FREIE UNIVERSITÄT BERLIN

MASTERARBEIT AM INSTITUT FÜR INFORMATIK DER FREIEN UNIVERSITÄT BERLIN, ARBEITSGRUPPE INTELLIGENTE SYSTEME UND ROBOTIK

Effektives Einordnen autonomer Fahrzeuge in den Straßenverkehr - ein Ansatz zu Schwarmverhalten basierend auf neuronalen Netzen

Autor:
Marcus JAHNS
Matrikelnummer:
4370222

Betreuer:
Prof. Dr. Daniel GÖHRING

Berlin, 12. September 2017

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Unterschrift:

Datum:

Freie Universität Berlin

Zusammenfassung

Institut für Informatik
Fakultät für Mathematik und Informatik

Master of Science

Effektives Einordnen autonomer Fahrzeuge in den Straßenverkehr - ein Ansatz zu Schwarmverhalten basierend auf neuronalen Netzen

von Marcus JAHNS

In den vergangenen Jahren gab es immer weitere Entwicklungen auf dem Gebiet der Robotik und autonomen Systeme. Ein Bereich davon ist die Forschung an autonomen Fahrzeugen und daran sie in den normalen Straßenverkehr integrieren zu können. Das wohl bekannteste Beispiel für ein solches Auto ist das von Google seit 2009 entwickelte Waymo. Einer der wichtigsten Aspekte bei dieser Entwicklung ist, neben der Sicherheit für Fahrzeuginsassen und andere Fahrer, zu gewährleisten, dass es sich möglichst optimal in den Verkehr eingliedern kann und nicht zur Behinderung für normale Autofahrer wird.

Im Rahmen dieser Arbeit soll es darum gehen, mit Hilfe von maschinellem Lernen und gesammelten Daten von autonomen Fahrzeugen der Autonomous Cars Projektgruppe, zu ermitteln wie sich ein solches Fahrzeug abhängig von anderen Autos am besten auf der Straße einordnen sollte. Dabei soll es nicht darum gehen optimal den vorhandenen Platz auf der Straße auszunutzen, sondern stattdessen möglichst gut das Fahrverhalten eines durchschnittlichen Fahrers zu imitieren um sich möglichst natürlich in den Verkehr einzugliedern.

Dabei sollen Sensordaten von Testfahrten betrachtet werden und die Positionsdaten anderer Fahrzeuge mit Hilfe von neuronalen Netzen angelernt werden um für eine gegebene Verkehrssituation eine möglichst gute Positionierung angeben zu können.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

AutoNOMOS Labs

Bei AutoNOMOS Labs handelt es sich um das Projekt für autonome Fahrzeuge an der Freien Universität Berlin, in dessen Rahmen auch diese Arbeit verfasst wurde. Begonnen hat dieses im Jahr 2007 mit der Arbeit am Spirit of Berlin Auto und befasste sich seitdem mit verschiedenen Forschungsthemen auf dem Gebiet des autonomen Fahrens. Im Laufe der Jahre wurden noch zwei weitere autonome Fahrzeuge entwickelt, an denen zur Zeit Forschung betrieben wird. Zum einem das Elektroauto e-Instein, bei dem es sich um einen modifizierten Mitsubishi i-MiEV handelt und zum anderen ein Volkswagen Passat Variant 3c mit dem Namen MadeInGermany (MIG), auf dessen Fahrten auch die in dieser Arbeit verwendeten Daten basieren. Seit Frühjahr 2011 dürfen diese auch nach langen Tests auf dem Flughafengelände Tempelhof - im Rahmen von Testfahrten unter Überwachung und mit Insassen - im normalen Straßenverkehr durch Berlin fahren[1].

ABBILDUNG 1.1: AutoNOMOS Fahrzeug e-Instein (von [3]).

Für die Fahrten durch Berlin gibt es dabei ein strenges Sicherheitskonzept. So muss sich stets ein Sicherheitsfahrer eingriffsbereit hinter dem Steuer befinden, der gemeinsam mit einem Co-Piloten alle Aktivitäten des Fahrzeugs überwacht. Aktionen werden dabei vom Computer durch eine synthetische Stimme vorab angekündigt und durch ein Antippen des Bremspedals oder das Betätigen eines der Notschalter am Armaturenbrett oder im Fußbereich lässt sich jederzeit sekundenschnell die Kontrolle über das Fahrzeug übernehmen. Ebenfalls lässt sich durch eine Fernbedienung das Auto jederzeit durch eine dritte Sicherheitsperson von außen stoppen. Zusätzlich zur menschlichen Überwachung befindet sich im Kofferraum eine elektronische Schaltung, die die Dynamik des Fahrzeugs überwacht und alle unsicheren

Befehle wie große Lenkbewegungen bei hoher Geschwindigkeit grundsätzlich blockiert[4].

ABBILDUNG 1.2: AutoNOMOS Fahrzeug MadeInGermany (von [2]).

1.1 MadeInGermany

Das Auto MadeInGermany wurde unter anderem mit dem Ziel entwickelt, die Fusion von Sensordaten verschiedener Hersteller zu erforschen und damit neue Impulse für die Industrie zu setzen. So verfügt es zum einem über insgesamt sechs Ibeo Lux-Laserscanner. Jeweils drei davon befinden sich jeweils an der Vorder- und Rückseite des Autos um zusammen eine 360 Grad Erfassung von Hindernissen zu ermöglichen. Jeder dieser Laserscanner sendet regelmäßig Lichtimpulse und ist in der Lage auf bis zu 100 Metern Reichweite die Reflexion und Zeitspanne bis zum Echo zu messen. Zusätzlich befindet sich auf dem Dach des Autos ein Velodyne Laserscanner. Dieser rotiert und versendet dabei 64 Strahlen im Infrarotbereich welche eine Erfassung von bis zu 70 Metern in alle Richtungen ermöglicht[2]. Zusätzlich ist MadeInGermany mit Kameras, Radar und GPS zur Positions- und Umgebungserfassung ausgestattet.

