen kann. Die LQR basierte Methode betrachtet den dynamischen Fahrzeugzustand, aber erzielt nur auf geraden Strecken die besten Ergebnisse.

Folglich ist ein hybrides Verfahren zum Path Tracking vorteilhaft. Dies gilt für das in dieser Arbeit vorgeschlagene Verfahren, welches den kinematischen Fahrzeugzustand und gleichzeitig vor dem Fahrzeug liegende Ziele berücksichtigt.

1.3 Zielsetzung

Folgende Zielvereinbarungen werden für das Konzept des Spurführungsverfahrens getroffen:

- Das Verfahren soll zusammen mit einem polynombasierten Spurerkennungssystem eine autonome Fahrbahnverfolgung realisieren.
- Das Verfahren soll den kinematischen Fahrzeugzustand, den Verlauf der Fahrspurgeometrie und die Lenkungstotzeit berücksichtigen.
- Das Verfahren soll mittels maschineller Lernverfahren die optimale Lenkungsaktion für den aktuellen Fahrzeugzustand bestimmen können.
- Das Verfahren soll sowohl an einem im FAUST Projekt entwickelten Modellfahrzeug als auch in einer Simulationsumgebung verifiziert werden.

1.4 Gliederung der Arbeit

Nach dieser Einführung werden im Kapitel 2 die Grundlagen von Q-Learning vorgestellt. Zusätzlich wird der Fitted Q Iteration (FQI) Algorithmus beschrieben, welcher eine Approximation der Q-Funktion für kontinuierliche Zustands- und Aktionsräume liefert. Dazu werden die Approximationsmethoden künstliche neuronale Netze und Support-Vektor-Regression erläutert.

Kapitel 3 geht auf verwandte Arbeiten ein. Hier werden zwei NFQ-Anwendungen beschrieben. Wenn FQI künstliche neuronale Netze zur Approximation verwendet, spricht man von Neural Fitted Q Iteration (NFQ).

In Kapitel 4 wird das in dieser Arbeit entwickelte prädiktive NFQ-Spurführungsverfahren vorgestellt.

Kapitel 5 beschreibt detailliert die Testumgebung und die Implementierung des prädiktiven NFQ-Spurführungsverfahrens.

In Kapitel 6 wird die Approximationsleistung von künstlichen neuronalen Netzen und Support-Vektor-Regression verglichen. Anschließend folgt eine Auswertung des Spurführungsverfahrens.

Kapitel 7 fasst die Inhalte dieser Arbeit zusammen und in Kapitel 8 werden weitere Entwicklungsschritte zur Optimierung des Verfahrens vorgestellt.

2 Q-Learning für kontinuierliche Zustands- und Aktionsräume

Das in dieser Arbeit vorgeschlage Verfahren zur Spurführung basiert auf dem Algorithmus Fitted Q Iteration (FQI). FQI liefert eine Approximation der Q-Funktion, sodass Q-Learning auch bei Systemen mit kontinuierlichen Zuständen und Aktionen angewandt werden kann (vgl. Abschnitt 2.2). In Abschnitt 2.3 werden Grundlagen von künstlichen neuronalen Netzen und Support-Vektor-Maschinen beschrieben. Diese Methoden können zur Approximation der Q-Funktion in FQI eingesetzt werden.

2.1 Q-Learning

Reinforcement Learning (RL) Systeme interagieren mit deren Umgebung, indem sie eine Reihe von Aktionen durchführen, und die daraus entstehenden Kosten oder Belohnungen bewerten, um eine optimale Strategie für die zu lösenden Aufgaben zu ermitteln [Sutton und Barto 1998].

Q-Learning [Watkins 1989] ist eine modellfreie RL Methode, die iterativ Zustand-Aktions-Paare bewertet. Zu jedem Zustand-Aktions-Paar wird ein Q-Wert zugewiesen, welcher die Bewertung der Aktion a in dem Zustand s entspricht. Nach jedem Zustandsübergang des Systems wird der entsprechende Q-Wert wie folgt aktualisiert:

$$Q_{k+1}(s,a) := (1-\alpha)Q(s,a) + \alpha(c + \gamma \min_b Q_k(s',b)). \tag{2.1}$$

Der Lernfaktor $0 <= \alpha <= 1$ bestimmt wieviel der aktuelle Zustandsübergang den Q-Wert verändert. Bei kleinem Lernfaktor wird die bis zum Zeitpunkt gesammelte Erfahrung stärker gewichtet und bei großem Lernfaktor spielen nur die von aktuellsten Bewertungen der Zustand-Aktions-Paare ermittelten Kosten c eine signifikante Rolle. Der Diskontfaktor γ gewichtet den Q-Wert von Folgezustand s' und Aktion b, wobei b im Vergleich mit allen anderen Aktionen die kleinsten Kosten in s' auslöst.

Wenn die Aktualisierung der Q-Werte nach jedem Zustandsübergang erfolgt, wird vom soge-

nannten online Lernen gesprochen. Dagegen werden beim offline Lernen die Q-Werte erst aktualisiert, wenn die Trainingsphase abgeschlossen ist.

Die optimale Strategie $\pi^* : S \to A$ nutzt die Q-Werte, um die beste Aktion für den aktuellen Zustand zu bestimmen. Die Aktion, welche die kleinsten Kosten verursacht, wird ausgewählt. Die optimale Strategie wird formal wie folgt definiert:

$$\pi^* = \arg\min_a Q(s,a). \tag{2.2}$$

Es ist theoretisch bewiesen, dass Q-Learning unter gewissen Voraussetzungen zur optimalen Strategie konvergiert [Watkins 1989].

In der Trainingsphase von Q-Learning und anderen RL-Methoden selektiert eine sogenannte Explorationsstrategie welche Aktion ausgeführt wird. Wie schnell Q-Learning zur optimalen Strategie konvergiert, hängt stark davon ab. Bereits in der Trainingsphase vergleichen manche Explorationsstrategien die bis zum Zeitpunkt ermittelten Q-Werte. Die beste Aktion bezüglich der aktuellen Q-Werte wird als greedy bezeichnet. Populär sind die $\varepsilon - greedy$ Strategien, welche immer mit einer $1-\varepsilon$ Wahrscheinlichkeit die beste Aktion selektieren, wobei $0 < \varepsilon < 1$. Und mit einer Wahrscheinlichkeit von ε wird eine zufällige Aktion aus dem definierten Aktionsraum ausgewählt. Für manche Aufgaben kann es nachteilig sein, dass alle Aktionen bis auf der Beste gleichwahrscheinlich ausgeführt werden. Alternativ dazu kommen die Softmax-Explorationsstrategien zum Einsatz. Diese gewichten die Aktionswahrscheinlichkeiten bezüglich der Q-Werte, indem zum Beispiel die Gibbs-Boltzmann-Verteilung verwendet wird. Die Wahrscheinlichkeit, dass eine Aktion ausgeführt wird, ist in diesem Fall gegeben als:

$$\frac{e^{Q(s,a)/\tau}}{\sum_{a_i=1}^{n} e^{Q(s,a_i)/\tau}}, \tag{2.3}$$

wobei hier die Q-Werte nicht aus Kosten, sondern aus Belohnungen ermittelt sind. τ ist positiv definiert und wird Temperatur genannt. Je kleiner τ ist, desto größer Rolle die Q-Werte bei der Gewichtung spielen. Bei $\tau \to 0$ wird eine $\varepsilon - greedy$ ähnliche Exploration geleistet. Großes τ macht die Ausführung jeder Aktion nahezu gleichwahrscheinlich. Welche Explorationsstrategien und Parametrisierung zu einer schnelleren Konvergenz der Q-Funktion führen, hängt von der zu lösenden Aufgabe.

Die Q-Werte werden tabellarisch verwaltet, wenn Zustände und Aktionen diskret sind. Bei kontinuierlichen Zustands- und Aktionsräumen kann eine Diskretisierung angewandt werden. Diskretisierung ist aber für hochdimensionale Probleme nicht geeignet. Eine andere Alternative ist das Ganze als Regressionsproblem zu betrachten. Hier geht es um eine Aufgabenstellung des überwachten Lernens, wobei ein Regressor $Q(s,a \mid \theta)$ definiert wird, der s und a als Ein-

gabe nutzt und durch einen Vektor von Parametern θ parameterisiert wird, um die Q-Werte zu approximieren [Alpaydin 2004].

Zu diesem Zweck eignen sich künstliche neuronale Netze. Das Problem bei der online Implementierung dieses Ansatzes ist, dass jede einzige Aktualisierung der Neurongewichten die gesamte Approximationsleistung beeinträchtigen kann. Aus diesem Grund kann es sehr lange dauern bis die Q-Funktion erfolgreich approximiert wird [Riedmiller 2005]. Andere Methoden des überwachten Lernens sind nicht schnell genug, um die Aktualisierung der Approximation online durchzuführen. Folglich sind offline Methoden zur Approximation der Q-Funktion vorteilhaft.

2.2 Fitted Q Iteration

Eingaben: Die Trainingsdaten T und der Approximationsalgorithmus $Q(s,a \mid \theta)$
Ausgabe: Die Approximation der Q-Funktion Q_N, wobei N die Iterationenanzahl ist.
Initialisierung:
$N := 0$
$Q(s,a) := 0$ überall in $S \times A$
Iterationen:

- $N := N + 1$
- Bilde den Trainingsdatensatz $P = \{(i^l, o^l), l = 1, ..., |T|\}$ wie folgt:

$$i^l = (s^l, a^l), \tag{2.4}$$

$$o^l = c^l + \gamma \min_b Q_{N-1}(s'^l, b). \tag{2.5}$$

- Trainiere $Q(s,a \mid \theta)$ mit P, um Q_N zu bilden.

Abbildung 2.1: Fitted Q Iteration Algorithmus

Fitted Q Iteration (FQI) ist ein offline Algorithmus zur Approximation der Q-Funktion [Ernst u. a. 2005]. Der Algorithmus ist in Abbildung 2.1 dargestellt. Die Trainingsdaten werden während einer Trainingsphase in 4-Tupel der Form (s,a,s',c) gespeichert. Der Trainingsdatensatz wird mit

T bezeichnet. Fitted Q Iteration besteht aus zwei wesentlichen Schritten: Die Generation eines Trainingsdatensatzes P und das Trainieren der Approximationsmethode mit diesem Datensatz. P besteht aus Eingaben (Zustands- und Aktionsvariablen) und einer Ausgabe, welche den Q-Wert entspricht. Dieser Q-Wert wird aus der Summe von in der Training ermittleten Kosten des Zustandsübergangs und den minimalen geschätzten Kosten für den Folgezustand gebildet. Zur Berechnung der Q-Werte fehlt der Lernfaktor, weil beim offline Lernen alle Zustandsübergänge gleich gewichtet werden.

In der ersten Iteration approximiert der Funktionsapproximator die Kosten für die Zustand-Aktions-Paare, indem Zustands- und Aktionsvariablen als Eingaben und die beim Training ermittelten Kosten als Ausgaben eingesetzt werden. Danach bleiben bei jeder Generation des Trainingsdatensatzes P die Eingaben unverändert und die Ausgaben werden bezüglich der in der vorherigen Iteration approximierten Q-Werte der Folgezustände aktualisiert. Somit approximiert die resultierende Q-Funktion nicht nur die Kosten für die aktuellen Zustandsübergänge, sondern auch die voraussichtlichen Kosten, die nach der Ausführung der besten Aktion im Folgezustand entstehen würden. Um den Algorithmus abzubrechen, kann der Anzahl der Iterationen fest definiert werden. Andernfalls iteriert der Algorithmus so lange bis die Q-Funktion sich nicht mehr stark ändert. Dieses Abbruchkriterium ist aber nur dann anwendbar, wenn die Folge Q_N konvergiert.

Die Auswahl der Approximationsmethode in der Fitted Q Iteration bleibt dem Anwender überlassen. Wenn als Approximationsmethode ein künstliches neuronales Netz eingesetzt wird, spricht man von Neural Fitted Q Iteration (NFQ). In [Riedmiller 2005] sind Erfolge von NFQ-Anwendungen bei typischen Regelungsaufgaben vorgestellt.

2.3 Funktionsapproximation

In diesem Abschnitt sind Grundlagen von künstlichen neuronalen Netzen (KNN) und Support-Vektor-Regression (SVR) beschrieben. In dieser Arbeit wurden diese Methoden zur Approximation eingesetzt und ausgewertet.

2.3.1 Neuronale Netze

Künstliche neuronale Netze, welche erfolgreich nichtlineare Funktionen approximieren, sind dem menschlichen Gehirn nachempfunden. Ähnlich dem Gehirn bestehen die neuronalen Netze aus mehreren Verarbeitungseinheiten (Neuronen), die miteinander in gewichteter Verbindung stehen. Das eigentliche Lernen oder Approximation einer Funktion wird durch Aktualisierung der Gewichtungen geleistet [Russell und Norvig 2009].

Die grundlegende Verarbeitungseinheit der neuronalen Netze ist das Perzeptron. Es summiert alle gewichteten Eingänge und setzt eine Aktivierungsfunktion ein, um Nichtlinearität in der Ausgabeberechnung einzubringen. Die Ausgabe der Aktivierungsfunktion sendet das Perzeptron an seinen Ausgänge (vgl. Abbildung 2.2).

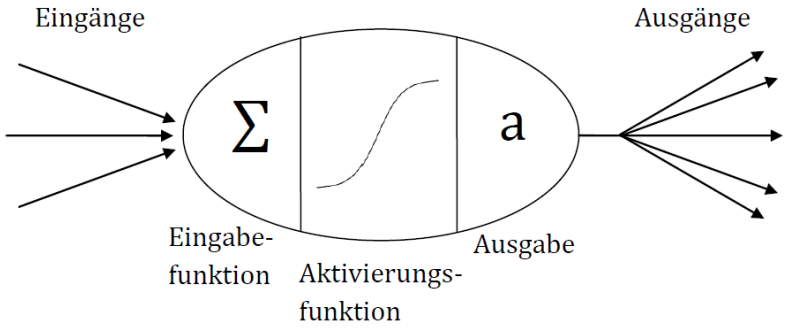


Abbildung 2.2: Perzeptron

Ein oder mehrere Neuronen bilden eine Schicht. Die Struktur eines neuronalen Netzes definiert wie viele Neuronen sich auf wie vielen Schichten befinden und wie diese miteinander verbunden sind. Typischerweise bestehen neuronale Netze aus einer Eingabeschicht, einer oder mehreren verdeckten Schichten und einer Ausgabeschicht. Meist verbreitet sind die Feedforward-Netze, bei denen Neuronen von einer Schicht immer nur mit Neuronen der nächst höheren Schicht verbunden sind (vgl. Abbildung 2.3). In der Regel sind Verbindungen nur in einer Richtung erlaubt.

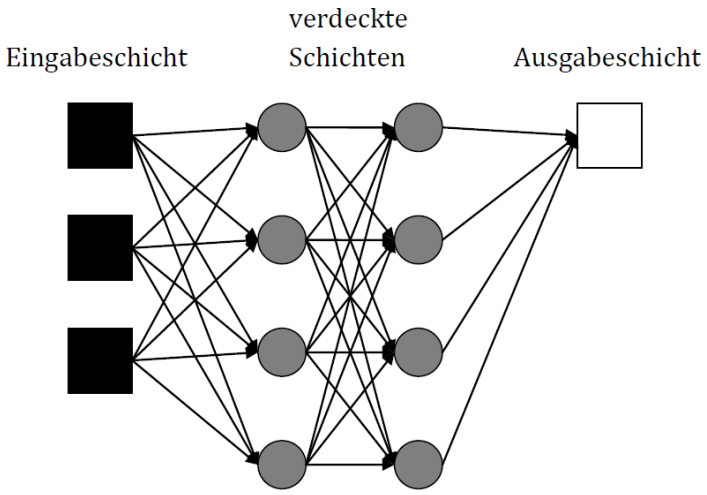


Abbildung 2.3: Feedforward-Netz mit zwei verdeckten Schichten

Jede kontinuierliche Funktion kann von neuronalen Netze mit einer verdeckten Schicht approximiert werden und Netze mit zwei verdeckten Schichten können jede beliebige Funktion approximieren. Die Anzahl der Neuronen in den verdeckten Schichten spielt bei der Approximation eine signifikante Rolle [Russell und Norvig 2009]. Neuronale Netze mit vielen Neuronen führen zu einer Überanpassung des vorgegebenen Datensatz und haben deshalb eine schlechte Generalisierungseigenschaft. Kleine Netze dagegen generalisieren zu stark. Es gibt keine Regeln, welche die optimale Anzahl der Neuronen in den verdeckten Schichten bestimmen und meistens wird die Größe des Netzes bezüglich der Anzahl der Neuronen in Eingabe- und Ausgabeschicht gewählt.

