Karlsruher Institut für Technologie

Kognitive Automobile Praktikum

Fahrtrajektorien

Studienarbeit
von

Juan Camilo Vasquez Tieck, Joachim Gehrung, Christopher Schulte, Matija Ilijas, Martin Asghar Schulze, Giorgio xxx

An der Fakultät für Informatik
Institut für Anthropomatik (IFA)
Humanoids and Intelligence Systems Laboratories (HIS)

Referees:	**Prof. Dr.-Ing. Rüdiger Dillmann**
Betreuender Mitarbeiter:	**Dipl.-Inform. ...**
Zweiter betreuender Mitarbeiter:	**Dipl.-Inform. ...**

Bearbeitungszeit: October 2012 – April 2013

Inhaltsverzeichnis

1 Einleitung

Randbedingungen, Dauer, Teilnehmer...

This project is being developed under the scope of the “Kognitive Automobile Praktikum” at FZI, KIT (Karlsruhe Institute of Technology, Karlsruhe, Germany).

1.1 Aufgabenstellung

blablabla...

blablabla... (**?**)

2 Überblick Framework

2.1 Arbeitsumgebung

blablabla

blablabla

2.2 Setup

blablabla

blablabla

2.3 CarMaker/Apo-Client Anpassungen

CarMaker ist eine von IPG Automotive (http://www.ipg.de) entwickelte Software, welche eine umfassende Simulation der Fahrzeugdynamik von Automobilen bietet. In unserem Versuchsaufbau ist diese Software für die Kommunikation mit dem Fahrsimulator und der Realisierung der Streckenführung zuständig. Der ApoClient bietet uns eine ROS Schnittstelle zum CarMaker, sodass wir auf alle vom CarMaker empfangenen Daten auch über ROS zugreifen können.

Für unsere Problemstellung ist allerdings nicht nur ein Empfangen der Daten notwendig, sondern auch ein Übermitteln, da die drei Zielgrößen in Echtzeit übermittelt werden sollen. CarMaker und ApoClient wurden also so angepasst, dass sie in jedem Zeitschritt auch die Werte der drei Zielgrößen empfangen und an das Automobil weitergeben.

Die anfängliche Abtastrate des Systems betrug beim Projektstart nur 4-5 Hz. Bei dieser niedrigen ist eine vollständige Rekonstruktion der Lenkbewegung nur schwer möglich, weil der Fahrsimulator die übermittelten Werte direkt exakt einstellt und somit keine natürliche und flüssige Lenkbewegung ausführt. Durch eine höhere Abtastrate wird diese Lenkbewegung wieder vollständiger und somit für die weitere Verarbeitung nützlicher. Um eine höhere Abtastrate zu erhalten wurden zwei Änderungen an dem System vorgenommen. Zum Ersten wurden alle relevanten Daten in einem ROS-Topic gebündelt, sodass nicht mehr 5 Topics veröffentlicht werden müssen. Dies bringt allerdings nur eine kleine zeitliche Verbesserung und wurde hauptsächlich wegen der Übersichtlichkeit vollzogen. Die zweite Änderung war eine Reduktion der abgefragten Daten auf die für uns relevanten Daten und somit eine erhöhte Abtastrate.

Insgesamt wurde die Abtastrate von 4-5 Hz auf konstante 15 Hz erhöht und eine bidirektionale Kommunikation in Echtzeit mit dem System ermöglicht.

2.4 Framework

blablabla

blablabla

2.5 Toolbox

blablabla

blablabla

2.6 Visualisierung

blablabla

2.6.1 Visualisierung1

blablabla

2.6.2 Visualisierung2

Eigentlich kann man hier nichts schreiben außer das Bild machen und auf den Datenaufnahme abschnitt verweisen. Dort wird ja das Verfahren genauer erklärt.. ist sonst redundant.