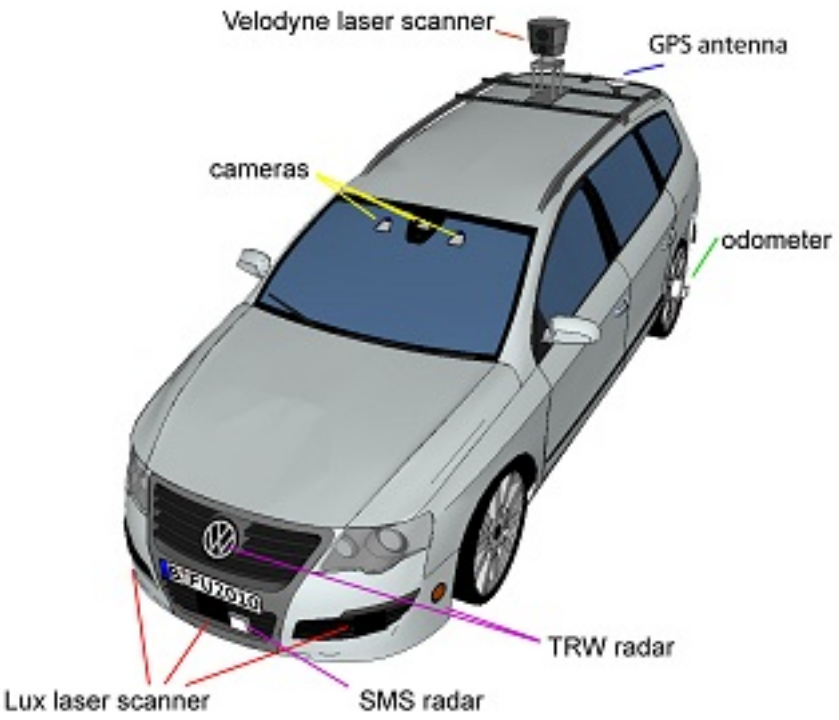

ABBILDUNG 1.3: AutoNOMOS Fahrzeug MadeInGermany (von [2]).

Diese Arbeit verwendet dabei die Daten der Ibeo Lux-Laserscanner von einer in etwa 20 minütigen Testfahrt vom Tempelhofer Gelände zur Freien Universität Berlin, da sich in diesen Aufzeichnungen viele Situationen mit anderen Autos im Straßenverkehr befinden, welche als Trainingsmenge für das maschinelle Lernen geeignet sind.

Kapitel 2

ROS

2.1 ROS

Bei ROS handelt es sich um ein Framework zur Erstellung von Software für Roboter, dessen Entwicklung von Willow Garage im Jahr 2007 begann und das seit dem, auch auf Grund der open-source Lizenz, zunehmend Verbreitung gefunden hat. Mittlerweile verzeichnet es nach eigenen Angaben zehntausende Nutzer. Der Gedanke dabei ist, dass keine zentrale Entwicklung stattfindet, sondern jeder das Framework verwenden, weiterentwickeln und seine Arbeit anderen zur Verfügung stellen kann aber nicht muss[5].

Das Framework bietet Dienste, die mit denen eines Betriebssystems vergleichbar sind wie Hardware-Abstraktion, Implementierung verbreiteter Funktionalitäten, Nachrichtenübertragen zwischen Prozessen und das Verwalten von Datenpaketen. Es wird von der Open Source Robotics Foundation betrieben.

2.2 Die ROS-Architektur

2.2.1 Die ROS-Graphenstruktur

In ROS arbeiten Prozesse in Form von Knoten (ROS nodes) eines Graphen in einem peer-to-peer Netzwerk. Dabei hat jeder Knoten eine spezifische Aufgabe. So kann beispielsweise ein Knoten Daten eines Laserscanners bereitstellen, ein weiterer für Pfadplanung zuständig sein und ein anderer eine graphische Ausgabe anbieten. Der Datenaustausch zwischen Knoten erfolgt dabei in Form von Nachrichten (ROS messages). Jedoch werden diese nicht direkt zwischen Knoten versandt, sondern basieren auf einem publisher und subscriber System. Das bedeutet, das ein Knoten seine Daten verbreiten kann, indem er sie an ein bestimmtes topic publiziert. Andere Knoten haben die Möglichkeit als subscriber Daten von diesem Topic gesandt zu bekommen. Dabei kann ein Topic sowohl mehrere Subscriber, als auch mehrere publisher gleichzeitig besitzen. Im Normalfall wissen diese Knoten dabei nichts von der Existenz der anderen. Dies dient der Entkoppelung der Bereitstellung und Weiterbearbeitung der Daten. Ebenfalls bietet ROS services in Form von einem Paar von Nachrichten an. Dabei kann ein Knoten durch eine Nachricht an einen anderen Knoten, der einen Service anbietet, eine Anfrage senden und erhält das Ergebnis in einer Antwort ebenfalls durch eine Nachricht zurückgesandt. Eine weitere Funktionalität in der ROS-Architektur sind bags. Dies ist ein Format zum Speichern und Wiedergeben von Nachrichten-Daten. Sie ermöglichen die Daten die beispielsweise während einer Testfahrt mit einem autonomen Fahrzeug über Nachrichten verschickt wurden zu speichern und später erneut abzuspielen, wodurch sich die exakt selbe Fahrt beliebig oft simulieren lässt, ohne tatsächlich neu Daten erfassen zu müssen[6].

2.2.2 Das ROS-Dateisystem

In ROS werden die Komponenten in Paketen (ROS Packages) strukturiert. Diese können neben einer XML-Datei mit Informationen über das Paket Knoten, Nachrichten, Dienste, aber auch Bibliotheken oder Software von Drittanbietern enthalten. Sie stellen die atomarsten Bausteine von ROS Systemen dar[6].