Um eine Funktionsapproximation zu leisten, müssen die neuronalen Netze mit Trainingsdaten trainiert werden. Diese Daten bestehen aus Eingabe- und Ausgabevektoren, die Beobachtungen von dem zu lösenden Problem beschreiben. Der Eingabevektor jeder Beobachtung wird an der Eingabeschicht des Netzes angelegt und vorwärts durch das Netz propagiert. Auf diese Weise wird ein Ausgabevektor mit den aktuellen Verbindungsgewichten berechnet. Der Ausgabevektor des Netzes wird mit dem Sollausgabevektor verglichen, um den Fehler des Netzes zu bestimmen. Wie die Gewichte entsprechend diesem Vergleich aktualisiert werden, hängt vom Lernalgorithmus ab. Der populärste Algorithmus zu diesem Zweck ist Backpropagation [Rumelhart u. a. 1988]. Um die Gewichte zu aktualisieren, wird der Fehler des Netzes über die Ausgabe- zur Eingabeschicht zurück propagiert. Obwohl Backpropagation bei vielen Anwendungen erfolgreich eingesetzt wird, hat der Algorithmus einige Nachteile. Zum Beispiel ist es nicht garantiert, dass er konvergiert und selbst wenn, dann geschieht es sehr langsam. Außerdem konvergiert der Algorithmus auf ein lokales Minimum. Folglich kann es nicht sichergestellt werden, dass die beste Lösung für das zu lösende Problem ermittelt wird. Trotzdem ist Backpropagation besonders beim online Lernen sehr beliebt, weil die Aktualisierung der Gewichte sehr schnell erfolgt. Beim offline Lernen werden meistens fortgeschrittene Algorithmen wie Levenberg-Marquardt und Rprop [Riedmiller und Braun 1993] bevorzugt, weil diese schneller konvergieren und öfters ein besseres Minimum finden.

2.3.2 Support-Vektor-Maschinen

Eine Support-Vektor-Maschine (SVM) kann sowohl zur Klassifizierung als auch zur Approximation eingesetzt werden. Das Ziel bei der Klassifizierung von zwei linear trennbaren Klassen ist eine Hyperebene einzupassen, welche die zwei Klassen optimal trennt. Dabei wird der Abstand zu den Objekten, welche der Hyperebene am nächsten liegen, maximiert. In [Alpaydin 2004] wird es wie folgt formuliert:

$$\min \frac{1}{2} \|w\|^2 \text{ unter } r^t (w^T x^t + w_0) \geq +1, \forall t, \tag{2.6}$$

wobei $\chi = \{x^t, r^t\}$ die Stichproben sind, $r^t = +1$, falls $x^t \in C_1$ und $r^t = -1$ falls $x^t \in C_2$. Das ist ein Standartproblem der quadratischen Optimierung, welches direkt gelöst werden kann, um w und w_0 zu bestimmen. Dann beträgt die Entfernung von der eingepassten Hyperebene zu den am nächsten liegenden Objekten auf beiden Seiten $1/\|w\|$ und der Gesamtabstand (Margin) ist somit $2/\|w\|$ (vgl. Abbildung 2.4).

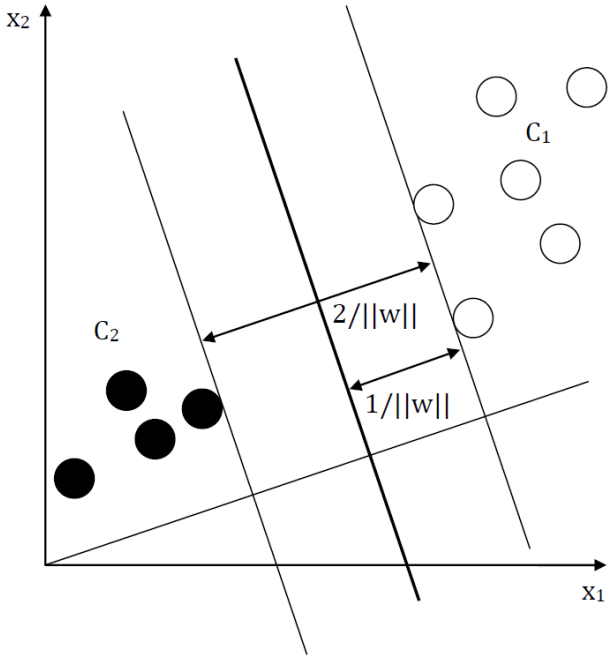


Abbildung 2.4: Optimale SVM-Trennebene

Falls die Daten nicht trennbar sind, wird die Hyperebene, die den geringsten Fehler auslöst, gesucht. Um nichtlinear trennbare Klassen zu klassifizieren, kommt der Kerneltrick zum Einsatz. Mittels nichtlinearen Kernfunktionen wird das Problem in einer höheren Dimension überführt, wo die Klassen linear trennbar sind. Hierfür werden üblicherweise Polynome, radiale Basis- oder Sigmoidfunktionen verwendet. Um eine Regression zu leisten, werden die Support-Vektor-Maschinen um eine Verlustfunktion erweitert [Drucker u. a. 1996].

3 NFQ-Anwendungen

Das in dieser Arbeit entwickelte Verfahren zur Spurführung setzt den Neural Fitted Q Iteration Algorithmus ein, um die Q-Funktion zu approximieren. In diesem Kapitel werden zwei Anwendungen zur Regelung beschrieben, welche von NFQ profitieren.

3.1 Lernverfahren zur autonomen Fahrzeugspurführung

In [Montemerlo u. a. 2007] wird vorgestellt wie ein Fahrzeug innerhalb von 20 Minuten lenken "lernt". Das Verfahren basiert auf NFQ [Riedmiller 2005] und ist komplett datengetrieben, d.h. Simulation oder Fahrzeugmodell sind nicht erforderlich. NFQ ist eine Art der Fitted Q Iteration, wobei ein neuronales Netz zur Approximation eingesetzt wird.

3.1.1 Der NFQ-Lenkungsregler

Ziel des NFQ-Lenkungsreglers ist die Abweichung zur Sollspur zu minimieren, sodass eine präzise autonome Fahrzeugquerführung realisisiert wird. Trainingsdaten werden während der Trainingsphase in 3-Tupel der Form (s,a,s') gespeichert. Ein neuronales Netz, welches die Q-Funktion approximiert, wird mittels NFQ trainiert. Der NFQ-Lenkungsregler verwendet das neuronale Netz, um die Kosten für mehrere Aktionen (Lenkungsstellwerte) in dem aktuellen Fahrzeugzustand zu approximieren. Die Aktion, welche die kleinsten Kosten verursacht, wird selektiert. Auf diese Weise minimiert der NFQ-Lenkungsregler kontinuierlich die Abweichung zur Sollspur cte. Der Schwerpunkt ist diese Abweichung kleiner als 0.5 m zu halten. Die Bewertungsfunktion $c(s,a)$ wird wie folgt definiert:

$$c(s,a) = \begin{cases} 0 & \text{, wenn } |cte| < 0.05\,m \text{ (Erfolg)} \\ +1 & \text{, wenn } |cte| < 0.5\,m \\ 0,01 & \text{, sonst} \end{cases} \tag{3.1}$$

Basierend auf [Hoffmann u. a. 2007] wird der Fahrzeugszustand durch folgende fünf kontinuierlichen Variablen beschrieben: die Abweichung zur Sollspur cte, die erste Ableitung davon $\dot{cte}$,

die Geschwindigkeit v, der Kursfehler ψ (vgl. Abbildung 3.1) und yaw-rate-matching yrm, wobei es sich aus Änderung der Sollspurgeometrie und der Fahrzeugfahrtrichtung zusammensetzt.

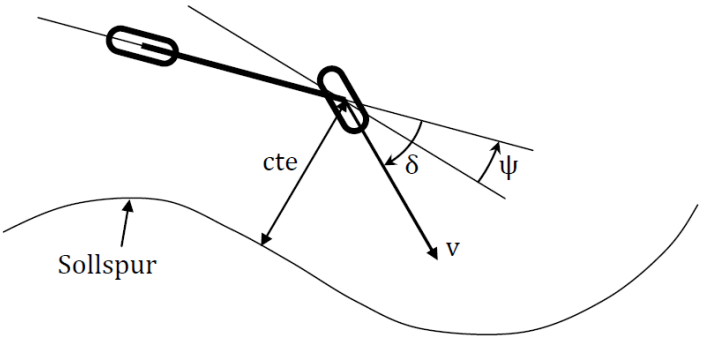


Abbildung 3.1: Das kinematische Modell

Eine Diskretisierung der Lenkungsstellwerte u ist erforderlich, weil das trainierte neuronale Netz die Q-Werte aller möglichen Aktionen in Echtzeit approximieren soll, damit der NFQ-Lenkungsregler die beste selektieren kann. Diese Diskretisierung kann zu einem instabilen Lenkungsverhalten führen. Ein Dynamic Output Element mit einem Integrator (I-DOE) [Riedmiller 1997] kommt zum Einsatz, um diese Problematik zu lösen. Das I-DOE summiert die Regleraktionen kontinuierlich und dessen Zustand lässt sich durch

$$s_{DOE}(t) = s_{DOE}(t-1) + u'(t) \tag{3.2}$$

ausdrücken, wobei $u'(t)$ die vom Regler ausgewählte Aktion repräsentiert und der I-DOE Ausgang $u(t) = s_{DOE}$ als Stellwert für die Lenkung benutzt wird. Der Ausgang wird gleichzeitig als Eingang für den Regler verwendet, damit Information über den gesamten Systemzustand vorhanden ist. Der daraus resultierenden I-DOE-Regler erreicht ein sehr glattes Verhalten, obwohl nur fünf Aktionen angewandt worden sind [Montemerlo u. a. 2007].

3.1.2 Ergebnisse

Das in diesem Abschnitt beschriebene Verfahren wurde auf einem echten Fahrzeug erfolgreich getestet. Bereits nach 16 Minuten bei einer Geschwindigkeit bis zu 9 m/s wird der NFQ-Lenkungsergler gelernt. Dieser Regler schaft die Abweichung zur Sollspur kleiner als 0.5 m innerhalb einer vollen Runde der Teststrecke zu halten [Montemerlo u. a. 2007].

3.2 Das inverse Pendel

Das inverse Pendel ist eine klassische Aufgabe der Regelungstechnik für die Stabilisierung einer instabilen Regelstrecke. Ein Wagen mit einem darauf montierten Stab ist eine Variante dieser Aufgabe (vgl. Abbildung 3.2). Die Pendelbewegung wird durch die horizontale Bewegung des Wagens geregelt, um den Stab in aufrechter Stellung zu bringen bzw. halten.

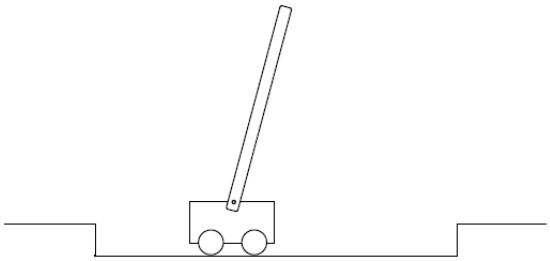


Abbildung 3.2: Das inverse Pendel

In [Riedmiller 2005] wird der Neural Fitted Q Iteration Algorithmus vorgestellt und bei Standartaufgaben der Regelungstechnik untersucht. NFQ basiert auf dem Fitteq Q Iteration und verwendet ein neuronales Netz für Approximation. Die erste erfolgreiche Strategie für Regelung des inversen Pendels wird durchschnittlich bereits nach fünf Minuten Training erreicht.

[Riedmiller 2005] stellt die Ergebnisse des NFQ-Pendelreglers vor. Das neuronale Netz, welches die Q-Funktion approximiert, besteht aus fünf Eingabeneuronen für die Zustand-Aktions-Beschreibung, zwei verdeckte Schichten mit jeweils fünf Neuronen und einem Ausgabeneuron. Die Reglerfrequenz beträgt 50 Hz. Zwei Aktionen sind anwendbar, -10 N und +10 N, um den Wagen horizontal zu bewegen. Die Zielposition des Wagens ist genau in der Mitte der 4,8 m langen Führungsbahn definiert.

In der Tabelle 3.1 sind die durchschnittlichen Zeiten, Kosten und Anzahl von benötigten Episoden und Zyklen zum Erreichen von der ersten und besten erfolgreichen Strategien. Diese Ergebnisse basieren auf 1000 Versuche, wobei bei jedem Versuch das neuronale Netz neu initialisiert wird und jeder Versuch aus 500 Episoden besteht. Am Anfang jeder Episode befindet sich der Wagen an einer zufälligen Position auf der Führungsbahn und der Stabwinkel liegt im Bereich von -0,3 bis 0,3 rad. Alle Episoden sind maximal 100 Zyklen lang und werden generiert, indem die Aktionen $greedy$ bezüglich der Q-Funktion gewählt werden. Nach jeder Episode wird eine Iteration des NFQ-Algorithmus ausgeführt und somit die Q-Funktion aktualisiert. Eine Reglerstrategie ist erfolgreich, wenn am Ende der Episode der Stab noch aufrecht steht und der Wagen maximal 0,05 m von der Führungsbahnmitte entfernt ist. Die innerhalb einer Episode entstehenden Kosten dienen zur Auswertung der Reglerstrategien.

Erste erfolgreiche Strategie			
Episoden	Zyklen	Zeit	Kosten
197,3	14439,8	4m 49s	319,1
Beste erfolgreiche Strategie			
Episoden	Zyklen	Zeit	Kosten
354,0	28821,1	9m 36s	132,9

Tabelle 3.1: NFQ-Ergebnisse zur Regelung des inversen Pendels

4 Prädiktives NFQ-Spurführungsverfahren

In diesem Kapitel wird das in dieser Arbeit entwickelte prädiktive NFQ-Spurführungsverfahren vorgestellt. Das Verfahren basiert auf dem NFQ Algorithmus. Aus Trainingsdaten generiert NFQ einen Datensatz zum Trainieren eines neuronalen Netzes, welches die Q-Funktion in kontinuierlichen Zustands- und Aktionsräumen approximiert (vgl. Abschnitt 2.2). Zur Bewertung von Aktionen aus dem Aktionsraum A in dem aktuellen Zustand s approximiert die Q-Funktion $Q(s,a)$ die Kosten für die Zustand-Aktions-Paare $\{s \times A\}$.

Der Pure Pursuit Algoritmus wird zur Begrenzung des Aktionsraums eingesetzt. Fünf Variablen der Zustandbeschreibung erfassen den kinematischen Fahrzeugzustand. Die Lenkungsaktionen, welche aufgrund der Lenkungstotzeit noch nicht eingesetzt sind, erweitern die Zustandbeschreibung, sodass die Auswertungen der Zustand-Aktions-Paare nicht nur auf der aktuellen Fahrzeugkinematik basiert. Zusätzlich wird auch das Lenkungsverhalten während der Totzeit berücksichtigt, was eine prädiktive Auswertung ermöglicht.

In Abschnitt 4.1 wird die Aufgabenstellung definiert. Es folgt eine detaillierte Beschreibung des prädiktiven NFQ-Spurführungsverfahrens (vgl. Abschnitt 4.2) und die Parametrisierung des Verfahrens wird in Abschnitt 4.3 beschrieben.

4.1 Aufgabenstellung

Ziel des Verfahrens ist die Generierung eines Lenkungsreglers, welcher eine präzise und schnelle Spurführung bei in Abschnitt 1.1 beschriebenen Einschränkungen realisiert. Die Fahrspurgeometrie wird von einem polynombasierten Spurerkennungsalgorithmus identifiziert [Jenning 2009]. Dieser Algorithmus liefert zwei Polynome zweiten Grades, welche die zwei zur Fahrspur gehörigen Markierungen im Fahrzeugkoordinatensystem approximieren. Anhand dieser Polynome, des aktuellen Lenkwinkels und der Geschwindigkeit soll der Lenkungsregler den optimalen Stellwert für die Lenkung bestimmen, sodass das Fahrzeug möglichst präzise die Mitte der Fahrspur verfolgt.

Methoden zur Verfolgung einer definierten Fahrbahn unterteilen sich in drei Gruppen. Die ers-

te Kategorie verwendet nur geometrische Zusammenhänge zwischen Fahrzeug und Fahrbahn, um den Lenkwinkel zu bestimmen. Andere Methoden basieren auf dem kinematischen oder dynamischen Fahrzeugmodell. Es soll ein hybrides Verfahren entwickelt werden, welches sowohl von der Robustheit der geometrischen Methoden als auch von der Leistung der kinematischen Methoden profitiert.