3 Datenaufnahme

3.1 Probleme, Lösungen, Vergleiche...

Die Datenaufnahme bezeichnet die Gewinnung von Lerndaten für die ausgewählten Lernalgorithmen. Die Hauptschwierigkeit hierbei war die Darstellung der zu fahrenden Streckenführung und des jeweiligen Standortes des Referenzfahrzeugs. Eine gute räumliche und zeitliche Verfolgung der Referenzfahrt muss zu jedem Zeitpunkt gegeben sein. Zur Verfügung stand uns das im CarMaker integrierte Tool zur Streckengestaltung und ein kleiner Monitor innerhalb des Fahrzeugs, auf dem wir normale Desktopanwendungen ausführen konnten. Folgende Gedankengänge haben wir implementiert und ausprobiert.

3.1.1 Streckenführung und Ghostcar per CarMaker

Bei unserer ersten Implementierung wurde eine Strecke mittels des CarMakers gebaut und abgefahren. Diese Fahrt wurde als Referenzfahrt gekennzeichnet und durch ein Ghostcar dargestellt. Dieses Vorgehen funktionierte relativ gut, allerdings konnte man durch das fehlende Geschwindigkeitsgefühl und ohne Vorwissen über die Referenzfahrt das Ghostcar nicht gut verfolgen. Abrupte Fahrmanöver wie starkes Bremsen oder Beschleunigen wurden so meist zu spät erkannt, wodurch man sich zu weit von der Referenzfahrt entfernte und die Fahrt wiederholen musste. Hier wäre also zuerst ein menschliches Training nötig gewesen, um dann die Referenzfahrt möglichst gut absolvieren zu können.

3.1.2 Streckenführung und Marker per CarMaker

Hier wurde zusätzlich zur Streckenführung auf Marker zugegriffen. Die CarMaker-Software stellt unterschiedliche Arten von Markern zur Verfügung, wobei hier nur räumliche Marker wie z.B. Straßenmarkierungen, Höchstgeschwindigkeitsschilder oder Bäume benutzt werden können. Allerdings konnte man durch die Nutzung von Schildern eine Vorrauschau für den Fahrer bieten und er konnte sich frühzeitig auf abrupte Veränderungen der Geschwindigkeit einstellen. Da jedoch die Darstellung der Zeit, also wo man sich eigentlich zu dem aktuellen Zeitpunkt befinden müsste, hier nicht von betroffen war, bestand das Problem weiterhin, dass man nicht exakt nachvollziehen konnte, ob man aktuell zu schnell oder zu langsam fährt.

3.1.3 Streckenführung und Marker per CarMaker, Visualisierung per RVIZ

Diese Lösung sieht eine Visualisierung anhand der Streckenführung und Straßenschildern vor, wird allerdings durch einen zweidimensionalen Plot der ROS-eigenen Visualisierung RVIZ ergänzt. Hierzu wurden die zukünftige Positionen der nächsten drei Sekunden der Referenzfahrt eingezeichnet und der Fahrer konnte abrupte Fahrmanöver mit Vorausschau erkennen. Der Plot wurde dann auf den kleinen Bildschirm im Fahrsimulator gelegt, sodass der Fahrer die benötigten Daten aus beiden Quellen ablesen konnte: die grobe Streckenführung und Geschwindigkeit aus der Visualisierung des CarMakers und die exakte Streckenführung und die Zeitschritte bezüglich der Referenztrajektorie aus der RVIZ Visualisierung. Diese Implementierung wurde letztendlich für die Aufnahme unserer Trainingsdaten genutzt.

4 Features

4.1 Strukturierung

blablabla

blablabla

4.2 Berechnung der benutzten Features

blablabla

blablabla

4.3 Vergleiche, Korrelation

blablabla

blablabla

5 Verfahren

Wollt ihr hier alle Verfahren haben, oder ein Datei für jeder?