Kapitel 3

Theoretische Grundlagen

3.1 Schwarmverhalten

3.1.1 Schwarmverhalten in der Natur

Der Begriff der Schwarmintelligenz hat seinen Ursprung in der Biologie. Seien es Vogelschwärme und ihre Art in einer V-Formation zu fliegen um die größtmögliche Distanz zurücklegen zu können, die Bewegungen von Fischschwärmen oder das komplexe Verhalten von Bienenschwärmen, so wird mit allen von ihnen der Begriff Schwarmverhalten oder auch Schwarmintelligenz verbunden. Dabei lassen sich Ergebnisse beobachten die nicht nur die Möglichkeiten sondern auch das Verständnis eines einzelnen Individuums überschreiten würden. Zwei der beeindruckenderen Beispiele sind dafür zum einen die Ameisengattung Eciton Burchelli bei der Nahrungssuche. Dies kann bis zu 200.000 Arbeiter des Schwarms involvieren, welche innerhalb eines einzigen Tages eine Fläche von 1500 Quadratmetern absuchen können[7]. Dies zu koordinieren wäre ein Aufwand zu dem kein einzelnes Individuum im Stande wäre. Ein zweites Beispiel ist die afrikanische Termitenart Macrotermes Bellicosus, deren Nester Ausmaße von bis zu 30m Durchmesser und 6m Höhe annehmen können[7]. Zusätzlich zu den bereits genannten Vorkommen von Schwarmverhalten, lässt es sich unter anderem auch bei Bakterien, Amöben oder aber bei Menschengruppen beobachten[7].

Früher wurde dabei fälschlicherweise angenommen, dass der Schwarm von einer kontrollierenden Einheit in Form eines einzelnen Individuums organisiert wird - z.B. der Bienenkönigin. Dies lies sich in den letzten Jahrzehnten jedoch widerlegen. Tatsächlich handelt jedes einzelne Individuum des Schwarms auf der Basis seiner eigenen lokalen Informationen ohne den Einfluss einer überwachenden Einheit. Dabei verfügt es über ganz elementare Verhaltensregeln. Diese führen auch zur Informationsverbreitung innerhalb des Schwarms durch Interaktion der Individuen untereinander anhand dieser Regeln. Dies geschieht für gewöhnlich durch Stigmergie, welche eine indirekte Form der Kommunikation durch Veränderung der Umgebung bezeichnet[7].

Beispiele für Stigmergie in Schwärmen lassen sich gut an Wespen oder Ameisen zeigen. So entscheiden Wespen beim Bauen des Nests anhand der bereits existierenden Struktur. Die Wahrscheinlichkeit an einer Stelle mit dem Bau einer neuen Zelle anzufangen hängt dabei davon ab, wie viele andere Zellen dort bereits angrenzen. Bei drei angrenzen Zellen wäre so die Wahrscheinlichkeit sehr hoch, wohingegen die Chance, dass eine Wespe eine neue Reihe anfängt, wo nur ein oder zwei Zellen angrenzen würden, deutlich geringer ist. Das Resultat dieser Verhaltensregeln ist, dass eine Wespe durch den Bau einer Zelle nicht nur die Struktur des Nests beeinflusst,

sondern dadurch die anderen Wespen in ihrer Umgebung beeinflusst - in Form von indirekter Kommunikation.

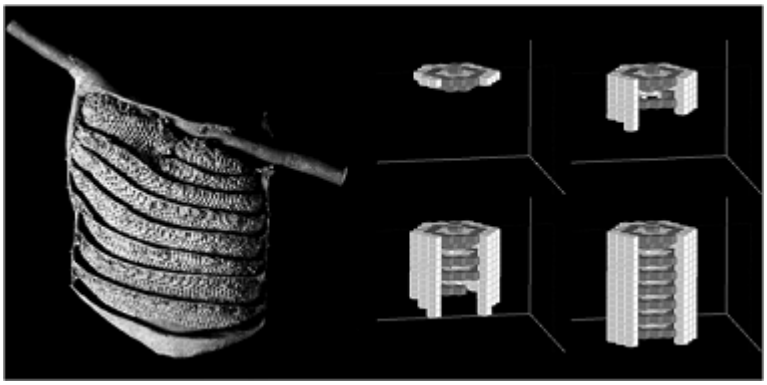

ABBILDUNG 3.1: Struktur eines Wespennests (links) und Simulation kollektiven Bauens einer hexagonalen Struktur (rechts). (von [7])

Bei Ameisen lässt sich die Stigmergie besonders bei der Nahrungssuche beobachten. Dabei ist nur eine einzige Verhaltensregel entscheidend, nämlich das eine Ameise, die Nahrung zurück zum Nest transportiert, Pheromone absondert. Dies hat zur Folge, dass andere Ameisen mit einer höheren Wahrscheinlichkeit diesen Weg auf der Nahrungssuche einschlagen, und falls sie dort ebenfalls auf Nahrung gestoßen sind, selbst Pheromone auf dem Rückweg hinterlassen. So bilden sich Spuren vom Nest zu Nahrungsquellen, allein basierend darauf, dass die einzelnen Ameisen des Schwarms mit ihrer Umgebung interagieren und dadurch auf indirekte Weise dem Schwarm kommunizieren, wo Nahrung zu finden ist.