4.2 Spurführungsverfahren

In diesem Abschnitt wird das prädiktive NFQ-Spurführungsverfahren vorgestellt. Die Auswahl und die Begrenzung des Aktionsraums werden in Abschnitt 4.2.1 beschrieben. Nachfolgend werden die Variablen zur Beschreibung des Fahrzeugzustandes vorgestellt (vgl. Abschnitt 4.2.2). Die Bewertungsfunktion wird in Abschnitt 4.2.3 definiert. Anschließend wird beschrieben, wie das Beste aus mehreren trainierten neuronalen Netzen bestimmt wird (vgl. Abschnitt 4.2.4). Ablauf der Trainingsphase wird in Abschnitt 4.2.5 vorgestellt.

4.2.1 Aktion

Zur Auswertung aller möglichen Aktionen in dem aktuellen Zustand liefert NFQ eine Approximation der Q-Funktion. Für die Aufgabe der Spurführung besteht die Möglichkeit den Reglersollwert als Aktion zu verwenden, falls ein Lenkungsregler vorhanden ist. Die NFQ-Spurführung setzt Lenkungsstellwerte als Aktionen ein. Somit ist das Ziel, den optimalen Stellwert für die Lenkung bei dem entsprechenden Fahrzeugzustand zu ermitteln. Das Regulieren der Fahrgeschwindigkeit wird separat von der Lenkungssteuerung erledigt. In der Regel lassen sich Lenkservos von Modellfahrzeugen im Wertebereich von -100 (voller Lenkeinschlag nach links) bis 100 (voller Lenkeinschlag nach rechts) steuern. Das gilt auch für die im FAUST Projekt entwickelten Fahrzeuge (vgl. Abschnitt 5.1.1). In der Simulationsumgebung TORCS sind Stellwerte von -1.0 bis 1.0 für die Lenkung zugelassen (vgl. Abschnitt 5.1.2).

Bei der Ermittlung der besten Aktion soll die Auswertung aller Aktionen in einem definierten Zeitraum erfolgen. Das erfordert eine Begrenzung und eine Diskretisierung des Aktionsraums. Außerdem ist es vorteilhaft, Aktionen, die sehr wahrscheinlich die Präzision der Fahrt verschlechtern, auszuschließen. Zur Begrenzung kommt der Pure Pursuit Algorithmus zum Einsatz (vgl. Abschnitt 1.2). Vor Auswertung der Aktionen im aktuellen Zustand wird ein Pure Pursuit basierter Lenkungsstellwert y_{pp} wie folgt berechnet:

$$y_{pp} = K_p \arctan\left(\frac{2g_y L}{D^2}\right), \tag{4.1}$$

wobei K_p ein Verstärkungsfaktor darstellt. Als nächstes approximiert das trainierte neuronale Netz die Q-Werte nur für die Stellwerte im Bereich von $y_{pp} - expl_{max}$ bis $y_{pp} + expl_{max}$, wobei $expl_{max}$ der maximale Explorationswert ist. Dieser wird so ausgewählt, dass das Verlassen der zu verfolgenden Fahrspur in allen Fällen verhindert wird. Auf diese Weise wird die kamerabasierte Spurerkennung unterstützt. Außerdem wird der Aktionsraum kleiner, was die Approximation entlastet. Der auszuwertende Aktionsraum ergibt sich aus:

$$A := \{y_i \mid i \in \mathbb{N} \wedge 0 <= i <= 10\} \tag{4.2}$$

wobei

$$y_i = y_{pp} - expl_{max} + \left(\frac{i * 2expl_{max}}{10}\right). \tag{4.3}$$

Zur Steuerung der FAUST-Fahrzeuge werden die Stellwerte gerundet. Dieselbe Begrenzung und Diskretisierung des Aktionsraums wird auch in der NFQ-Trainingsphase eingesetzt.

4.2.2 Fahrzeugzustand

Der Fahrzeugzustand enthält alle Variablen, welche die Kinematik des Fahrzeugs in seinem Koordinatensystem beschreiben. Die Sollspur wird durch ein Polynom zweiten Grades $p(x) = ax^2 + bx + c$ im Fahrzeugkoordinatensystem approximiert. Zur Beschreibung der Fahrzeugkinematik werden die Koeffizienten dieses Polynoms zusammen mit dem aktuellen Lenkwinkel und der Geschwindigkeit als Zustandsvariablen eingesetzt. Das verwendete Fahrzeugkoordinatensystem wird in Abbildung 4.1 dargestell und in Abschnitt 5.1.1 detailliert erläutert. Mit f wird der vordere Überhang des Fahrzeugs bezeichnet.

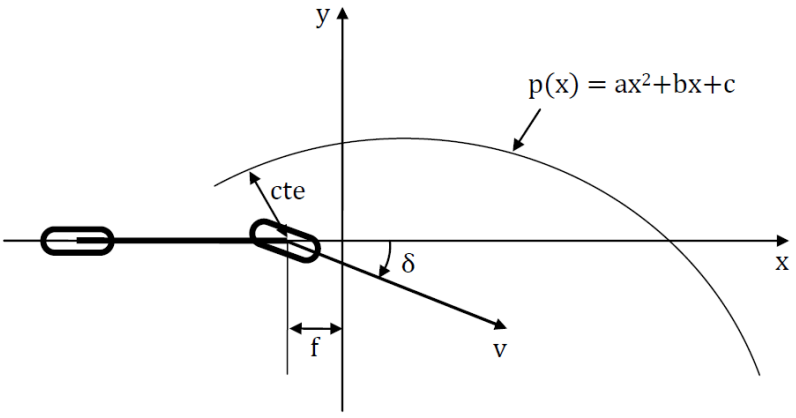


Abbildung 4.1: Fahrzeugkoordinatensystem

Wie bereits erwähnt, ist die Lenkungstotzeit T_t für die Stabilität der Spurführung ausschlaggebend (vgl. Abschnitt 1.2). Aus diesem Grund nutzen manche Verfahren ein Systemmodell zur Prädiktion des Fahrzeugzustands nach Ausführung der Lenkungsaktionen [Rankin 1997]. Das prädiktive NFQ-Spurführungsverfahren erweitert die Zustandsbeschreibung um die letzten k Lenkungsaktionen, sodass der FQI-Regressor die Wirkung aller Aktionen, welche aufgrund der Totzeit das Lenkungsverhalten noch nicht beeinflusst haben, berücksichtigt. Die Lenkung wird mit einer Frequenz von $1/T$ Hz gesteuert, wobei die Periodendauer $T = T_t/k$ ist. Außerdem in der Trainingsphase werden die Kosten der Zustand-Aktions-Paare (s,a) erst nach $k+1$ Perioden ermittelt, sodass nur die Auswirkung der vor $k+1$ Perioden ausgeführten Aktion bewertet wird. In Abbildung 4.2 wird der Fall für $T_t = 0,3$ s und $k = 3$ dargestellt. Hier repräsentiert a_n die Aktion, welche am Anfang der Periode t_n ausgeführt wird.

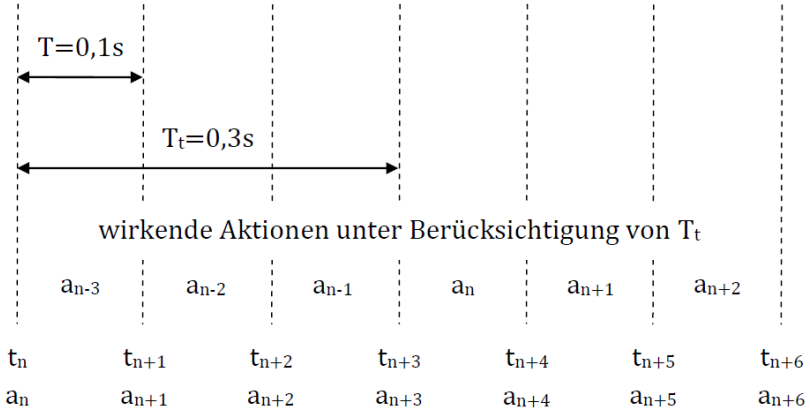


Abbildung 4.2: Wirkende Aktionen unter Berücksichtigung der Lenkungsverzögerung

Bei $k = 3$ wird der Fahrzeugzustand durch folgende Variablen beschrieben:

- a: Koeffizient des quadratischen Polynoms
- b: Koeffizient des quadratischen Polynoms
- c: Koeffizient des quadratischen Polynoms
- δ: Lenkwinkel
- v: Geschwindigkeit
- y_{t-3}: vorvorletzte ausgeführte Lenkungsaktion
- y_{t-2}: vorletzte ausgeführte Lenkungsaktion
- y_{t-1}: letzte ausgeführte Lenkungsaktion

4.2.3 Bewertungsfunktion

Während der Trainingsphase werden Daten in 5-Tupel der Form (s,a,s',y'_{pp},c) gesammelt, wobei die Kosten c die Auswertung von Zustand-Aktions-Paare (s,a) darstellen (vgl. Abschnitt 2.2). Zur Ermittlung dieser Kosten kommt eine Bewertungsfunktion zum Einsatz. Ähnlich wie in [Montemerlo u. a. 2007] ist die Bewertungsfunktion mit der Abweichung zur Sollspur cte gebunden (vgl. Abbildung 4.3). Im Abschnitt 5.4 wird beschrieben, wie cte ermittelt wird. Vor der Bewertung wird eine skalierte Sollspurabweichung cte_s wie folgt berechnet:

$$cte_s = \left| \frac{0,5 * cte}{cte_{set}} \right|, \tag{4.4}$$

wobei cte_{set} die zu tolerierende Sollspurabweichung darstellt. Die Kosten der Zustand-Aktions-Paare werden wie folgt ermittelt:

$$c_{s,a}(cte_s) = \begin{cases} 1,0 & \text{, wenn } cte_s > 2 \\ 0,1 * 2^{1+cte_s} & \text{, wenn } cte_s > 0,5 \\ 0,01 & \text{, sonst (Erfolg)} \end{cases} \tag{4.5}$$

Die Bewertungsfunktion wird auch in Abbildung 4.3 dargestellt. Obwohl kein weiterer Verlauf der Fahrspurgeometrie berücksichtigt wird, reicht die aktuelle Sollspurabweichung zur Auswertung aus, weil bei mehreren FQI-Iterationen die Q-Werte auch von Auswertungen der Folgezustände abhängig sind. Somit werden Aktionen besser bewertet, welche eine präzise Spurführung nach deren Ausführung ermöglichen.

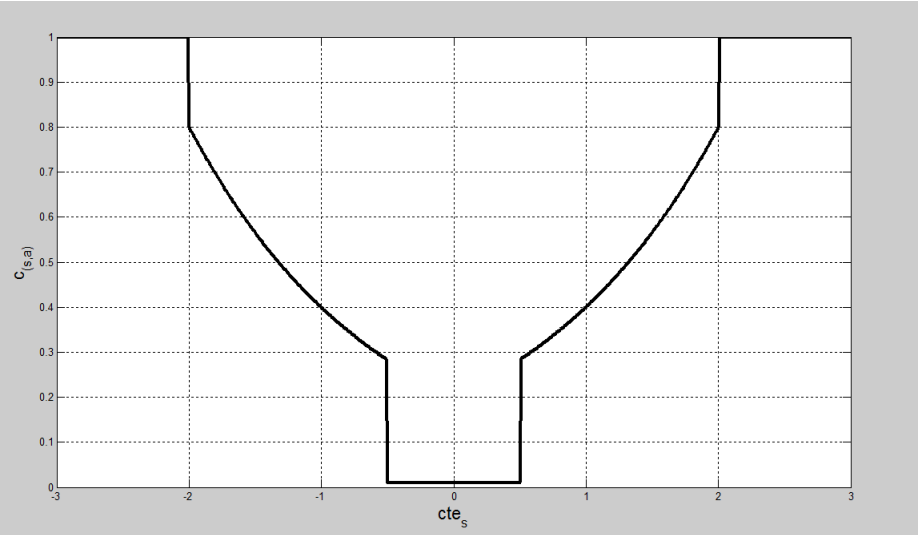


Abbildung 4.3: Bewertungsfunktion

4.2.4 Approximationsmethode

Die Auswahl der Approximationsmethode in der Fitted Q Iteration bleibt dem Anwender überlassen. Hinsichtlich der Spurführung soll diese Methode auch für hochdimensionale nichtlineare Funktionen eine genaue Approximation leisten können, weil der Fahrzeugzustand durch mehrere kontinuierliche Variablen beschrieben wird. Viele FQI-Anwendungen setzen künstliche neuronale Netze zur Approximation der Q-Funktion ein (vgl. Kapitel 3). In dieser Arbeit wurde die Approximationsleistung von neuronalen Netzen und Support-Vektor-Maschinen verglichen (vgl. Abschnitt 6.1). Neuronale Netze liefern eine deutlich präzisere Approximation.

Das prädiktive NFQ-Verfahren trainiert zehn neuronale Netze bei jeder FQI-Iteration. Zur Approximation der Q-Funktion wird das Beste eingesetzt. Wie bereits in Abschnitt 2.2 beschrieben, generiert FQI in jeder Iteration den Trainingsdatensatz $P = \{(i_l, o_l), l = 1, ..., |T|\}$. Die neuronalen Netze werden mit P trainiert, wobei i_l als Eingaben und die Auswertungen o_l als Ausgaben verwendet werden. Der neundimensionale Vektor i_l besteht aus acht Zustands- und einer Aktionsvariable (s_l, a_l). Somit wird der Definitions- und Bildbereich der Approximation von neuronalen Netzen wie folgt formuliert:

$$KNN\text{:} \quad \mathbb{R}^8 \times \mathbb{R}^1 \rightarrow \mathbb{R}^1 \tag{4.6}$$

Die Auswertung der Approximation eines KNN wird wie folgt ermittelt:

$$KNN_{app} = \frac{|Q|}{|P|}, \tag{4.7}$$

wobei

$$Q = \{p_l \mid l = 1 \ldots |P| \wedge p_l \in P \wedge |KNN(s_l, a_l) - o_l| < 0,1\} \tag{4.8}$$

und alle Variablen des Trainingsdatensatzes P bei Trainieren und Auswertung der neuronalen Netze im Bereich $[0,1;0,9]$ linear skaliert sind. Die Skalierung der Daten wird detailliert in Abschnitt 5.3 beschrieben.

4.2.5 Trainingsphase

In der Trainingsphase werden Trainingsdaten in 5-Tupel der Form (s, a, s', y'_{pp}, c) gespeichert, wobei y'_{pp} eine mittels Pure Pursuit ermittelte Lenkungsaktion für den Zustand s' darstellt. Bei der Ermittlung der besten Aktion im s' wird y'_{pp} zur Aktionsbegrenzung eingesetzt.

Die Trainingsphase besteht aus mehreren Episoden. Die Länge einer Episode wird mit $episodeLen$ parametrisiert. Am Ende jeder Episode wird FQI ausgeführt, wobei neuronale Netze zur Approximation verwendet werden. Bei jeder FQI-Iteration wird das Beste aus zehn trainierten neuronalen Netzen gespeichert und in der nächsten Iteration zur Approximation der

Q-Werte eingesetzt. Nach jeder Episode fährt das Fahrzeug eine Runde der Teststrecke zur Analyse der NFQ-Lenkungsregler. Zur Bestimmung der optimalen Lenkungsaktion bezüglich des aktuellen Zustandes wird das in der letzen FQI-Iteration trainierte KNN zur Auswertung der Aktionen verwendet. Nach dieser Testrunde folgt die nächste Episode.

In der ersten Episode der Trainingsphase wird es zufällig ausgewählt, welche Lenkungsaktionen ausgeführt werden. In jeder weiteren Episode approximiert das in vorheriger Episode trainierte neuronale Netz die Q-Werte der Zustand-Aktions-Paare und zur Selektion der Lenkungsaktionen kommt die Softmax-Explorationsstrategie zum Einsatz (vgl. Abschnitt 2.1). Die Explorationsaktion a_{expl} wird unter Berücksichtigung der approximierten Q-Werte wie folgt ermittelt:

$$a_{expl} = \arg\min_{a} randKNN(s,a), \tag{4.9}$$

wobei für jede Aktion aus dem Aktionsraum eine zufällige ganze Zahl $rand$ im Bereich [0;100] generiert wird. Wenn während Training $cte > \frac{s_w}{4}$, eingesetzt wird immer die am besten ausgewertete Lenkungsaktion, um Verlassen der Fahrspur zu vermeiden. Mit s_w wird die Breite der Fahrspur bezeichnet.

In der ersten Trainingsepisode wird die Fahrgeschwindigkeit im Bereich von v_{min} bis v_{max} begrenzt. In jeder weiteren Episode werden diese Grenze um die Differenz $v_{max} - v_{min}$ erhöht. Auf diese Weise entsteht nach jeder Episode ein neuronales Netz, welches für den entsprechenden Geschwindigkeitsbereich trainiert wird. Somit bilden alle neuronale Netze zusammen einen mehrstufigen Lenkungsregler. Zur Realisierung dieses Ansatzes ist ein Tempomat erforderlich, sodass während Training die gewünschte Geschwindigkeit gehalten werden kann.