blablabla

5.1 K-Nearest Neighbor

blablabla

blablabla

5.2 Random Forests

blablabla

blablabla

5.3 Neuronale Netze Martin ODER KOMMT NIX?!

blablabla

blablabla

5.4 Neuronale Netze

Künstliche Neuronale Netze nehmen das menschliche Gehirn als Vorbild und versuchen so eine Abstraktion von Informationsverarbeitung zu schaffen. Ein neuronales Netz besteht aus mehreren Schichten von Neuronen, die beliebig miteinander verbunden sein können. Zusätzlich zur Netztopologie wird noch eine Aktivierungsfunktion festgelegt, welche das „Feuern“ eines Neurons beschreibt. Der Ablauf bei dem Training des Neuronalen Netzes ist folgendermaßen: zuerst wird eine Netztopologie, eine Anzahl von Neuronen und die Aktivierungsfunktion festgelegt. Die Eingabewerte werden an die Neuronen der ersten Schicht (Eingabeneuronen) angelegt und mittels gewichteter Verbindungen und der Aktivierungsfunktion durch das Netz transportiert. Am Ende liefert die letzte Neuronenschicht (Ausgabeneuronen) einen Wert, welcher mit dem Soll-Wert verglichen wird. Das endgültige Lernen des Neuronalen Netzes ist nun die Anpassung der gewichteten Verbindungen zwischen den Neuronen des Netzes. Allerdings spielt auch die Netztopologie und die Anzahl der Neuronen eine wichtige Rolle. Eine genauere Beschreibung kann z.B. in (James A. Anderson: A simple neural network generating an interactive memory. In: Mathematical Biosciences. 14, 1972, ISSN 0025-5564, S. 197–220.) nachgeschlagen werden. Als Implementierung wurde das Framework von Boby Anguelov (http://takinginitiative.net/2008/04/03/basic-neural-network-tutorial-theory/) auf die Aufgabenstellung und die verwendeten Daten angepasst. Dieses Framework bietet ein dreischichtiges FeedForward-Netz und erlaubt eine variable Anzahl an Neuronen pro Schicht. Um die optimale Anzahl an Neuronen zu finden, werden optional sowohl konstruktive als auch destruktive Verfahren angeboten. Wie in Kapitel 4 erläutert, wurde ein bestimmtes Set an Eingabevektoren ausgewählt und auch die Ausgabeneuronen waren durch die Anzahl der Stellgrößen vorgegeben. Weiterhin kann die Aktivierungsfunktion, sowie die Lernparameter wie Lernrate und Momentumterm angepasst werden.

Der erste Teil des Trainings befasste sich also damit, die optimale Anzahl an Neuronen für unsere Aufgabenstellung und die Eingangsdaten zu ermitteln. Als desktruktives Verfahren wurde der Optimal Brain Damage Algorithmus (http://yann.lecun.com/exdb/publis/pdf/lecun-90b.pdf) verwendet, als konstruktives Verfahren die Cascade-Correlation (http://www.cs.iastate.edu/ honavar/fahlman.pdf).