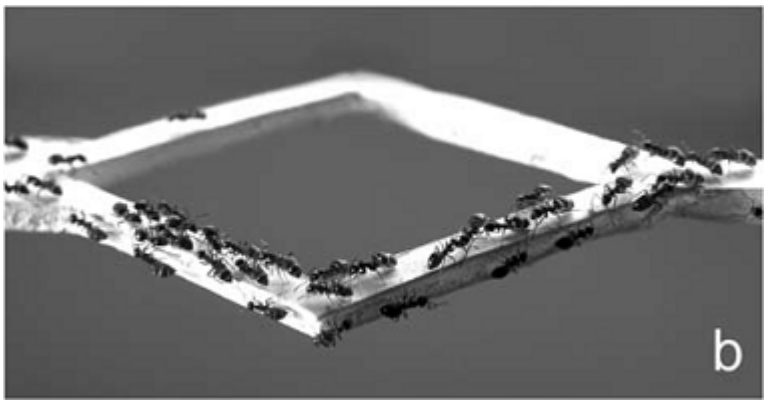

ABBILDUNG 3.2: Ameisen zwischen Nest und Nahrungsquelle. Obwohl beide Wege gleich lang sind, wählen aufgrund der vorhandenen Pheromonspur fast alle Ameisen den selben Weg. (von [7])

Doch das Beispiel an den Ameisen zeigt nicht nur, wie neue Nahrungsquellen

für den Schwarm erschlossen werden. Denn wenn nun eine dieser Quellen aufgebraucht ist und die ankommenden Ameisen keine Nahrung mehr vorfinden, werden sie auch keine Pheromone auf dem Weg zum Nest mehr hinterlassen. Dies hat zur Folge dass die existierende Spur sich mit der Zeit auflöst und die Individuen des Schwarms mit zunehmender Wahrscheinlichkeit wieder auf anderen Wegen nach Nahrung suchen. Auf diese Weise ist das System der Nahrungsversorgung des Schwarms als Ganzes in der Lage, neue Nahrungsquellen zu integrieren sowie versiegte Quellen aus dem Netzwerk von wegen wieder zu entfernen. Und noch eine weitere Besonderheit lässt sich beobachten. So haben Ameisen zwar eine deutlich höhere Chance einer Pheromonspur zu folgen, doch wie auch in Abbildung 3.2 zu sehen ist, können einzelne Individuen trotzdem andere Wege einschlagen. Wenn nun eine Ameise auf diese Weise auf eine andere Nahrungsquelle stößt, die näher am Nest liegt, oder sogar einen kürzeren Weg zu einer bereits erschlossenen Quelle findet, so hinterlässt sie auf diesem Weg Pheromone, die andere Ameisen ebenfalls anlocken. Da dieser Weg jedoch kürzer ist, können die Individuen des Schwarms diesen in kürzerer Zeit zurücklegen, wodurch ein einzelnes seine Pheromone öfter in der selben Zeit hinterlässt, als es auf dem Weg zu einer anderen erschlossenen Nahrungsquelle der Fall wäre. Auf diese Weise prägt sich die neue Pheromonspur schneller aus und wird auf lange Sicht die Mehrheit der Schwarmmitglieder vom längeren Pfad weglocken, insbesondere wenn beide Wege zur gleichen Nahrungsquelle führen. Auf diese Weise sind Ameisen als Schwarm sogar in der Lage für ihre Nahrungsversorgung das Problem von kürzesten Wegen zu lösen, welches als Thema auch in der Informatik Relevanz besitzt. Das bemerkenswerte daran ist, dass die Individuen des Schwarms, in diesem Fall also die einzelne Ameise als solche, sich nie mit diesem Problem auseinander setzt, sondern nur seine elementaren Verhaltensregeln befolgt. Dies wird auch als Selbstorganisation in Schwärmen bezeichnet.

Formal definieren lässt sich Selbstorganisation in Schwärmen als eine Menge von dynamischen Mechanismen, die zu Strukturen im globalen System führen, basierend auf Interaktionen auf individuellem Level ohne dass diese Strukturen auf diesem codiert sind[7]. Im Falle der Ameisen ist die Struktur im globalen System also die Nahrungsversorgung des Schwarms durch das Verwalten der Pfade zu Nahrungsquellen, inklusive dem Hinzufügen neuer, dem Löschen nicht mehr relevanter Pfade sowie dem Finden kürzester Wege. Dabei ist auf dem individuellen Level nur das Absondern von Pheromonen codiert und die erhöhte Wahrscheinlichkeit der Individuen diesen zu folgen. Es gibt vier Voraussetzungen für Selbstorganisation (vgl. [7]). Bei den ersten beiden handelt es sich um positives und negatives Feedback, welches entweder das Entstehen von Strukturen fördert oder im anderen Fall als Gegengewicht dient. Im Beispiel der Nahrungssuche von Ameisen ist das Feedback das Finden oder aber das nicht Finden von Nahrung. Die dritte Voraussetzung ist Fluktuation, welche es ermöglicht neue Lösungen zu entdecken. Bei den Ameisen ist dies die Wahrscheinlichkeit einer Pheromonspur zu folgen und die damit gegebene Gegenwahrscheinlichkeit einen anderen Weg einzuschlagen. Die abschließende vierte Voraussetzung ist das Vorhandensein von direkter oder indirekter Interaktion zwischen den Individuen des Schwarms, da sonst Informationen von Individuen nicht zum Schwarm gelangen könnten. Systeme die auf Selbstorganisation basieren, haben dabei ebenfalls vier charakteristische Eigenschaften (vgl. [7]). Zum einem sind sie aufgrund der zahlreichen Interaktionen der Individuen, sowie des Einflusses von positivem und negativem Feedback, dynamisch. Weiterhin weisen diese Systeme komplexere Eigenschaften als den Beitrag ihrer Individuen auf. Diese

entstehen durch nicht-lineare Kombination von Interaktionen zwischen den Individuen. Außerdem können diese nicht-linearen Interaktionen bei einer Veränderung der Parameter des Systems zu neuen stabilen Lösungen führen. Als letzte Eigenschaft sind diese Systeme multistabil. Das bedeutet, dass für einen gegebenen Ausgangszustand mehrere stabile Zustände erreicht werden können. Dies lässt sich an Abbildung 3.2 gut erkennen. Abhängig davon auf welchem der beiden gleich langen Wege die erste Ameise Nahrung zurückbringt, wird sich in den meisten Fällen auf diesem zuerst eine ausgeprägt Pheromonspur bilden, welche dafür sorgt das nahezu alle Ameisen diesen benutzen werden. Somit gibt es in diesem Fall zwei stabile Zustände für das Erschließen dieser einzelnen Nahrungsquelle.