4.3 Parametrisierung

Die prädiktive NFQ-Spurführung wird durch in Tabelle 4.1 aufgelistete Variablen parametrisiert. In Abschnitte 4.3.1 und 4.3.2 wird vorgestellt, wie Pure Pursuit und FQI parametrisiert werden.

Parameter	Beschreibung
D	Zielentfernung in Pure Pursuit (vgl. Gleichung 4.1)
K_p	Verstärkungsfaktor in Pure Pursuit (vgl. Gleichung 4.1)
$iterations$	Anzahl FQI-Iterationen (vgl. Abschnitt 2.2)
$episodeLen$	Anzahl Trainingsdaten, die innerhalb einer Episode gespeichert werden
$explMax$	maximaler Explorationswert (vgl. Gleichung 4.2)
cte_{set}	zu tolerierende Sollspurabweichung (vgl. Gleichung 4.4)
v_{min}	minimale Geschwindigkeit in der ersten Episode
v_{max}	maximale Geschwindigkeit in der ersten Episode

Tabelle 4.1: Parameterliste der prädiktiven NFQ-Spurführung

4.3.1 Pure Pursuit Parametrisierung

Vor der FQI-Trainingsphase ist eine Parametrisierung von Pure Pursuit erforderlich, weil dieser Algorithmus zur Begrenzung des Aktionsraums eingesetzt wird (vgl. Abschnitt 4.2.1). Die Entfernung D vom Fahrzeug zum verfolgenden Ziel ist der wichtigste Parameter, wie bereits in Abschnitt 1.2 beschrieben wurde. Dieser Parameter wird empirisch ermittelt, indem Fahrten bei unterschiedlicher Zielentfernung untersucht werden. Ausgewählt wird eine möglichst große Entfernung, welche eine fehlerfreie Fahrt hinsichtlich der gestellten Anforderungen ermöglicht. Denn je größer die Zielentfernung, desto stabiler die Fahrt ist (vgl. Abschnitt 6.2). Die empirische Ermittlung von der Zielentfernung soll bei einer möglichst höhen Geschwindigkeit erfolgen. In der Trainingsphase wird diese Geschwindigkeit als v_{min} eingesetzt, sodass die prädiktive NFQ-Spurführung bei höheren Geschwindigkeiten trainiert wird.

4.3.2 FQI-Parametrisierung

Bei der Ermittlung von Q-Werten gewichtet der Diskontfaktor γ den erwarteten Q-Wert des Folgezustandes (vgl. Abschnitt 2.1). Dieser Faktor wird in der Regel annähernd eins gewählt, sodass langfristig minimale Kosten verursacht werden. Das gilt auch für das NFQ-Spurführungsverfahren, weil ein Lenkungsregler, welcher langfristig die Kosten minimiert, garantiert auch in Folgezustände eine präzise Fahrt.

Die Anzahl der FQI-Iterationen parametrisiert ebenfalls wie langfristig die Kosten oder die Belohnung minimiert bzw. maximiert werden. In der ersten FQI-Iteration werden nur die in der Trainingsphase entstandenen Kosten approximiert. Mit jeder weiteren Iteration werden die Q-

Werte auch bezüglich der Folgezustände berechnet und die Q-Werte der Folgezustände sind immer in der vorherigen Iteration bezüglich deren Folgezustände aktualisiert worden. Aus diesem Grund sind die von FQI generierten Q-Werte für langfristige Aufgaben optimiert, wenn die Anzahl der Iterationen groß ausgewählt wird. In diesem Fall ist es nachteilig, dass die Q-Werte von Folgezuständen von einem Regressor approximiert werden. Bei mehreren FQI-Iterationen wird der Regressor öfter angewandt und folglich hat der Approximationsfehler größeren Einfluss auf die Aktualisierung der Q-Werte.

Wie bereits in Abschnitt 4.2.2 erwähnt, ist die Steuerfrequenz von Lenkungstotzeit und Anzahl von letzten ausgeführten Aktionen, welche den Fahrzeugzustand erweitern, abhängig. Bei dem zu testendem Modellfahrzeug beträgt die Lenkungstotzeit 0,3 s (vgl. Abschnitt 6.3). In der Auswertung von dem prädiktiven NFQ-Spurführungsverfahren wird der Fahrzeugzustand um die letzte drei Aktionen erweitert. Folglich zum Sammeln von Trainingsdaten für FQI wird das Lenkungssystem mit 10 Hz gesteuert und der FQI-Regressor wird für diese Steuerfrequenz trainiert.

5 Implementierung des Verfahrens als TORCS-Robot und FAUSTplugin

In diesem Kapitel wird beschrieben wie das prädiktive NFQ-Spurführungsverfahren für die Testumgebung implementiert wurde. Die Testumgebung wird in Abschnitt 5.1 beschrieben. Die Softwarearchitekture wird in Abschnitt 5.2 präsentiert. In Abschnitt 5.3 wird definiert wie die Trainingsdaten skaliert werden. Die Ermittlung der Sollspurabweichung und des aktuellen Lenkwinkels wird in Abschnitt 5.4 bzw. 5.5 erläutert.

5.1 Testumgebung

Die Testumgebung besteht aus dem FAUST-Fahrzeug Onyx und der Autorennen-Simulation TORCS.

5.1.1 Onyx Fahrzeug

Zur Teilnahme am Carolo-Cup 2009 wurde an der HAW Hamburg das Fahrzeug Onyx entwickelt. Dieses Fahrzeug basiert auf einem Ford F-350 Pickup-Modell im Maßstab 1:10 (vgl. Abbildung 5.1). Die Aktorik besteht aus einem Eraser 13.5 Elektromotor und einem Blue Bird BMS630MG Lenkservo, welcher mit 50 Hz gesteuert wird.

Die Auswahl von Sensoren wurde anhand der Erfahrungen aus dem Carolo-Cup 2008 und den dort zu bewältigenden Aufgaben getroffen [Hensel 2008]. Zur Umfelderfassung verfügt das Fahrzeug über zwei Inkrementalgeber an Vorderrädern, vier Ultraschallsensoren, vier Infrarotsensoren, einem Kompass und einer Kamera. Zur Spurerkennung wird eine Monochrom-USB-Kamera UI-1226LE-M aus der Ueye-Serie der Firma IDS-Imaging mit einem Fisheye-Objektiv Lensagon BF2M15520 verwendet. Die UI-1226LE-M ist eine äußerst kompakte Ein-Platinen-Kamera mit Aptina CMOS-Sensor in Wide VGA-Auflösung (752x480 Pixel). Diese Kamera kann bis zu 87 Bilder pro Sekunde liefern und die Belichtungszeit kann bis auf 80 μs reduziert werden, so dass auch für bewegliche Objekte ausreichend scharfe Fotos entstehen. Durch Verwendung der weit verbreiteten USB 2.0-Technologie ist die Anbindung an verschiedenste

Systeme problemlos möglich. Um die Integration zu erleichtern, sind die Programmiersprachen C, C++, Microsoft .NET und Visual Basic unterstützt.

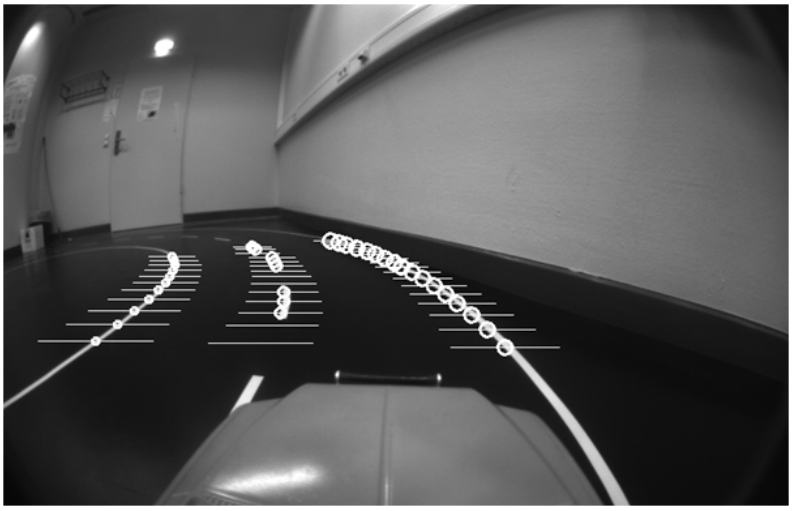
Abbildung 5.1: Onyx Fahrzeug

Als Steuerungselektronik werden drei ARM 7 Prozessoren mit entsprechenden IO-Plattinen für die Sensordatenverarbeitung verwendet. Die eigentliche Rechenleistung stellt ein Acer Aspire One Subnotebook zur Verfügung, welches mit einem Intel Atom Prozessor mit 1.6 GHz ausgestattet ist. Die Kamera und die ARM-Prozessoren sind jeweils über USB mit dem Notebook verbunden. Auf dem Acer Aspire One Subnotebook ist ein Debian Linux mit dem RT-Preempt Realtime-Kernel eingerichtet.

Die C++ Systemsoftware FAUSTcore stellt abstrakte Basiskomponenten zur Implementierung von Treibern und Tasks (FAUSTplugins) zur Verfügung [Jenning 2009]. Zudem ermöglicht der FAUSTcore die bequeme Steuerung der Anwendung durch Parametereinstellungen über eine Web-Oberfläche. Der Task-Scheduler basiert auf der Subsumption Architektur [Brooks 1986]. In der softwaretechnischen Umsetzung von Algorithmen für die Bildverarbeitung und das maschinelle Lernen wird die OpenCV Bibliothek [Bradski 2000] verwendet.

Der FAUSTplugin Polaris [Jenning 2009] analysiert die Kamerabilder und bestimmt die Fahrspurgeometrie. Da Bildverarbeitung in der Regel sehr aufwendig ist, werden nur ein paar Betrachtungsbereiche pro Fahrspurmarkierung ausgewertet (vgl. Abbildung 5.2). Im Englischen wird der Betrachtungsbereich als Region Of Interest (ROI) bezeichnet. Um die Kanten zu detektieren wird jeder Betrachtungsbereich des Bildes mit dem horizontalen Sobel-Operator gefiltert. Nach der Faltung mit dem Sobel-Operator wird eine Schwellwert-Funktion zur Erkennung der größten Kanten angewandt. Anschließend wird maximal ein Fahrspurmarkierungspunkt pro Betrachtungsbereich extrahiert.

(a) Originales Kamerabild



(b) Erkennung von Fahrspurpunkten

Abbildung 5.2: Fahrspurerkennung mittels Polaris

Des Weiteren werden die Bildpunkte entzehrt und durch eine projektive Transformation in das Fahrzeugkoordinatensystem übertragen. Der Ursprung des Fahrzeugkoordinatensystems wird von Polaris direkt mittig vor dem Fahrzeug platziert, wobei die Fahrzeuglängsachse auf der Abzisse liegt. Fahrspurpunkte, die rechts for dem Fahrzeug liegen, werden positive y-Werte zugewiesen. Zur Approximation der Fahrspurmarkierungen werden Polynome zweiten Grades verwendet. Um die Parameter der Polynome zu approximieren wird die Methode der kleinsten Quadrate angewendet. Auf diese Weise liefert der Algorithmus Polaris Polynome, welche die Fahrbahnmarkierungen im Fahrzeugkoordinatensystem beschreiben.

5.1.2 TORCS Simulator

TORCS (The Open Racing Car Simulator) ist eine freie Simulation für Autorennen [Espié u. a. 2006]. TORCS ist sowohl ein normales Spiel als auch eine Softwareplattform für AI-Algorithmen. Es läuft unter Linux, FreeBSD, MacOS X und Windows. Die Simulation verfügt über mehrere Strecken, Rennwagen und Gegner. Bei den Rennwagen werden Aerodynamik, Kollisionsschaden, Rad- und Reifeneigenschaften simuliert.

Abbildung 5.3: TORCS

TORCS bietet die Möglichkeit das Fahrverhalten über Softwaremodule in C oder C++ zu implementieren [Wymann]. Diese Module werden als Robots bezeichnet. Die Fahrspurgeometrie wird im Weltkoordinatensystem durch mehrere Segmente beschrieben. Die Fahrzeugposition und die Fahrtrichtung werden ebenfalls im Weltkoordinatensystem gegeben, sodass die Fahrzeuglage bezüglich der Fahrbahn, die Abweichung zur Sollspur und weitere Kenngrößen ermittelt werden können. Der Druck von Gas- und Bremspedal kann im Bereich von 0 bis 1.0 gesteuert werden und für die Lenkung sind Stellwerte von -1.0 bis 1.0 zugelassen. Außerdem ermöglicht ein Streckeneditor die Erstellung eigener Strecken. Somit ist TORCS sehr wertvoll zur Analyse von AI-Algorithmen.

5.2 Softwarearchitekture

Das FQI-Spurführungsverfahren wurde in der Programmiersprache C++ implementiert. In Abbildung 5.4 ist die Softwarearchitektur dargestellt.

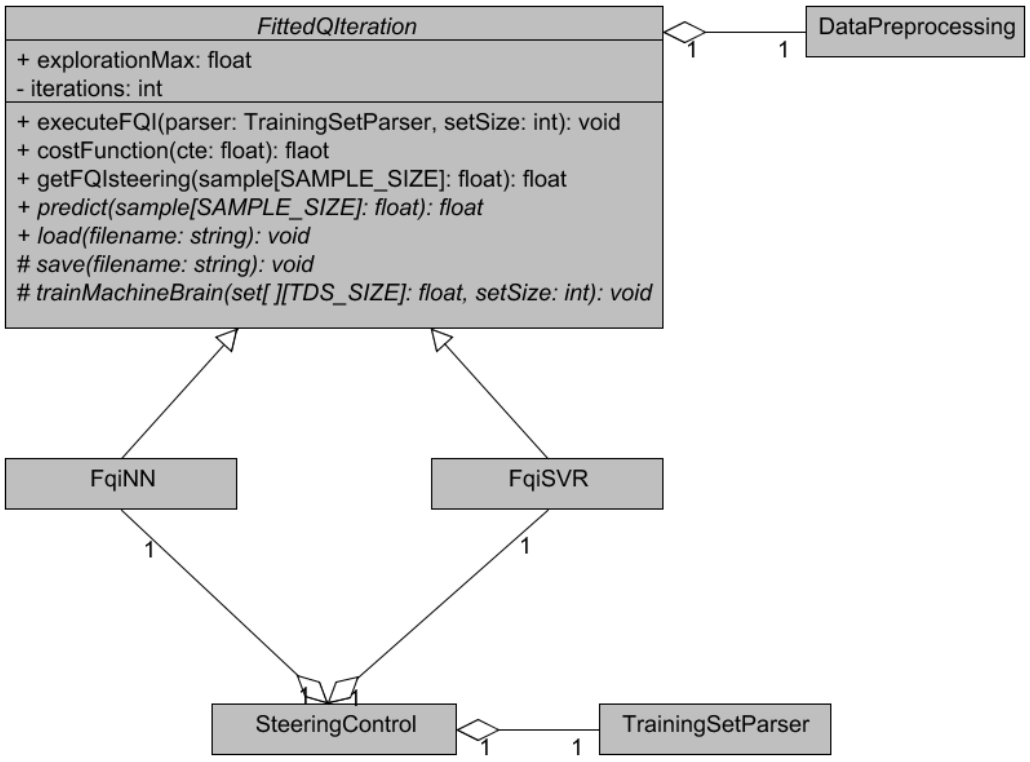


Abbildung 5.4: Softwarearchitekture

Die Implementierungen für das Einsatzfahrzeug Onyx und die Simulationsumgebung TORCS unterscheiden sich nur in der Klasse SteeringControl, welche die Steuerungsbefehle kapselt und ein FAUSTplugin bzw. ein TORCS-Robot darstellt. In beiden Fällen ruft ein Scheduler regelmäßig eine SteeringControl-Funktion zur Steuerung der Aktorik auf. SteeringControl wird in folgenden Modi getrieben: PURE_PURSUIT, USE_FQI, TRAIN_FQI und ANALYSE_FQI. Im ersten Modus wird der Stellwert für die Lenkung mittels Pure Pursuit ermittelt. Zur Bestimmung von dem besten Lenkungsstellwert im USE_FQI-Modus werden die Q-Werte aller möglichen Aktionen approximiert. Zu diesem Zweck kommt ein trainierter FQI-Regressor zum Einsatz. Das Ziel von TRAIN_FQI-Modus ist das Sammeln von Trainingsdaten für den FQI-Algorithmus. Die Klasse TrainingSetParser verwaltet das Speichern und das Laden von Trainingsdaten. Wenn die gewünschte Anzahl von Trainingsdaten einer Episode erreich ist,

wird FQI ausgeführt. Nach dieser Ausführung werden der trainierte Regressor und seine Skalierungswerte in Dateien gespeichert. Anschließend wird der ANALYSE_FQI-Modus für eine Testrunde geschaltet. Im Verlauf dieser Testrunde wird der trainierte Regressor zur Approximation der Q-Funktion eingesetzt und das resultierende Fahrverhalten wird analysiert. Es wird die durchschnittliche Abweichung zur Sollspur berechnet. Zusätzlich wird ermittelt wie lange das Fahrzeug die gewünschte Abweichung hält, sodass nach jeder Trainingsepisode der trainierte Regressor ausgewertet wird. Nach der Testrunde folgt die nächste Episode und es wird wieder TRAIN_FQI aktiviert.