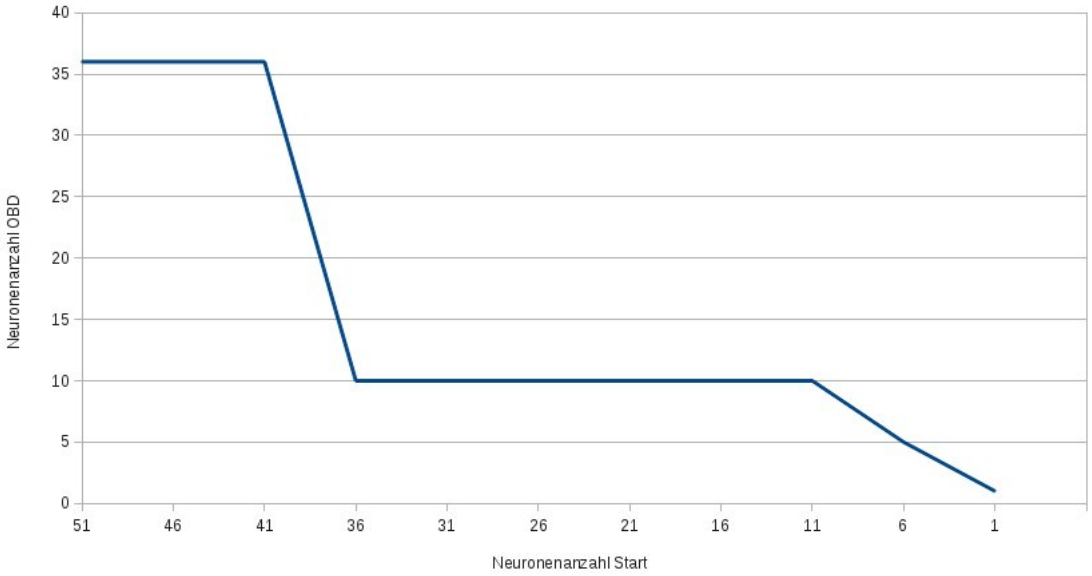

Abbildung 5.1: Ermittelte Neuronenzahl des OBD Algorithmus in Abhängigkeit der Startzahl an Neuronen

Abbildung 5.1 zeigt die von OBD ermittelte Anzahl an Neuronen abhängig von der Ausgangsneuronenanzahl. Es wird deutlich, dass ein Wert von 10 Neuronen vom Algorithmus als optimal aufgefasst wird; niedrige Werte verschlechtern das Ergebnis deutlich und auch höhere Werte sind bestenfalls gleichwertig. Allerdings benötigt das größere Netz deutlich mehr Zeit und auch eine größere Menge an Trainingsdaten.

Abbildung 5.2 zeigt jeweils die von OBD und dem Cascade Training ermittelte Neuronenzahl in Abhängigkeit vom Startwert für das Intervall zwischen 1 und 10 Neuronen. Das konstruktive Cascade Training lässt die optimale Neuronenzahl sehr deutlich gegen 10 konvergieren, welches vom OBD Algorithmus schon von oben eingegrenzt wurde. Für die weiteren Tests wurde also ein Netz mit 18 Neuronen verwendet, wobei 5 Neuronen als Eingabeneuronen, 10 Neuronen als Hidden-Neuronen, sowie 3 Neuronen als Ausgabeneuronen fungierten.

Mit dem festen Netzaufbau wurden nun die Lernrate und der Momentumterm untersucht. Das Training des Neuronalen Netzes wird mittels Gradientenabstieg vorgenommen. Bei diesem Verfahren wird versucht ein globales Minimum in der von den Fehlertermen aufgespannten Hyperebene zu finden. Folgende Probleme können hierbei auftreten: Lokale Minima: Anstatt des globalen Minimums wird nur ein lokales Minimum gefunden und nicht mehr verlassen. Obwohl eventuell eine bessere Gewichtsverteilung existiert, wird diese nicht mehr gefunden. Flaches Plateau: Wenn sich der Gradient über eine große Anzahl von Gewichtsanpassungen nicht mehr verändert, dann stagniert das Verfahren. Der Gradient wird so über weitere Schritte so klein, dass keine weitere Erforschung des Raumes mehr erfolgt. Verlassen guter Minima: Falls ein ausgeprägtes Minimum mit relativ geringer Ausdehnung vorliegt, so kann es übersprungen werden. Also könnte auch so ein globales Minimum übersprungen werden. Oszillation: Das Verfahren springt von einem Minimum in das nächste, wobei es zwischen beiden Minima hin und her springt. In diesem Fall verändern sich nicht die Beträge der Gradienten, sondern lediglich ihr Vorzeichen. Eine hohe Lernrate ermöglicht ein schnelles Erforschen der kompletten Hyper-