Bei der dritten Eigenschaft von Systemen basierend auf Selbstorganisation wurde beschrieben, dass Schwärme in der Lage sind, sich an eine Änderung der Parameter des Systems anzupassen. Dabei gilt jedoch zu bemerken, dass eine Änderung des Schwarmverhaltens nur durch eine Änderung im Individualverhalten geschehen kann. Dies funktioniert dadurch, dass die Wahrscheinlichkeit des Auftretens eines bestimmten Verhaltens von einem Individuum variiert in Abhängigkeit von den lokalen Bedingungen. Diese unterschiedlichen Verhaltensmöglichkeiten wiederum haben eine Auswirkung auf die globale Struktur des gesamten Systems. Auch hier liefern Ameisen wieder ein gutes Beispiel. So haben diese die Eigenschaft ihre toten auf Haufen anzusammeln. In einem Experiment zur Ermittlung wie sich Schwärme auf Veränderungen von Lebensbedingungen anpassen wurde dabei verglichen, wie ein Ameisenschwarm die Toten anordnet wenn keine äußeren Einflüsse vorhanden sind und wenn ein konstanter Luftstrom aus einer bestimmten Richtung einwirkt. Dabei konnte beobachtet werden, dass die Anordnung der Haufen unter der Zufuhr des Luftstromes anders erfolgte, wie in Abbildung 3.3 zu sehen ist.

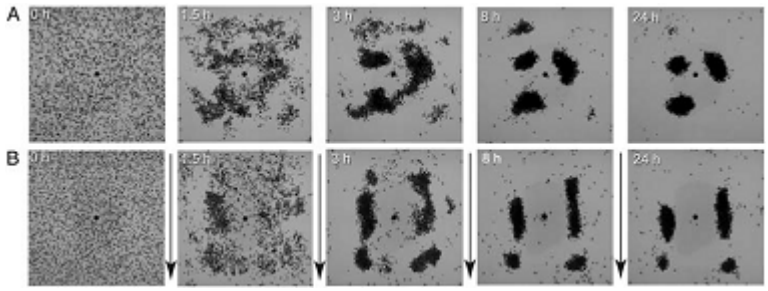

ABBILDUNG 3.3: Experiment zur Anpassung von Ameisen an Veränderungen der Lebensbedingungen. Reihe A zeigt das Experiment ohne Luftstrom und Reihe B mit einem Luftstrom in Richtung der Pfeile (von [7])

Das Experiment zeigte, dass die Stelle an der eine Ameise den Körper einer anderen toten Ameise ablegte durch die Luftzufuhr beeinflusst wurde. So ist die Wahrscheinlichkeit dort am höchsten, wo der Luftzug am geringsten ist, also auf der windstillen Seite von bereits existierenden Haufen. Dies hatte zur Folge das im Gegensatz zur Versuchsreihe A die Haufen eine weniger runde Form angenommen haben, sondern entlang der Richtung des Luftstroms in die Länge gezogen wurden um eine Form zu erreichen die insgesamt weniger Angriffsfläche bietet, was eine

intelligente Anpassung des Schwarms auf die veränderten Lebensbedingungen darstellt.

Bei diesem Experiment handelte es sich um eine Anpassung auf äußere Einflüsse. Doch wie sieht es bei Einflüssen innerhalb des Schwarms aus? Hierzu wurde das Arbeitsverhalten von Wespen in Schwärmen abhängig von der Schwarmgröße beobachtet. Dabei wurde zuerst einmal beobachtet, dass jedes Individuum für bestimmte Aufgaben eine gewisse Schwelle besitzt die durch einen Reiz überwunden werden muss, damit dieses Individuum diese Arbeit verrichtet. Dabei sinkt diese Schwelle für ein Individuum während es über einen Zeitraum hinweg diese Arbeit verrichtet, wohingegen die Schwelle für eine Arbeit steigt, während eine andere Arbeit oder gar keine Arbeit verrichtet wird. Dies hat zur Folge, dass wenn mehrere Individuen für eine Arbeit in Frage kommen, dass das Individuum, welches die niedrigste Schwelle und damit auch die höchste Spezialisierung, zuerst diese Arbeit beginnt. Im Rahmen der Beobachtungen entdeckte man dabei, dass in größeren Schwärmen Individuen zu mehr Spezialisierungen neigen. So sieht die normale Aufgabenreihe beim Nestbau so aus, dass zuerst Wasser beschafft werden muss, anschließend Zellstoff und abschließend daraus die Masse zum Bau einer Zelle geformt wird. In großen Schwärmen sieht man dabei, wie jede dieser drei Aufgaben von verschiedenen Individuen verrichtet wird. Die Ursache ist, dass aufgrund der Schwarmgröße auch eine große Menge an Aufgaben vorhanden ist, die es erlaubt das Wespen länger eine Arbeit am Stück ausführen und sich dadurch stark spezialisieren. Wenn jedoch die Schwarmpopulation geringer ist, sind auch weniger Aufgaben vorhanden, was zur Folge hat, dass weniger Möglichkeit zur Spezialisierung besteht und Individuen mehr verschiedene Aufgaben verrichten. So werden in kleineren Schwärmen die drei Arbeitsschritte beim Nestbau häufiger von einem einzelnen Individuum verrichtet. Diese Verhaltensweisen zeigen eine intelligente Anpassung des Schwarms an die unterschiedlichen Anforderungen in Abhängigkeit zur Schwarmgröße und den damit vorhandenen Aufgaben. In einem kleinen Schwarm sind weniger Aufgaben vorhanden, wodurch es zu ineffizient ist, viele spezialisierte Arbeiter zu haben, die zu oft ohne Beschäftigung in ihrem speziellen Aufgabenbereich wären. Stattdessen werden viele allgemeine Arbeiter für alle Arbeiten benötigt, mit der Fähigkeit zwischen verschiedenen Aufgaben zu wechseln. Bei größeren Schwärmen hingegen ist genug Arbeit vorhanden, um spezialisierten Arbeitern Beschäftigung zu bieten, sodass diese Art der Arbeitsverteilung dort deutlich effizienter ist.