Der FQI-Algorithmus, die Bewertungsfunktion und die Ermittlung der optimalen Lenkungsstellwertes mittels des FQI-Regressors werden in der abstrakten Klasse FittedQIteraion implementiert. Bevor der FQI-Regressor trainiert werden darf, müssen die von FQI generierten Trainingsdaten skaliert werden. Die DataPreprocessing-Klasse erledigt diese Aufgabe.

Die Klassen FqiNN und FqiSVR, welche das Trainieren und die Verwendung von neuronalen Netzen bzw. Support-Vektor-Regression verwalten, sind von der FittedQIteration abgeleitet. In der softwaretechnischen Umsetzung des neuronalen Netzes wurde die OpenCV-Bibliothek [Bradski 2000] verwendet. Das neuronale Netz besteht aus neun Eingabe-Neuronen (acht Zustandsvariablen plus eine Aktion), zwei verdeckten Schichten mit jeweils fünf Neuronen und einem Ausgabe-Neuron. Zur Initialisierung von den Verbindungsgewichten setzt OpenCV standardmäßig den Nguyen-Widrow Algorithmus ein [Nguyen und Widrow 1990]. Zum Trainieren wurde der Lernalgorithmus Rprop ausgewählt. Die neuronalen Netze liefern eine sehr genaue Approximation, wenn der Abruchskriterium-Parameter $0 < EPS < 0,00001$ gewählt wird. Die LIBSVM-Bibliothek [Chang und Lin 2011] wurde zur Support-Vektor-Regression (SVR) angewandt. Eine nu-SVR mit einer Radiale-Basis-Funktion als Kernelfunktion wurde eingesetzt. Die SVR wurde defaultmäßig parametrisiert (vgl. Tabelle 5.1).

SVR-Parameter		KNN-Parameter	
svm_type	NU_SVR	train_method	RPROP
kernel_type	RBF	rp_dw0	1,0
gamma	0,125	TERMCRIT_ITER	1000
eps	0,1	TERMCRIT_EPS	0,000001
C	1,0		
nu	0,5		
cache_size	256		

Tabelle 5.1: SVR- und KNN-Parametrisierung

5.3 Datenskalierung

FQI generiert einen Datensatz zum Trainieren eines Regressors, welcher die Q-Funktion approximiert. Vor dem Trainieren wird dieser Datensatz im Bereich [0,1; 0,9] linear skaliert. Die Auswahl von Bereichsgrenzen hat keine Auswirkung auf der SVR-Approximationsleistung [Chang und Lin 2011]. OpenCV skaliert standartmäßig den Trainingsdatensatz für den entsprechenden KNN-Trainingsalgorithmus [Bradski 2000]. Der skalierte Wert x_s einer Stichprobenvariable x des Trainingsdatensatzes P wird wie folgt ermittelt:

$$x_s = \frac{(x - x_{min})(0,9 - 0,1)}{x_{max} - x_{min}} + 0,1. \tag{5.1}$$

x_{min} und x_{max} geben den kleinsten bzw. den größten Wert der zu skalierenden Variable in dem Trainingsdatensatz P an. Wenn es erforderlich ist, werden diese Variablen zurück im originalen Bereich skaliert (vgl. Gleichung 5.2).

$$x = \frac{(x_s - 0,1)(x_{max} - x_{min})}{0,9 - 0,1} + x_{min} \tag{5.2}$$

Zur Auswertung von Aktionen anhand des FQI-Regressors werden Zustand-Aktions-Paare und deren approximierten Bewertungen skaliert bzw. zurückskaliert. Zu diesem Zweck ist die Speicherung von x_{min} und x_{max} aller Stichprobenvariablen für jeden FQI-Regressor erforderlich.

5.4 Ermittlung der Sollspurabweichung für das Fahrzeug Onyx

Die Abweichung zur Sollspur cte wird zur Auswertung der Zustand-Aktions-Paare im prädiktiven NFQ-Spurführungsverfahren verwendet. Die kamerabasierte Spurerkennung Polaris liefert Polynome zweiten Grades, um die Fahrspurmarkierungen für das FAUST-Fahrzeug Onyx zu beschreiben. Die Fahrspurmitte wird als optimale Sollspur definiert und wird ebenfalls durch ein Polynom $p(x) = ax^2 + bx + c$ approximiert. Als Approximation der Sollspur wird das Polaris-Polynom, welches die rechte Fahrspurmarkierung beschreibt, eingesetzt, wobei die Hälfte der Fahrspurbreite $\frac{1}{2}s_w$ von der Koeffizient c dieses Polynoms abgezogen wird.

Der Koordinatenursprung wird von Polaris direkt mittig vor dem Fahrzeug platziert. cte wurde als der kleinste Abstand von Mitte der Fahrzeugvorderachse zum Sollspur definert (vgl. Abbildung 4.1). Im Fahrzeugkoordinatensystem ist die Mitte der Fahrzeugvorderachse als der Punkt

$(-f,0)$ definiert, wobei f der vordere Fahrzeugüberhang ist. Der Abstand r von $(-f,0)$ zu einem Punkt eines Polynoms zweiten Grades $p(x)$ ergibt sich aus:

$$r = \sqrt{(x--f)^2 + (p(x)-0)^2} \tag{5.3}$$

$$r = \sqrt{(x+f)^2 + p(x)^2} \tag{5.4}$$

und folglich:

$$r^2 = (x+f)^2 + (ax^2+bx+c)^2. \tag{5.5}$$

Zur Bestimmung von cte wird der Punkt $(x, p(x))$, welcher am nächsten von $(-f,0)$ liegt, gesucht. Zu diesem Zweck sind die Nullstellen der ersten Funktionsableitung $\frac{\partial r^2}{\partial x}$ zu ermitteln. Diese Nullstellen sind Extremwerte der Funktion r^2 (vgl. Abbildung 5.5). Die Funktionen r^2 und $|r|$ haben dieselben Minima. Folglich kann r_{min} bestimmt werden, indem die Extremwerte von r^2 untersucht werden.

Die erste Ableitung der Funktion r^2 beträgt:

$$\frac{\partial r^2}{\partial x} = 2x + 2f + 2(ax^2+bx+c)(2ax+b) \tag{5.6}$$

$$\frac{\partial r^2}{\partial x} = 4a^2x^3 + 6abx^2 + 2(2ac+b^2+1)x + 2bc + 2f \tag{5.7}$$

Zur Bestimmung von Nullstellen der kubischen Gleichung $\frac{\partial r^2}{\partial x}$ kommt die Cardano-Methode zum Einsatz [Ramana 2006].

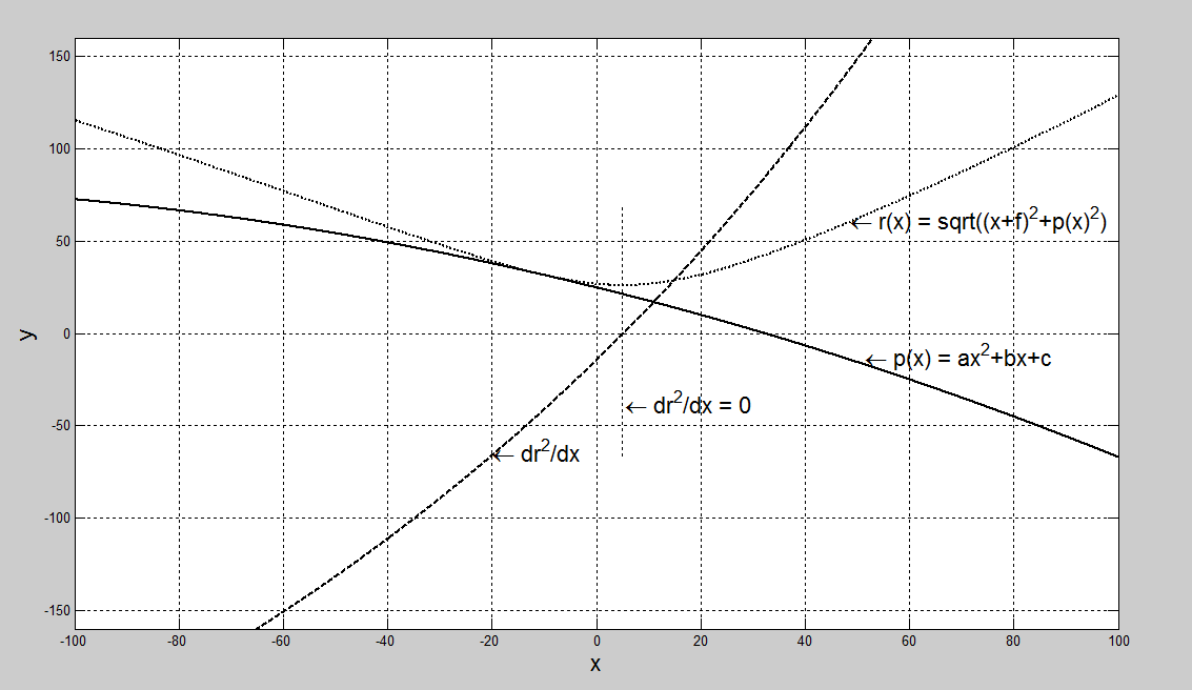


Abbildung 5.5: Ermittlung von Sollspurabweichung bei f=10

5.5 Lenkwinkelberechnung mittels Inkrementalgeber

Der aktuelle Lenkwinkel δ wird als Zustandsvariable im prädiktiven NFQ-Spurführungsverfahren verwendet. Die Berechnung dieses Winkels erfolgt mittels Inkrementalgeber an den Vorderrädern des Fahrzeuges Onyx. Es ist zu beachten, dass Rutschen oder Durchdrehen der Räder nicht erfasst werden, was die Lenkwinkelmessung beeinträchtigen kann. Die Inkrementalgeber liefern die gefahrenen Wegstrecken der Vorderräder. Zur Berechnung des aktuellen Lenkwinkels werden die relativen Wegstrecken s_L und s_R verwendet, die durch

$$s_L = R_L \theta \tag{5.8}$$

$$s_R = (R_L + b)\theta \tag{5.9}$$

berechnet werden, wobei R_L der Kurvenradius des linken Rades ist, b den Abstand zwischen der Vorderräder, und der Drehwinkel durch θ dargestellt wird (vgl. Abbildung 5.6). Wenn beide Gleichungen nach θ gelöst und gleichgesetzt werden, gilt

$$\frac{s_L}{R_L} = \frac{s_R}{R_L + b} \tag{5.10}$$

und folglich

$$R_L = \frac{s_L b}{s_R - s_L}. \tag{5.11}$$

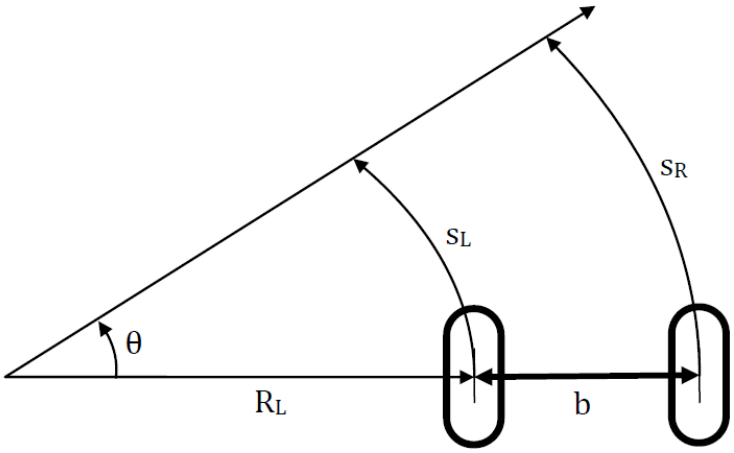


Abbildung 5.6: Trajektorien der Räder in einer Kurve

Der mittlere Kurvenradius ergibt sich aus

$$R = R_L + \frac{1}{2}b. \tag{5.12}$$

Der aktuelle Lenkwinkel δ kann dann mittels Gleichung 1.1 berechnet werden.

6 Experimentelle Auswertung

In diesem Kapitel werden die experimentellen Auswertungen dargestellt. In Abschnitt 6.1 wird die Approximationsleistung von künstlichen neuronalen Netzen und Support-Vektor-Regression verglichen. In Abschnitt 6.2 wird die Auswirkung von Lenkungstotzeit und Zielentfernung auf Pure Pursuit dargestellt. Der Versuch zur Messung der Lenkungstotzeit von dem FAUST-Fahrzeug Onyx wird in Abschnitt 6.3 beschrieben. Anschließend werden Lenkungsverhalten von Pure Pursuit und prädiktiven NFQ-Spurführung in der Testumgebung verglichen (vgl. Abschnitt 6.4).

6.1 Approximationsauswertung von KNN und SVR

FQI setzt eine Approximationsmethode zur Regression der Q-Werte von Zustand-Aktions-Paaren. Zu diesem Zweck ist eine Methode erforderlich, welche eine gute Approximation für hochdimensionale nichtlineare Funktionen leistet, weil der Fahrzeugzustand durch acht Variablen beschrieben wird. In diesem Abschnitt wird die Approximationsleistung von künstlichen neuronalen Netzen und Support-Vektor-Regression untersucht. In Abschnitt 5.2 wurden bereits deren Parametrisierungen beschrieben.

Die Größe eines neuronalen Netzes spielt eine wichtige Rolle bei der Approximation. Zu viele Neuronen in den verdeckten Schichten führen zu Überanpassung und Netze mit wenigen Neuronen generalisieren zu stark (vgl. Abschnitt 2.3.1). Zur Bestimmung der optimalen Netzgröße werden am häufigsten neuronale Netze mit unterschiedlicher Anzahl von Neuronen mit einem Trainingsdatensatz trainiert und diejenige, welche am besten einem Validierungsdatensatz generalisieren, werden eingesetzt [Alpaydin 2004].

Außerdem bestimmt der Trainingsalgorithmus Rprop wie die Verbindungsgewichte aktualisiert werden, indem die Trainingsdaten in einer zufälligen Reihenfolge verarbeitet werden. Aus diesem Grund ist das Trainieren eines neuronalen Netzes nicht deterministisch und folglich ist die Approximationsleistung vom mehreren Netzen mit gleicher Struktur unterschiedlich. Deshalb ist es vorteilhaft bei der Auswertung von Netzgrößen mehrere Netze mit gleicher Struktur zu trainieren. SVR bietet den Vorteil, dass das Trainieren deterministisch ist.

In Tabelle 6.1 ist die Untersuchung der Approximation von SVR und neuronalen Netzen mit

zwei verdeckten Schichten mit jeweils N Neuronen dargestellt. Die Zahlen in der Tabelle entsprechen dem prozentuellen Anteil von Daten, welche bei der Approximation einen absoluten Fehler $\varepsilon < 0,1$ aufweisen, wobei

$$\varepsilon = |KNN(s_l, a_l) - o_l| \tag{6.1}$$

Dieser Auswertungskriterium wurde bereits in Abschnitt 4.2.4 formal definiert. Zur Auswertung wurde in TORCS ein Trainingsdatensatz der Größe $|T| = 5000$ generiert. Aus diesen Daten wurde ein Datensatz Tr zum Trainieren und ein Datensatz Te zur Validierung der Approximationsleistung wie folgt gebildet:

$$Tr = \{p_l \mid l = 1 \ldots |T| \wedge p_l \in T \wedge l \not\equiv 0 \bmod 5\}, \tag{6.2}$$

$$Te = \{p_l \mid l = 1 \ldots |T| \wedge p_l \in T \wedge p_l \notin Tr\}. \tag{6.3}$$

Auf diese Weise werden 80% der Daten zum Trainieren des Regressors verwendet und der Rest zur Validierung der Approximation. Bei den neuronalen Netzen werden für jeden Datensatz die maximalen max und die durchschnittlichen avg prozentuellen Anteile eingegeben, weil pro Netzgröße zehn Netze ausgewertet wurden. Außerdem wird bei dem Regressor, welcher den Trainingsdatensatz am besten approximiert, auch die Auswertung für die Testdaten in Klammern eingegeben.