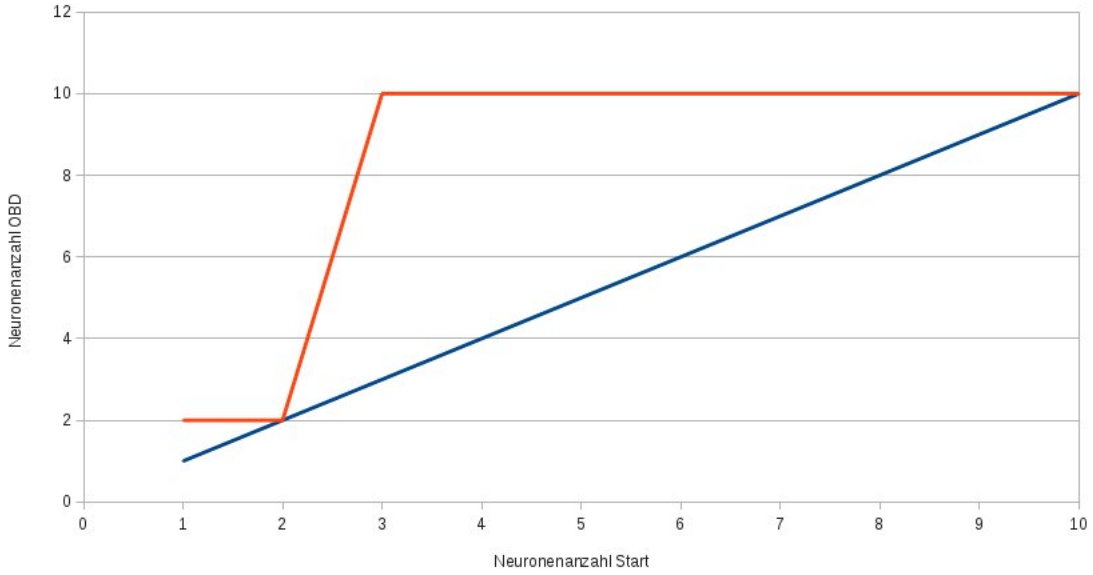

Abbildung 5.2: Ermittelte Neuronenzahl des OBD Algorithmus und des Cascade-Trainings in Abhängigkeit der Startzahl an Neuronen

ebene des Fehlerterms. Flache Plateaus werden schnell durchlaufen und vom Startpunkt weit entfernte Minima werden schnell erreicht. Allerdings steigt auch die Gefahr gute (bzw. globale) Minima zu überspringen oder zwischen gefundenen Minima zu oszillieren. Bei einer Lernrate von 0.9 trat genau dies bei den Trainingsdaten auf. Das Training benötigte zwar nur eine kurze Zeit (eine Minute), allerdings zeigte sich dann eine komplette Oszillation. Eine niedrige Lernrate kann komplexe Lerndaten, sowie eine große Datendichte besser bewältigen, ist weniger anfällig für Oszillation und für das Überspringen von Minima. Allerdings erhöht sich dadurch die Trainingszeit und Plateaus, sowie lokale Minima können nicht mehr überwunden werden. Dies zeigte eine Lernrate von 0.01 deutlich: das Training benötigte mehr als 12 Stunden und es wurde das „erstbeste“ Minimum nicht mehr verlassen. Um die Nachteile einer niedrigen Lernrate auszugleichen, kann man ein Momentum einführen. Dieser Trägheitsterm berücksichtigt zusätzlich zur aktuellen Änderung des Gradienten auch die vorausgegangene Änderung. Hiermit wird ein schnelles überqueren von Plateaus ermöglicht. Als endgültige Parameter wurden eine Lernrate von 0.1 und ein Momentum von 0.9 gewählt.

Die letzte Stellschraube des Lernverfahrens war die erlaubte Varianz im Vergleich der Soll- und Istwerte der Ausgabeneuronen. Da hier keine klassische Klassifikation, also ein binärer Vergleich, stattfand, sondern eine kontinuierliche Funktion gelernt werden sollte, muss dem Lernalgorithmen ein gewisser Spielraum gegeben werden. Weiterhin ist dieser Schritt damit motiviert, dass der Fahrsimulator nur Werte mit bestimmter Genauigkeit annahm, z.B. die Gaspedalstellung in Tausendstel-schritten. Eine hohe Varianz erleichtert natürlich das grobe Einlernen dieser Funktion, wobei eine niedrige Varianz die Funktion besser beschreiben sollte.

Abbildung 5.3 zeigt die Erkennungsrate von Gaspedalstellung und Lenkwinkel in Abhängigkeit des erlaubten Lernfehlers. Eine allgemeine Aussage war hier nicht möglich, da in der Praxis die Gaspedalstellung mit wenig Fehlervarianz und somit niedriger Erkennungsrate, jedoch beim Lenkwinkel eine relativ hohe Fehlervarianz mit hoher Erkennungsrate besser funktionierte.