Zusammenfassend lässt sich als Erkenntnis aus den Formen von Schwarmverhalten in der Natur erkennen, dass die darauf basierenden Systeme sowohl sehr rubust dadurch sind, dass das Versagen eines einzelnen Individuums sehr leicht durch die zahlreichen Interaktionen zwischen den verschiedenen Individuen abgefangen werden kann, als auch zugleich sehr anpassungsfähig sind aufgrund des Einflusses der verschiedenen Formen von Feedback als auch die Anpassung des individuellen Verhaltens auf die lokalen Gegebenheiten und dessen Veränderungen. Beides sind Eigenschaften, die auch für Algorithmen von Vorteil sind, wodurch Schwarmverhalten ein so interessantes Konzept für die Informatik darstellt.

3.1.2 Anwendung von Schwarmintelligenz in der Informatik

Auf Grund der Eigenschaften von auf Schwarmverhalten basierenden Systemen, gab es auch bereits viele Ansätze, dieses Konzept für Algorithmen zu verwenden.

Ein populäres Beispiel dafür ist der Partikelschwarm, der sich besonders am Schwarmverhalten von Menschen orientiert. In diesem gibt es zwei Komponenten in Analogie zu den Systemen aus der Natur. Die High-Level Komponente umfasst das Entstehen von Mustern über die Individuen hinweg, bis hin zur Problemlösung, wohingegen die Low-Level Komponente sich mit den möglichen Verhaltensweisen des Individuums befasst. Letztere umfasst dabei drei Prinzipien (vgl. [8]). Das Erste ist, zu evaluieren wie es jedes Lebewesen mit jedem eingehenden Reiz tut. Ein weiteres Prinzip ist sich selbst als Individuum mit anderen zu vergleichen, die bei ihrer Evaluation ein besseres Ergebnis erzielt haben. Das letzte Prinzip ist das Imitieren, eines das in der Natur vorrangig beim Menschen und einigen wenigen Vogelarten auftritt. Dabei ist nicht nur das einfache Imitieren einer Handlung gemeint, sondern auch die zugrunde liegende Intention dahinter zu erfassen. Es sei angemerkt, dass sich in diesen Prinzipien auch bereits drei der vier beschriebenen Voraussetzungen für Selbstorganisation in Schwärmen wiederfinden lassen. So entsteht positives und negatives Feedback beim Evaluieren und Vergleichen und ebenso findet beim Vergleichen und Imitieren eine direkte oder indirekte Interaktion zwischen Individuen statt.

Der Partikelschwarm baut auf einzelnen Individuen (auch Partikel genannt) auf, die jedes für sich versuchen eine optimale Lösung zu finden. Nach der Berechnung einer Lösung evaluiert jedes Individuum des Schwarms sein Ergebnis anhand einer Bewertungsfunktion. Im Falle einer Pfadplanung könnte man sich eine solche Funktion beispielsweise als den euklidischen Abstand zum Zielpunkt vorstellen. Nachdem alle Lösungen evaluiert wurden folgt der Vergleich der Individuen untereinander. Dabei gibt es verschiedene Modelle wie lokales Vergleichen mit n anliegenden Nachbarn oder globales Vergleichen mit allen anderen Schwarmmitgliedern. Anschließend erfolgt eine Neuberechnung der Lösung wobei jedes Individuum zwei vorhandene Informationen einfließen lässt in die Berechnung. Bei diesen handelt es sich zum einen um die eigene Lösung die bisher das beste Ergebnis bei der Evalutionsfunktion erzielt hat und zum anderen um die bisher beste Lösung der Nachbarschaft in der verglichen wird. Dabei kann der Einfluss von beiden durch verschiedene Gewichtungen modifiziert werden. An dieser Stelle kommt auch die letzte Voraussetzung der Fluktuation hinzu, dadurch dass Individuen zwar von dem jeweils Besten aus der Nachbarschaft in eine Richtung des Lösungsraums gezogen werden, jedoch der Einfluss der eigenen besten Lösung und der eigenen aktuellen Lösung, welche als Ausgangspunkt dient, es ermöglicht noch neue bessere Lösungen zu finden.

```
Loop
  For i = 1 to number of individuals
    if G(x_i) > G(p_i) then do              //G() evaluates goodness
      For d = 1 to dimensions
        p_id = x_id                         //p_id is best so far
      Next d
    End do

    g = i                                   //arbitrary
    For j = indexes of neighbors
      If G(p_j) > G(p_g) then g = j         //g is index of best performer
                                            in the neighborhood
    Next j
    For d = 1 to number of dimensions
      v_id(t) = v_id(t − 1) + φ1(p_id − x_id(t − 1)) + φ2(p_gd − x_id(t − 1))
      v_id ∈ (−V_max, + V_max)
      if ρ_id < s(v_id(t)) then x_id(t) = 1; else x_id(t) = 0;
    Next d
  Next i
Until criterion
```

ABBILDUNG 3.4: Pseudocode zum Partikelschwarm für einen Binärstring (von [8])