Es lässt sich leicht erkennen, dass bei $N = 5$ durchschnitlich beste Ergebnisse erzielt werden. Außerdem liefiert das beste neuronale Netz dieser Größe eine vergleichsweise sehr gute Approximation. Des Weiteren ist es vorteilhafter, wenn mehrere Netze einer Netzstruktur mit den Trainingsdaten trainiert werden und dasjenige, welches die beste Approximationsleistung liefert, eingesetzt wird. Der Grund dafür ist, dass nicht jedes trainiertes Netz eine gute Approximation leistet. Das Trainieren von Support-Vektor-Regression ist deterministisch, aber im Vergleich zu neuronalen Netzen ist die SVR-Approximationsleistung deutlich schlechter.

Neuronale Netze				
N	Testdaten		Trainingsdaten	
	avg	max	avg	$max(test)$
2	65.1	75.0	65.7	75.6 (75.0)
3	75.5	82.0	77.3	83.5 (81.5)
4	73.8	85.2	76.1	86.5 (85.2)
5	80.3	86.6	82.4	88.5 (86.6)
6	71.9	83.1	73.9	85.3 (83.1)
7	76.1	87.5	78.2	89.4 (87.5)
8	66.1	84.9	68.7	88.6 (84.9)
9	71.3	82.7	74.1	86.2 (82.7)
10	68.0	86.8	70.7	89.5 (86.4)
11	71.1	85.6	74.8	88.8 (84.4)
12	66.3	88.8	69.6	90.5 (88.8)
13	67.8	88.2	71.7	90.9 (85.8)
14	57.3	72.3	61.6	78.5 (72.3)
15	70.9	85.1	75.2	89.7 (85.1)
16	68.2	80.0	73.2	85.8 (80.0)
Support-Vektor-Regression				
-	-	42,1	-	44,2 (42,1)

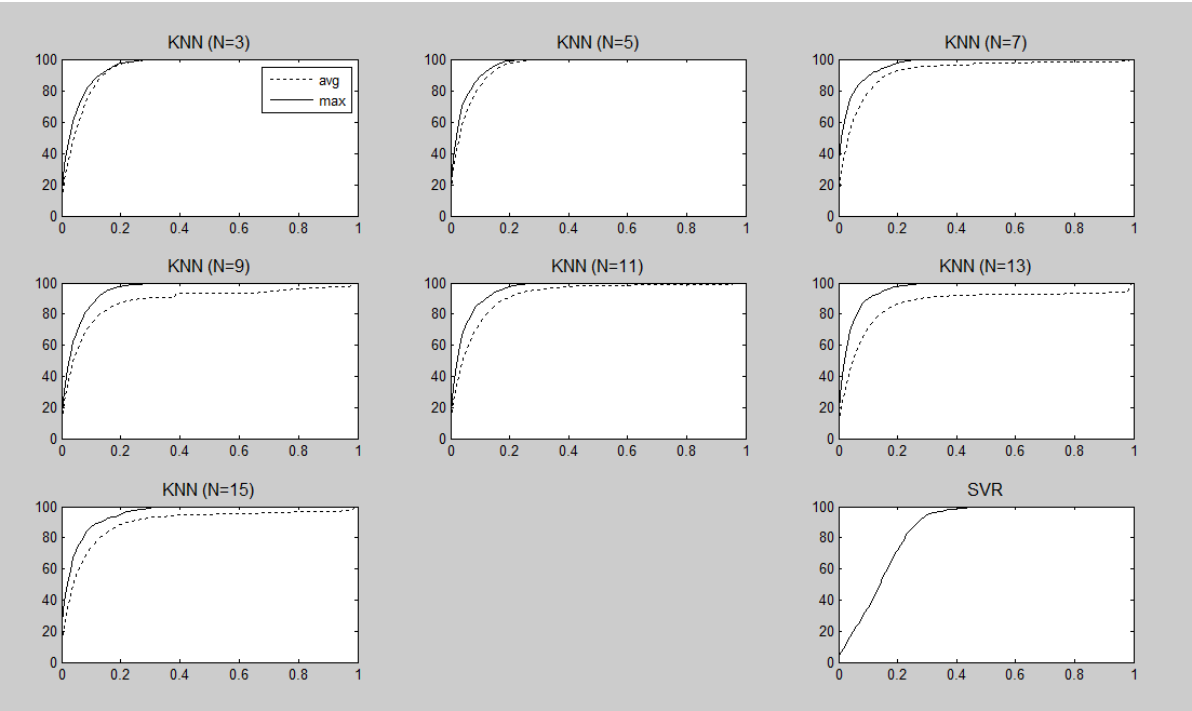
Tabelle 6.1: Untersuchung der Approximation von SVR und neuronalen Netzen mit zwei verdeckten Schichten mit jeweils N Neuronen

In Abbildung 6.1 ist die Analyse der Approximation von SVR und neuronalen Netzen unterschiedlicher Größe grafisch dargestellt. Auf der x-Achse ist ε im Bereich [0;1] eingetragen und auf der y-Achse ist die Approximationsauswertung y_{app} dargestellt, wobei

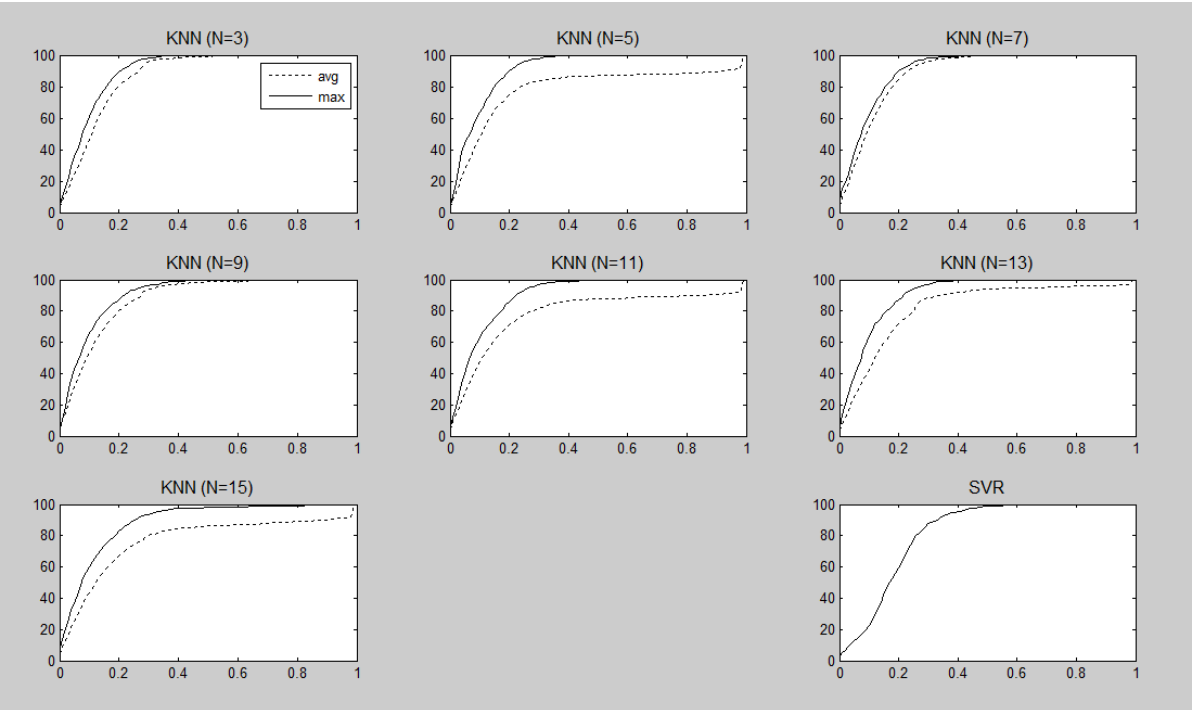
$$y_{app} = \frac{100 * |Q|}{|Te|}, \tag{6.4}$$

und

$$Q = \{p_l \mid l = 1 \ldots |Te| \wedge p_l \in Te \wedge |KNN(s_l, a_l) - o_l| < \varepsilon\}. \tag{6.5}$$



(a) TORCS-Trainingsdatensatz



(b) Onyx-Trainingsdatensatz

Abbildung 6.1: Analyse der Approximation von SVR und neuronalen Netzen unterschiedlicher Größe

Außerdem ist die Approximationsauswertung sowohl für TORCS- als auch für Onyx-Trainingsdaten dargestellt. Es ist ersichtlich, dass KNN und SVR eine ungenauere Approximation für die Onyx-Daten liefern. Das ist auf die Tatsache zurückzuführen, dass beim Trainieren mit dem Fahrzeug Onyx Datenrauschen entsteht.

Wie bereits in Abschnitt 5.5 beschrieben, wird die Messung des aktuellen Lenkwinkels mittels Inkrementalgeber realisiert. Eine fehlerhafte Messung aufgrund Durchdrehen oder Rutschen der Räder führt zum Datenrauschen. Außerdem beeinträchtigt die Ungenauigkeit der Inkrementalgeber ebenfalls die Messung.

Ein anderer Grund für das Rauschen ist eine teilweise fehlerhafte Identifizierung der Fahrspurgeometrie. Dadurch werden Variablen des Fahrzeugzustandes falsch ermittelt. Weiterhin fehlt eine Synchronisierung zwischen der Spurerkennungstask Polaris und dem Kameratreiber, welcher alle 20 ms das aktuelle Kamerabild in einem Datenkontainer speichert. Aus diesem Grund variiert die Existenzzeit der Bilder, welche zur Identifizierung der Fahrspurgeometrie untersucht werden, um 20 ms. Dadurch werden ebebfalls Variablen des Fahrzeugzustandes falsch ermittelt und somit ensteht eine teilweise fehlerhafte Bewertung der Zustand-Aktions-Paare.

6.2 Pure Pursuit Lenkungsverhalten

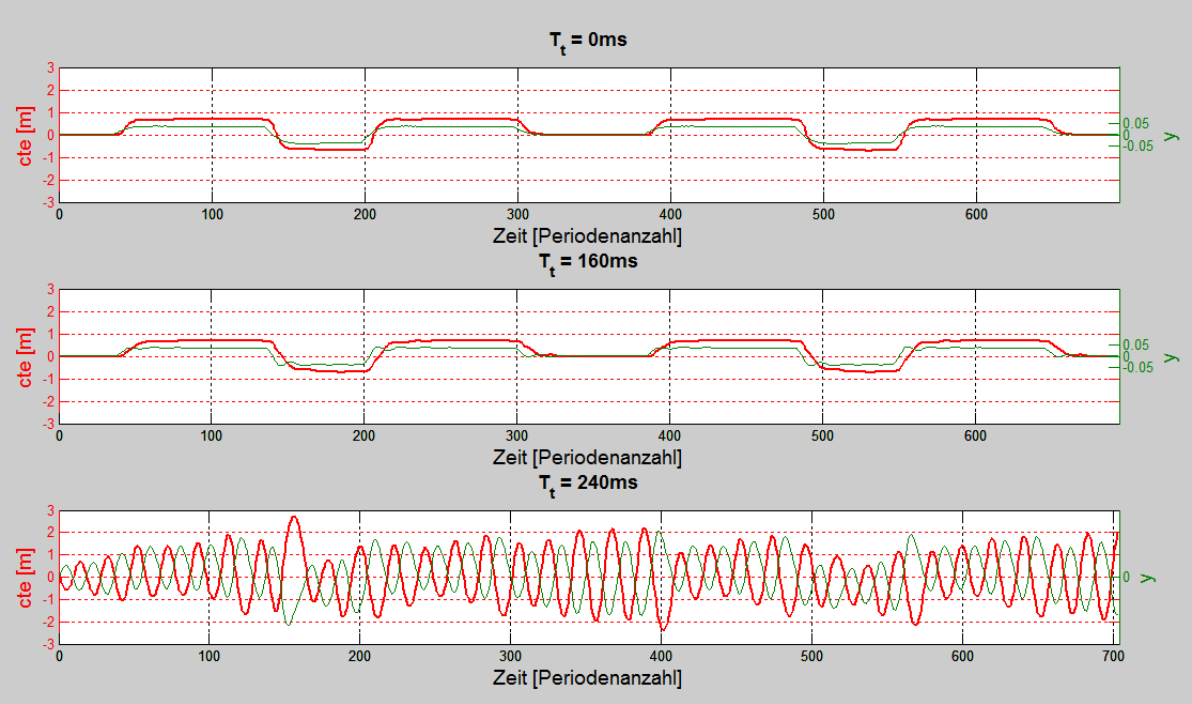


Abbildung 6.2: Verhalten von Pure Pursuit bei unterschiedlichen Totzeiten mit D = 20 m und v=100 km/h

In diesem Abschnitt werden Auswirkungen von Lenkungstotzeit T_t und Zielentfernung D auf dem Lenkungsverhalten von Pure Pursuit untersucht. Die Analyse wurde in der TORCS-Simulationsumgebung durchgeführt. In Abbildung 6.2 sind die Verläufe von Lenkungsstellwert y und Sollspurabweichung cte innerhalb einer Runde der Teststrecke E-Track 5 bei $T_t = 0$, $T_t = 0,16$ und $T_t = 0,24$ s dargestellt. Auf der x-Achse ist die Zeit eingetragen.

Bai $T_t <= 0,16$ s ist eine Beeinträchtigung des Lenkungsverhalten kaum zu bemerken. Jedoch bei einer Totzeit von 0,24 s werden auschlaggebende Lenkungsschwingungen ausgelöst und auf diese Weise bleibt die Fahrt ununterbrochen instabil.

Weiterhin wurde die Auswirkung von der Entfernung vom Fahrzeug zum Ziel untersucht. Diese Auswirkung wird in Abbildung 6.3 dargestellt.

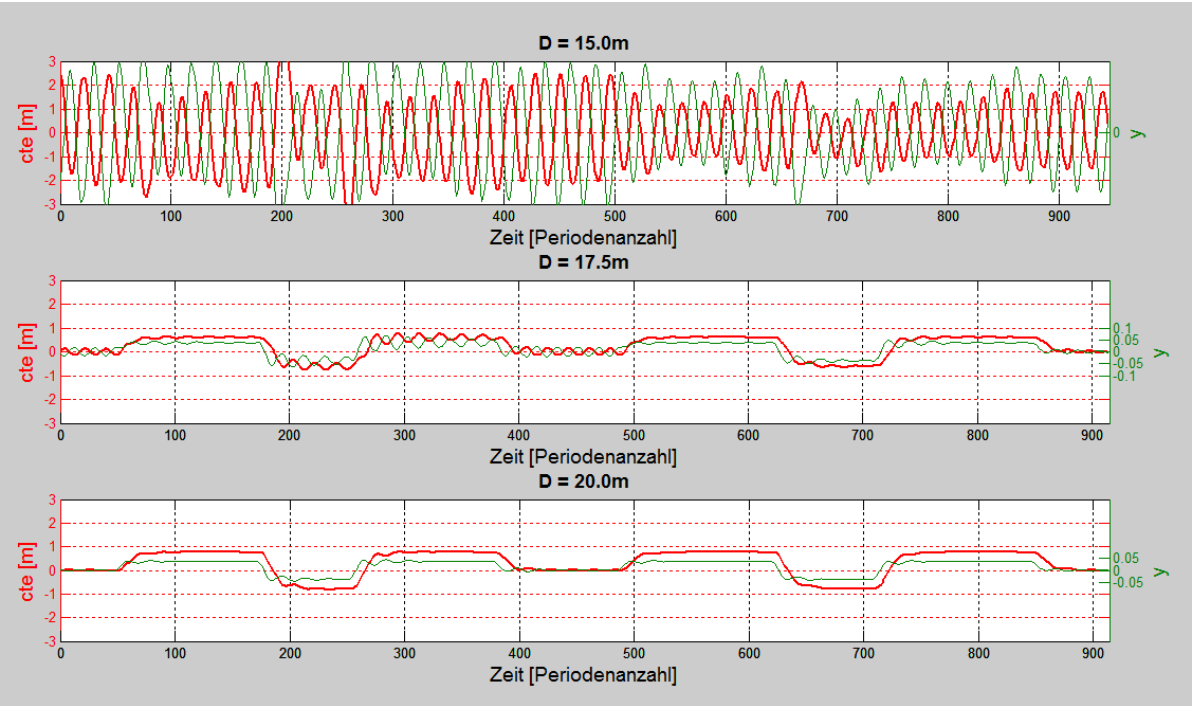


Abbildung 6.3: Verhalten von Pure Pursuit bei unterschiedlichen Zielentfernungen mit Tt = 0,24 s und v = 80 km/h

Es ist leicht zu erkennen, dass die Spurführung mit $D = 20\,\text{m}$ im Vergleich zu Zielentfernungen von 15 und 17,5 m die beste Ergebnisse erzielt bei $T_t = 0,24\,\text{s}$ und $v = 80\,\text{km/h}$. Jedoch wenn die Geschwindigkeit um 20 km/h erhöht wird, ist die Spurführung mit $D = 20\,\text{m}$ ununterbrochen instabil (vgl. Abbildung 6.4). In diesem Fall sind wieder höhere Zielentfernungen zu bevorzugen. Aus diesem Grund ist eine dynamische Zielentfernung für Pure Pursuit vorteilhaft. Dieser Ansatz verbessert die Stabilität der Fahrt, aber es ist auch zu erkennen, dass bei höheren Zielentfernungen und höheren Geschwindigkeiten die Abweichung zur Sollspur größer wird. Somit ist eine präzise Fahrt nicht zu erreichen.