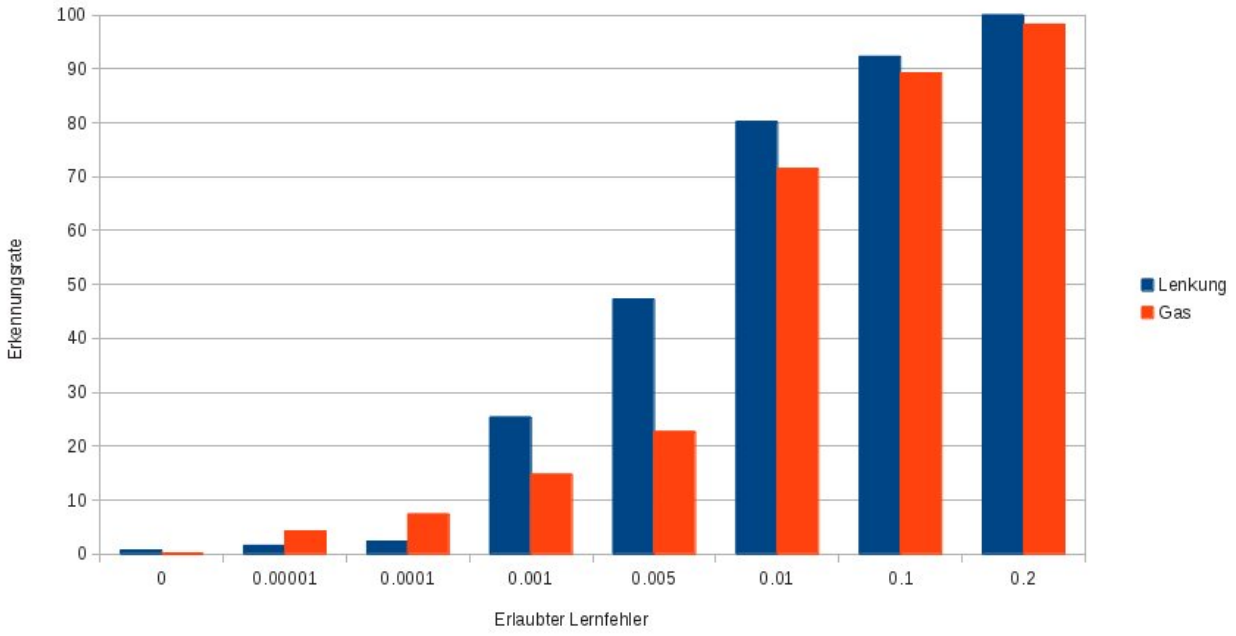

Abbildung 5.3: Erkennungsrate der Stellgrößen Gaspedalstellung und Lenkwinkel in Abhängigkeit der zugelassenen Fehlervarianz

5.5 Support Vector Regression

blablabla

blablabla

6 Ergebnise

6.1 Online

blablabla

blablabla

6.2 Offline

blablabla

blablabla

7 Fazit / Ausblick

- blablabla.
- blablabla.

Literaturverzeichnis

[Asfour u. a. 2006] ASFOUR, T. ; REGENSTEIN, K. ; AZAD, P. ; SCHRÖDER, J. ; VAHRENKAMP, N. ; DILLMANN, R.: ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. In: *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*. Genova, Italy, December 2006, S. 169–175

8 Annexes

8.1 Additional Resources

- A very interesting book, that includes many works on the field. Magnenat-Thalmann, N. & Thalmann, D. eds., 1998. Modelling and Motion Capture Techniques for Virtual Environments, Geneva, Switzerland: Springer. Available at: http://www.springer.com/computer/ai/book/978-3-540-65353-0.
- Previous work for one of the presented papers. Ross, D.A. et al., 2008. Unsupervised learning of skeletons from motion D. Forsyth, P. Torr, & A. Zisserman, eds. , 5304, pp.560-573. Available at: http://dl.acm.org/citation.cfm?id=1478172.1478217 [Accessed January 9, 2012].