In Abbildung 3.4 ist ein Pseudocode für den Partikelschwarm beschrieben, bei dem nach einem optimalen String aus Binärzahlen gesucht wird. Dabei bezeichnet der Vektor $\overrightarrow{x}$ den aktuellen und $\overrightarrow{p}$ den bisher am besten bewerteten Binärstring (jeweils für Individuum i an Position d). Bei $\overrightarrow{v}$ handelt es sich um einen Vektor aus Wahrscheinlichkeiten bei der nächsten Iteration eine Eins an Stelle d des Binärstrings zu wählen und als Gegenstück ist $\overrightarrow{\rho}$ ein Vektor von zufällig gewählten Wahrscheinlichkeiten, welche als Schwellwerte fungieren. ϕ_1 und ϕ_2 sind Zufallswerte zur Gewichtung des Einflusses der bislang besten Lösungen. Der Partikelschwarm durchläuft in jeder Iteration drei Schritte für jedes Individuum. Im Ersten wird die eigene Lösung evaluiert und mit der bisher eigenen besten Lösung verglichen. Ist das neue Ergebnis besser wird die bisher beste Lösung überschrieben. Anschließend wird die eigene beste Lösung mit der aller Nachbarn verglichen und sich der Index des Individuums mit der bisher besten Lösung der gesamten Nachbarschaft gemerkt. Im letzten Schritt wird der neue Wahrscheinlichkeitsvektor $\overrightarrow{v}$ berechnet aus dem Wert der vorherigen Iteration sowie dem Abstand zu den besten Lösungen des Individuums und der Nachbarschaft. Dabei wird überprüft, dass die Werte ein gegebenes Intervall nicht verlassen und anschließend mit der Sigmoid-Funktion gewährleistet, dass sich alle Werte zwischen null und eins befinden bevor der Vektor mit $\overrightarrow{\rho}$ stellenweise verglichen wird um zu ermitteln, welche Zahl an der jeweiligen Stelle der Lösung in der nächsten Iteration steht. Es ist möglich diesen Code auch für einen String aus reellen Zahlen zu verwenden. Dafür wird lediglich der Wahrscheinlichkeitsvektor durch einen Geschwindigkeitsvektor ersetzt der die Richtung des Partikels im Lösungsraum angibt. Dieser wird weiterhin in jeder Iteration anhand der bisherigen besten Lösungen neu berechnet und zu der Lösung der vorherigen Iteration hinzu addiert.

Zur Performanz des Partikelschwarm Algorithmus lässt sich dabei sagen, dass

er einen Beweis für die Leistungsfähigkeit auf Schwarmintelligenz basierender Algorithmen geliefert hat. In einem Versuch wurden Binärstrings mit den Längen 20 und 100 Stellen erzeugt, wobei es jeweils Versuche mit 20 und 100 optimalen Lösungen gab. Die Aufgabe der Algorithmen war eine dieser optimalen Lösungen zu finden. Verglichen wurde dabei der Partikelschwarm Algorithmus mit drei verschiedenen Arten von genetischen Algorithmen (einen auf Mutation basierenden, einen auf Kombination basierenden und einen Hybriden aus Mutation und Kombination).

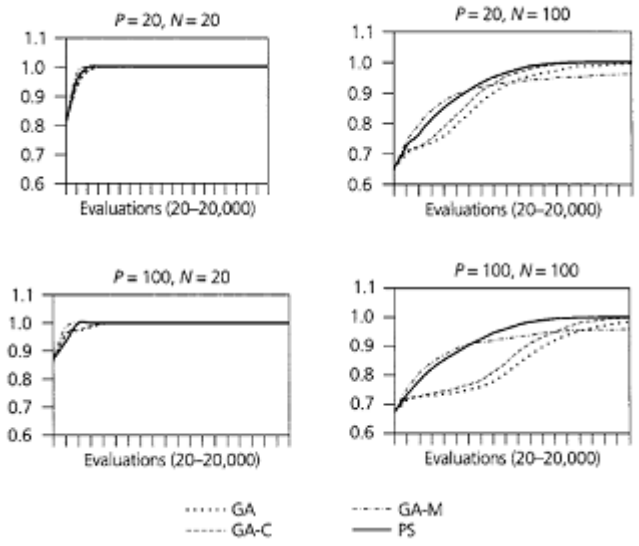

ABBILDUNG 3.5: Testergebnisse des Vergleichs verschiedener Algorithmen zum Finden optimaler Lösungen in Form von Binärstrings (von [8])

Dabei war der Partikelschwarm Algorithmus für Binärstrings der Länge 20 vergleichbar zu den genetischen Algorithmen und konnte vor allem bei Tests mit 100 Stellen langen Strings die besten Ergebnisse aufweisen.

3.2 Neuronale Netze

Bei neuronalen Netzen handelt es sich um ein Forschungsgebiet das mit der Entwicklung leistungsfähigerer Technik in den vergangenen Jahrzehnten zunehmend an Bedeutung gewonnen hat. Dabei gehen die Anfänge dieser Idee bis ins Jahr 1943 zurück, als Warren McCulloch und Walter Pitts ein erstes Modell von künstlichen Neuronen präsentierten. Mittlerweile finden sie besondere Beachtung in den Bereichen des maschinellen Lernens und der künstlichen Intelligenz. Der Aufbau von neuronalen Netzen orientiert sich dabei stark an der Funktionsweise des Gehirns und dessen Art der Informationsverarbeitung. Diese basiert auf tausenden bis zu Millionen miteinander verbundener Zellen. Dabei ist jede von ihnen in der Lage, eingehende Signale auf verschiedene Arten zu bearbeiten. Jedoch kommt die hohe Leistungsfähigkeit des Gehirns dabei durch die enorme hierarchische Netzwerkstruktur, welche komplexes Verhalten ermöglicht.

Der Aufbau eines Neurons umfasst im wesentlichen einen Zellkörper, der die chemischen Stoffe zur Arbeit bereitstellt, den Dendriten, über die Signale empfangen werden, dem Axon, das die Ausgangssignale des Neurons übermittelt, und den Synapsen die die Kontaktpunkte zwischen verschiedenen Neuronen bilden. Die Synapsen übernehmen dabei auch die Lagerung der Informationen des neuronalen Netzes. Wenn man diese Struktur nun auf ein künstliches neuronales Netz überträgt, stellen die Dendriten die eingehenden Informationen und das Axom die ausgehenden dar. Dazwischen liegt mit dem Zellkörper die Bearbeitung der Informationen in Form einer mathematischen Gleichung. Die Synapsen bilden analog zum Verbindungspunkt zwischen Neuronen die Verbindungen der mathematischen Funktionen.

Diese Netzwerke brachten, durch bereits vorhandene Modelle wie Logikgatter, jedoch vergleichbar wenig neue Berechnungsmöglichkeiten [9]. Erst mit Frank Rosenblatts Ansatz des Perzeptrons, welcher sich am menschlichen Sehen beziehungsweise der Verarbeitung von Bildsignalen im Gehirn orientiert (siehe Abbildung 3.6) entstanden neue Möglichkeiten.