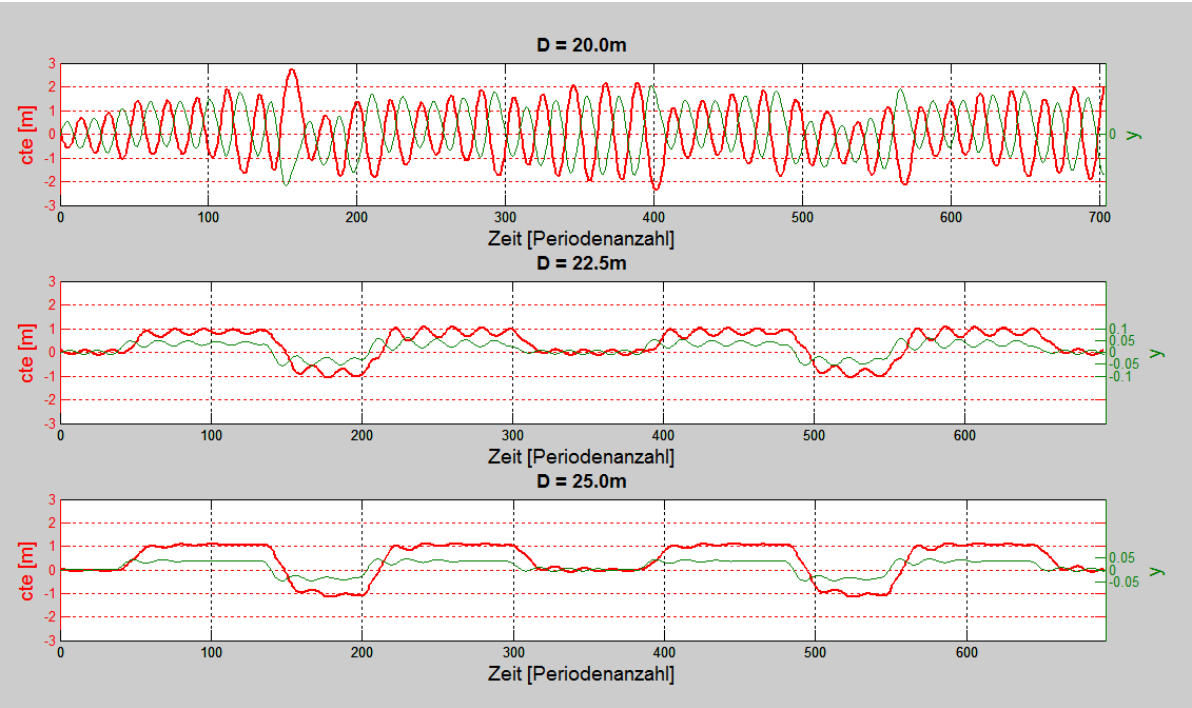


Abbildung 6.4: Verhalten von Pure Pursuit bei unterschiedlichen Zielentfernungen mit Tt = 0,24 s und v = 100 km/h

6.3 Messung der Lenkungstotzeit für das FAUST-Fahrzeug Onyx

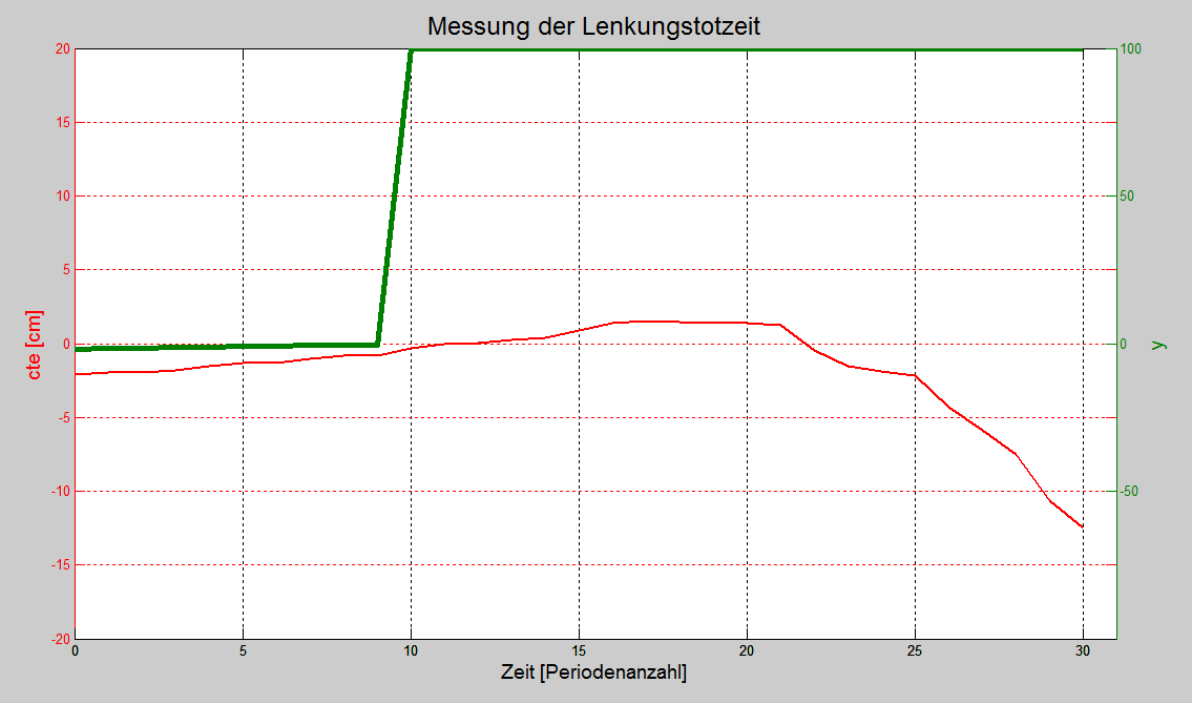


Abbildung 6.5: Messung der Lenkungstotzeit

In diesem Abschnitt wird der Versuch zur Messung der Lenkungstotzeit von dem FAUST-Fahrzeug Onyx beschrieben. Zur Ermittlung der Totzeit kommt die kamerabasierte Spurerkennung Polaris zum Einsatz (vgl. Abschnitt 5.1.1). Ursprünglich wird das Fahrzeug auf einer geraden Fahrspur gestellt. Danach fährt das Fahrzeug autonom einige Meter entlang der Fahrspur bis zum einem definierten Zeitpunkt, wo die Lenkung nach rechts voll eingeschlagen wird. Während dem Versuch werden Abweichung zur Sollspur und Lenkungsstellwerte gespeichert.

Die Lenkungstotzeit wurde bei einer Abtastrate von 40 Hz gemessen. Somit dauert eine Periode 25 ms. Eine Geschwindigkeit von ca. 1 m/s. wurde beim Versuch gemessen. In Abbildung 6.5 sind die gespeicherte Daten dargestellt. Auf der x-Achse ist die Zeit in Periodenanzahl eingetragen. Aus der Abbildung lässt sich erkennen, dass die Lenkung ab der zehnten dargestellten Periode nach rechts voll eingeschlagen wird ($y = 100$). Diesbezüglich ändert sich die Sollspurabweichung cte erst nach zwölf Perioden. Folglich beträgt die Onyx-Lenkungstotzeit ca. 0,3 s.

6.4 Lenkungsverhalten des prädiktiven NFQ-Spurführungsverfahrens

In diesem Abschnitt wird die Auswertung des in dieser Arbeit entwickelten prädiktiven NFQ-Spurführungsverfahrens beschrieben. Das Verfahren wurde sowohl in der TORCS-Simulation als auch auf dem Onyx-Fahrzeug analysiert.

6.4.1 TORCS

Wie bereits in Abschnitt 1.2 beschrieben, ist die Lenkungstotzeit T_t für die Stabilität der Spurführung ausschlaggebend. Zur Auswertung des Verfahrens wurde in TORCS eine $T_t = 0,24\,\text{s}$ simuliert. Aus diesem Grund wurde eine Steuerungsfrequenz von 12,5 Hz eingesetzt (vgl. Abschnitt 4.2.1). Die Parametrisierung von Pure Pursuit und FQI ist in Tabelle 6.2 beschrieben.

Parameter	Wert
D	20 m
K_p	1,5
$iterations$	5
$episodeLen$	5000
$explMax$	0,05
cte_{set}	1 m
v_{min}	90 km/h
v_{max}	110 km/h

Tabelle 6.2: Parametrisierung von Pure Pursuit und FQI in der TORCS-Simulation

Unter diesen Umständen wird die Spurführung mittels Pure Pursuit schon bei der Geschwindigkeit $v = 100\,\text{km/h}$ sehr unstabil. Die Leistungen von Pure Pursuit und prädiktiven NFQ wurden bei dieser Geschwindigkeit verglichen. Die Verläufe von Lenkungsstellwert y und Sollspurabweichung cte innerhalb einer Runde der Teststrecke E-Track 5 sind in Abbildung 6.6 dargestellt. Auf der x-Achse ist die Zeit eingetragen.

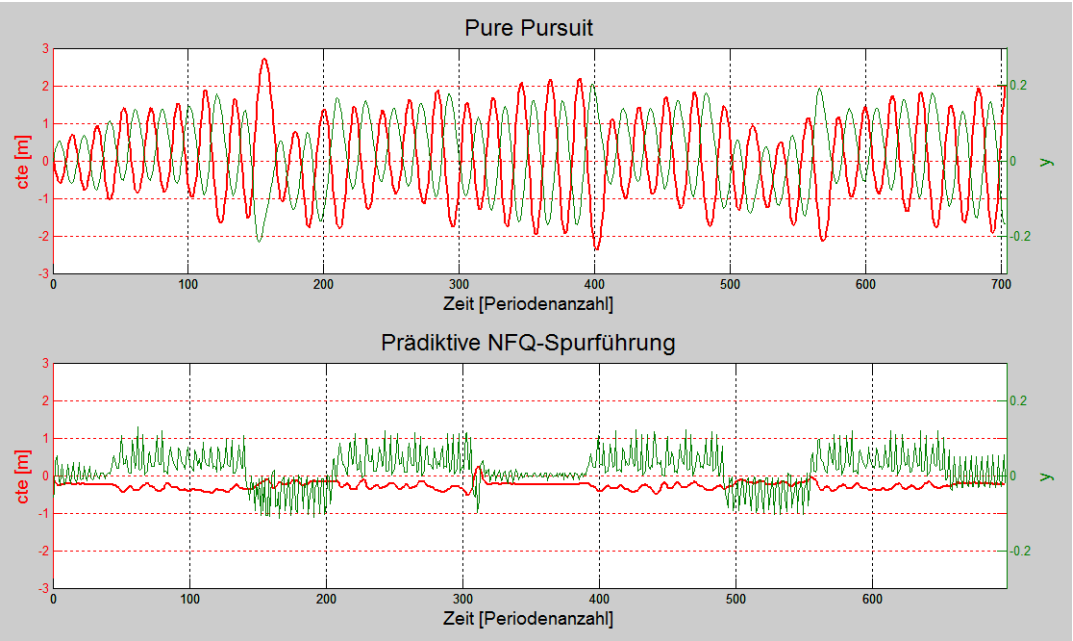


Abbildung 6.6: Verhalten von Pure Pursuit und prädiktiven NFQ in einer Runde der Teststrecke bei v = 100 km/h

Die Spurführung mittels Pure Pursuit generiert cte-Schwingungsamplituden bis zu fast 3 m. Nach einer Trainingsphase von ca. 7 Minuten erreicht die prädiktive NFQ-Spurführung ein zufriedenstellendes Lenkungsverhalten. Dieses Lenkungsverhalten hält erfolgreich die gewünschte Sollspurabweichung.

Der gelernte NFQ-Lenkungsregler wurde in drei Trainingsepisoden trainiert, wobei in der dritten Episode eine Geschwindigkeit von 150 km/h erreicht wurde. Bei dieser Geschwindigkeit wird ebenfalls eine präzise Spurführung erreicht (vgl Abbildung 6.7). Bei höherer Geschwindigkeit ist eine Trainingsphase nicht möglich, weil das Fahrzeug viel zu oft die zu verfolgende Fahrspur verlässt.

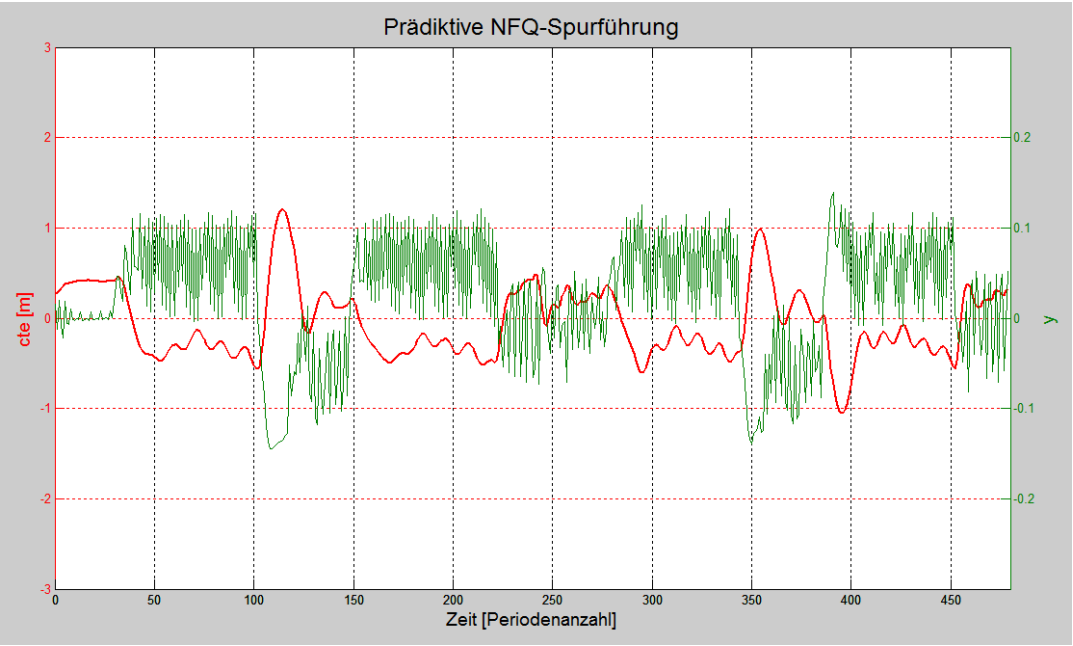


Abbildung 6.7: Verhalten von prädiktiven NFQ in einer Runde der Teststrecke bei v = 150 km/h

6.4.2 Onyx

Zur Analyse von Pure Pursuit auf dem Fahrzeug Onyx wurde eine Steuerfrequentz von 40 Hz verwendet. Die Lenkungstotzeit von Onyx beträgt 0,3 s (vgl. Abschnitt 6.3). Aus diesem Grund wurde die prädiktive NFQ-Spurführung bei einer Frequenz von 10 Hz ausgewertet (vgl. Abschnitt 4.2.1). Die Parametrisierung wird in Tabelle 6.3 beschrieben. Hier ist zu beachten, dass das Fahrzeug Onyx über keinen Tempomat verfügt. Deswegen kann es nicht sichergestellt werden, dass die gewünschte Geschwindigkeit beim Trainieren exakt gehalten wird.

Die Teststrecke ist gemäß [Carolo-Cup-Regelwerk 2011] regelkonform. Die Lenkungsverhalten von Pure Pursuit und prädiktiver NFQ-Spurführung bei der Strecke werden in Abbildung 6.8 verglichen. Daraus lässt sich erkennen, dass die prädiktive NFQ-Spurführung cte minimiert. Trotzdem wird $cte < cte_{set}$ nicht entlang der gesamten Teststrecke gehalten. Das ist auf die Tatsache zurückzuführen, dass aufgrund des Datenrauschens die von FQI trainierten neuronalen Netze unpräzise die Q-Werte der Zustand-Aktions-Paare approximieren.

Parameter	Wert
D	130 cm
K_p	250
$iterations$	5
$episodeLen$	3000
$explMax$	10
cte_{set}	5 cm
v_{min}	$\approx$ 180 cm/s
v_{max}	$\approx$ 200 cm/s

Tabelle 6.3: Parametrisierung von Pure Pursuit und FQI für Onyx

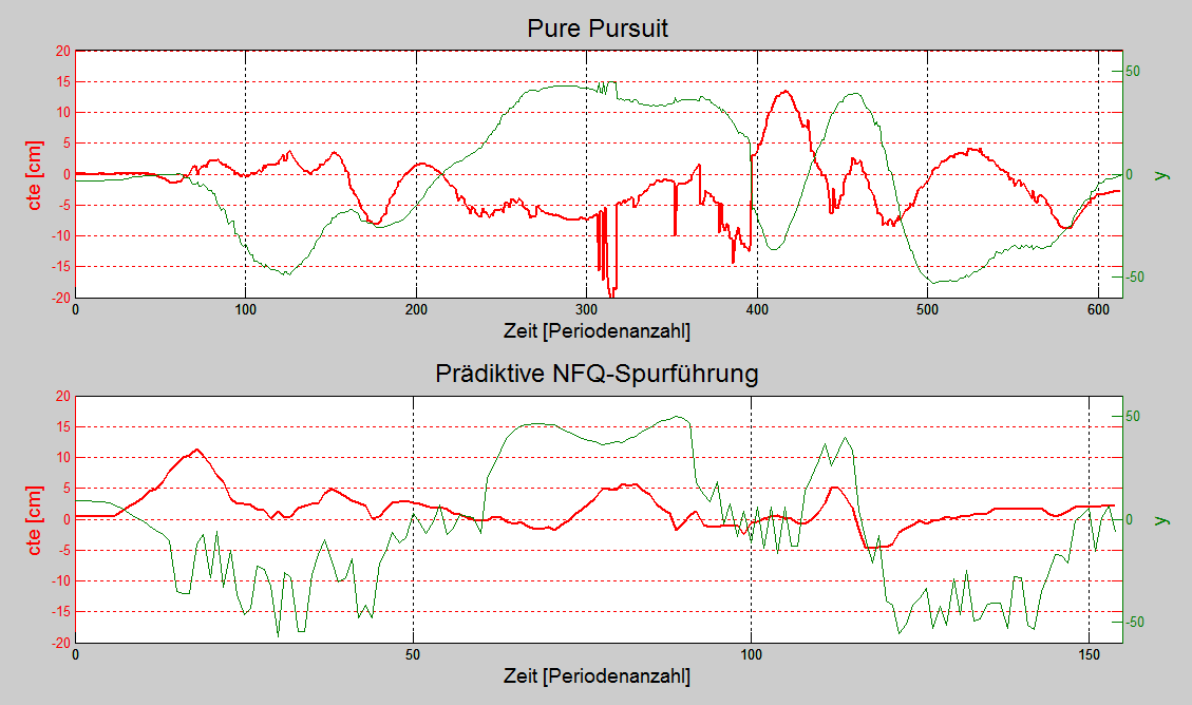


Abbildung 6.8: Verhalten von Pure Pursuit und prädiktiven NFQ in einer Runde der Teststrecke bei v = 2 m/s

7 Zusammenfassung

In dieser Arbeit wurde ein modellfreies Spurführungsverfahren entwickelt. Dieses Verfahren berücksichtigt die Fahrzeugkinematik, den Verlauf der Fahrspurgeometrie und die Lenkungsaktionen während der Lenkungstotzeit, um die optimale Lenkungsaktion für den aktuellen Fahrzeugzustand zu bestimmen.

Das in dieser Arbeit entwickelte prädiktive NFQ-Spurführungsverfahren basiert auf Q-Learning für kontinuierliche Zustands- und Aktionsräume. Das Verfahren setzt den Neural Fitted Q Iteration Algorithmus zum Trainieren eines neuronalen Netzes ein. Zur Ermittlung von der besten Lenkungsaktion bezüglich des Fahrzeugzustandes approximiert das trainierte neuronale Netz die Q-Werte der Zustand-Aktions-Paare.

Aus den Trainingsdaten generiert NFQ einen Datensatz zum Trainieren des neuronalen Netzes, wobei der Trainingsdatensatz aus Zustand-Aktions-Paaren und deren Bewertungen besteht. Ziel der Zustandsbeschreibung ist die Fahrzeugkinematik, den Verlauf der Fahrspurgeometrie und die Lenkungsaktionen während der Lenkungstotzeit zu erfassen. Ein quadratisches Polynom wird zur Approxmiation der Sollspur verwendet. Die Koeffizienten dieses Polynoms zusammen mit dem aktuellen Lenkwinkel und der Geschwindigkeit beschreiben die Fahrzeugkinematik. Weiterhin wird der Fahrzeugzustand durch die Lenkungsaktionen, welche aufgrund der Lenkungstotzeit noch nicht das Fahrverhalten beeinflusst haben, erweitert. Auf diese Weise wird nicht nur eine Aktion für den entsprechenden kinematischen Zustand bewertet, sondern eine Folge von Lenkungsaktionen. Das ermöglicht die Bestimmung der besten Lenkungsaktion unter Berücksichtigung der Totzeit.

Das Lenkungsverhalten des Verfahrens wurde in der Simulationsumgebung TORCS und auf dem FAUST-Fahrzeug Onyx mit der geometrischen Methode Pure Pursuit verglichen. In Torcs erzielt die prädiktive NFQ-Spurführung deutlich bessere Ergebnisse. Auf dem Fahrzeug Onyx wird das Lenkungsverhalten von Pure Pursuit ebenfalls verbessert. Jedoch wird die definierte maximale Sollspurabweichung nicht entlang der gesamten Teststrecke gehalten. Das ist auf die Tatsache zurückzuführen, dass aufgrund des Datenrauschens die von FQI trainierten neuronalen Netze unpräzise die Q-Werte der Zustand-Aktions-Paare approximieren.

8 Ausblick

In diesem Kapitel werden weitere Entwicklungsschritte zur Optimierung des prädiktiven NFQ-Spurführungsverfahrens vorgestellt. Zunächst werden Verbesserungen des Verfahrens in Abschnit 8.1 beschrieben. Optimierungsmöglichkeiten für das FAUST-Fahrzeug Onyx werden in Abschnit 8.2 genannt.

8.1 Verbesserung des Verfahrens

Zur Beschreibung der Sollspurgeometrie setzt das in dieser Arbeit entwickelte Verfahren ein Polynom zweiten Grades, weil die Spurerkennung Polaris die Fahrspurmarkierungen durch Polynome zweiten Grades approximiert (vgl. Abschnitt 4.2.2). Dieser Ansatz liefert keine genaue Beschreibung der Fahrspurgeometrie, insbesondere für S-Kurven. Zu diesem Zweck können Bézierkurven anstatt Polynome eingesetzt werden.

In dieser Arbeit wurde die Approximationsleistung von neuronalen Netzen und Support-Vektor-Regression ausgewertet (vgl. Abschnitt 6.1). Die prädiktive NFQ-Spurführung verwendet neuronale Netze zur Approximation der Q-Funktion, wobei der Rprop-Algorithmus zum Trainieren eingesetzt wurde. Zur Auswerteung der SVR wurde eine Radiale-Basis-Funktion als Kernelfunktion verwendet. Um die Präzision der Approximation zu verbessern, können weitere Trainingsalgorithmen und Kernelfunktionen untersucht werden.

8.2 Optimierung der Spurführung für das Fahrzeug Onyx

Zustand-Aktions-Paare werden während der Trainingsphase bezüglich der Sollspurabweichung bewertet, um einen Trainingsdatensatz für NFQ zu bilden. Bei dem Fahrzeug Onyx wird die Sollspurabweichung mit Hilfe der kamerabasierten Spurerkennung Polaris ermittelt (vgl. Abschnitt 5.4). Jedoch ermöglicht die Onyx-Kameraposition eine Identifizierung der Fahrspurgeometrie erst ab 10 cm vor dem Fahrzeug. Das führt zu einer ungenauen Schätzung der Fahrzeugposition bezüglich der Fahrspur. Zur Verbesserung dieser Schätzung ist eine Änderung der Kameraposition oder ein weiterer Sensor zur Erfassung der Fahrzeugposition erfor-

derlich. Außerdem führen Rutschen oder Durchdrehen der Räder zu einer falschen Ermittlung des aktuellen Lenkwinkels (vgl. Abschnitt 5.5). Zu diesem Zweck sind ebenfalls weitere Sensoren erforderlich.

Wie bereits in Abschnitt 6.1 beschrieben, fehlt auf der Softwareseite eine Synchronisierung zwischen der Spurerkennungstask Polaris und dem Kameratreiber, was zum Datenrauschen im Trainingsdatensatz führt. Außerdem soll ein Tempomat entwickelt werden, sodass die gewünschten Geschwindigkeiten während der Trainigsphase gehalten werden können (vgl. Abschnitt 4.2.5).

Literaturverzeichnis

[Alpaydin 2004] ALPAYDIN, Ethem: *Introduction to Machine Learning*. The MIT Press Cambridge, Massachusetts London, England, 2004

[Bradski 2000] BRADSKI, G.: The OpenCV Library. In: *Dr. Dobb's Journal of Software Tools* (2000). – Software available at `http://opencv.willowgarage.com/wiki/Welcome`

[Brooks 1986] BROOKS, R. A.: A Robust Layered Control System for a Mobile Robot. In: *IEEE Journal of Robotics and Automation 2*. 1986

[Carolo-Cup] CAROLO-CUP: *Homepage des Carolo-Cup Wettbewerbs*. – URL `http://www.carolocup.de`

[Carolo-Cup-Regelwerk 2011] CAROLO-CUP-REGELWERK: *Carolo-Cup-Regelwerk*. 2011. – URL `http://www.carolo-cup.de/uploads/media/Regelwerk.pdf`

[Chang und Lin 2011] CHANG, Chih-Chung ; LIN, Chih-Jen: LIBSVM: A library for support vector machines. In: *ACM Transactions on Intelligent Systems and Technology* 2 (2011), S. 27:1–27:27. – Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

[Coulter 1992] COULTER, R. C.: Implementation of the Pure Pursuit Tracking Algorithm / Robotics Institute, Carnegie Mellon University. 1992. – Forschungsbericht

[DARPA] DARPA: *DARPA Urban Challenge*. – URL `http://archive.darpa.mil/grandchallenge/index.asp`

[Drucker u. a. 1996] DRUCKER, Harris ; BURGES, Chris J. ; KAUFMAN, Linda ; SMOLA, Alex ; VAPNIK, Vladimir: *Support Vector Regression Machines*. 1996

[Ernst u. a. 2005] ERNST, Damien ; GEURTS, Pierre ; WEHENKEL, Louis ; LITTMAN, L.: Tree-based batch mode reinforcement learning. In: *Journal of Machine Learning Research* 6 (2005), S. 503–556

[Espié u. a. 2006] ESPIÉ, Eric ; GUIONNEAU, Christophe ; WYMANN, Bernhard ; DIMITRAKAKIS, Christos ; COULOM, Rémi ; SUMNER, Andrew: *TORCS, The Open Racing Car Simulator*. http://www.torcs.org. 2006. – URL `http://www.torcs.org`

[FAUST 2011] FAUST: *FAUST Homepage*. 2011. – URL `http://www.informatik.haw-hamburg.de/faust.html`

[Hensel 2008] HENSEL, Enrico: *Führungskonzept eines autonomen Fahrzeuges*, Hochschule für Angewandte Wissenschaften Hamburg, Bericht Anwendung 1, 2008

[Hoffmann u. a. 2007] HOFFMANN, G. M. ; TOMLIN, C. J. ; MONTEMERLO, M. ; THRUN, S.: Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In: *In To appear in the Proceedings of the 26th American Control Conference*, 2007

[Jenning 2009] JENNING, Eike: *Systemidentifikation eines autonomen Fahrzeugs mit einer robusten, kamerabasierten Fahrspurerkennung in Echtzeit*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2009

[Kelly 1994] KELLY, Alonzo: An Intelligent Predictive Controller for Autonomous Vehicles / CMU Robotics Institute. 1994. – Forschungsbericht

[Luca u. a. 1997] LUCA, Alessandro D. ; ORIOLO, Giuseppe ; DE, Alessandro ; SAMSON, Claude: *Feedback Control Of A Nonholonomic Car-Like Robot*. 1997

[Lundgren 2003] LUNDGREN, Martin: *Path Tracking and Obstacle Avoidance for a Miniature Robot*, Umea University, Masterarbeit, 2003

[Montemerlo u. a. 2007] MONTEMERLO, Mike ; DAHLKAMP, Hendrik ; RIEDMILLER, Martin: Learning to Drive a Real Car in 20 Minutes. In: *FBIT conference, Jeju, Korea. Special Track on autonomous robots.*, URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.70.3532&rep=rep1&type=pdf`, 2007

[Murphy 1994] MURPHY, Karl N.: Analysis of robotic vehicle steering and controller delay. In: *Yuh, Volume 5, Robotics and Manufacturing, pg 631, Wailea*, 1994, S. 631–636

[Nguyen und Widrow 1990] NGUYEN, D ; WIDROW, B: Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In: *International Joint Conference on Neural Networks* 3 (1990), S. 21–26. – URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=137819`

[Nikolov 2009] NIKOLOV, Ivo: *Verfahren zur Fahrbahnverfolgung eines autonomen Fahrzeugs mittels Pure Pursuit und Follow-the-carrot*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2009

[Ramana 2006] RAMANA, B. V.: *Higher Engineering Mathematics*. Tata McGraw-Hill Education, 2006

[Rankin 1997] RANKIN, Arturo L.: *Development of path tracking software for an autonomous steered-wheeled robotic vehicle and its trailer*. Gainesville, FL, USA, Dissertation, 1997. – UMI Order No. GAX98-01142

[Riedmiller 1997] RIEDMILLER, Martin: Generating continuous control signals for reinforcement controllers using dynamic output elements. In: *European Symposium on Artificial Neural Networks, Bruges*, URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.54.9217&rep=rep1&type=pdf, 1997

[Riedmiller 2005] RIEDMILLER, Martin: Neural Fitted Q Iteration - First experiences with a data efficient neural Reinforcement Learning Method. In: *European Conference on Machine Learning, Porto, Portugal*, URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.72.1193&rep=rep1&type=pdf, 2005

[Riedmiller und Braun 1993] RIEDMILLER, Martin ; BRAUN, Heinrich: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In: *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, 1993, S. 586–591

[Rumelhart u. a. 1988] RUMELHART, David E. ; HINTON, Geoffrey E. ; WILLIAMS, Ronald J.: *Learning representations by back-propagating errors*. S. 696–699. Cambridge, MA, USA : MIT Press, 1988. – URL http://dl.acm.org/citation.cfm?id=65669.104451. – ISBN 0-262-01097-6

[Russell und Norvig 2009] RUSSELL, S. J. ; NORVIG, P.: *Artificial Intelligence: A Modern Approach*. 3rd. Prentice Hall, 2009

[Snider 2009] SNIDER, Jarrod M.: Automatic Steering Methods for Autonomous Automobile Path Tracking / Robotics Institute. Pittsburgh, PA, March 2009 (CMU-RI-TR-09-08). – Forschungsbericht

[Sutton und Barto 1998] SUTTON, R. S. ; BARTO, A. G.: *Reinforcement Learning*. MIT Press, Cambridge, MA, 1998

[Volkswagen 2006] VOLKSWAGEN: *Automatischer GTI*. 2006. – URL http://www.volkswagenag.com/vwag/vwcorp/content/de/innovation/research_vehicles/automatic_gti.html

[Watkins 1989] WATKINS, C.: *Learning from Delayed Rewards*, University of Cambridge,England, Dissertation, 1989

[Weiser u. a. 2009] WEISER, Andreas ; BARTELS, Dr. A. ; STEINMEYER, Simon ; SCHULTZE, Dipl.-Ing K. ; MUSIAL, Dr. M. ; WEISS, Dr. K.: Intelligent Car - Teilautomatisches Fahren auf der Autobahn. In: *AAET*, 2009

[Wille u. a. 2010] WILLE, J. M. ; SAUST, F. ; MAURER, M.: Stadtpilot: Driving autonomously on Braunschweig's inner ring road. In: *IEEE Intelligent Vehicles Symposium (IV)*, URL `http://dx.doi.org/10.1109/IVS.2010.5548034`, Juni 2010

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

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. November 2011	
Ort, Datum	Unterschrift