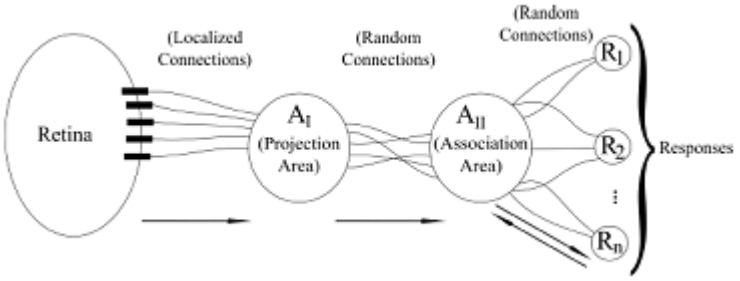

ABBILDUNG 3.6: Das klassische Perzeptron nach Rosenblatt (1958) [10]

Zu den Änderungen dieses Modells zählten das Einfügen numerischer Gewichte sowie eines speziellen Verbindungsmusters. Im Falle der Signalverarbeitung im menschlichen Gehirn werden die Signale von der Retina durch lokale Verbindungen in den Projektionsbereich geleitet von dem aus eine Weiterleitung durch zufällige, gewichtete Verbindungen erfolgt. Die dort ankommenden Signale werden in aufsummierter Form mit einem Schwellwert verglichen und bei Überschreitung als Ausgangssignal ausgegeben. Für künstliche neuronale Netze bedeutet das Hinzufügen der Gewichte eine Möglichkeit des Lernens für das Netzwerk.

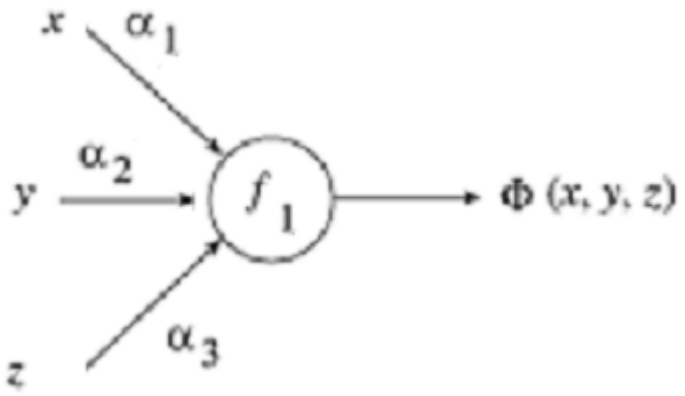

ABBILDUNG 3.7: Beispielmodell eines einzelnen Perzeptrons

In Abbildung 3.7 ist ein einzelnes Perzeptron mit Verwendung von Gewichten zu sehen. Dabei gibt es verschiedene Eingangsdaten in Form der Variablen x, y und z welche durch die Funktion verarbeitet und ausgegeben werden. Es ist möglich mit diesem einzelnen Perzeptron zu lernen, indem die Gewichte abhängig von Eingangsdaten und Ergebnis entsprechend angepasst werden. Jedoch lassen sich einige Funktionen wie zum Beispiel die logischen Operationen XOR oder XNOR nicht durch ein einzelnes Perzeptron abbilden sonder erfordern mehrere Schichten von Perzeptronen (vgl. [9]). Ein Algorithmus zum Lernen in mehrschichtigen neuronalen Netzen ist der Backpropagation-Algorithmus.

3.2.1 Der Backpropagation-Algorithmus

Der Backpropagation-Algorithmus ist ein populärer Ansatz zum Lernen in neuronalen Netzen mit Hilfe von Gradientenabstieg. Dabei ist das Ziel die Gewichte des Netzes anzupassen in Abhängigkeit einer hereingegebenen Trainingsmenge und des Abstands der für diese Daten berechneten Ergebnisse zu den für die Daten erwarteten Ergebnissen. Dazu wird eine Fehlerfunktion angewandt, deren Wert es durch das Anpassen der Gewichte zu minimieren gilt. Die Notwendigkeit den Gradienten dieser Fehlerfunktion zu berechnen, hat zur Folge, dass diese Funktion sowohl stetig als auch differenzierbar sein muss. Eine der am häufigsten verwendeten Funktionen ist die in Gleichung 3.1 dargestellte Sigmoid-Funktion, die für einen Wert x eine reelle Zahl zwischen null und eins als Ergebnis liefert. Es kann allerdings auch jede andere beliebige differenzierbare und stetige Funktion verwendet werden.

$$s_c(x) = \frac{1}{1 + e^{-cx}} \tag{3.1}$$

Der Parameter c in der Gleichung ist dabei eine Konstante die die Form der Sigmoid Funktion beeinflusst. Ein höherer Wert nähert die Sigmoid-Funktion dabei einer abschnittsweise definierten Funktion 3.2 an.

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3.2}$$

Die später benötigte Ableitung der Sigmoid-Funktion is definiert durch:

$$\frac{d}{dx}s(x) = \frac{e^{-}x}{(1+e^{-x})^2} = s(x)(1-s(x)) \tag{3.3}$$

Bei einem neuronalen Netz handelt es sich um einen Graphen, dessen Knoten als Recheneinheiten betrachtet werden können und deren gerichtete Kanten numerische Informationen von Knoten zu Knoten transportieren. Dabei ist jeder Knoten in der Lage eine primitive Funktion auf einen eingehenden Wert auszuführen. Im Gesamten betrachtet ist das neuronale Netz also eine Verkettung von Funktionsverknüpfungen die eine eingehende Menge von Daten zu einer ausgehenden Datenmenge transformiert. Diese Transformation von den eingehenden zu den ausgehenden Daten kann als Funktion des neuronalen Netzes betrachtet werden und beim Lernproblem gilt es diese durch das Finden der richtigen Kantengewichte bestmöglich zu approximieren, wobei die Funktion nicht explizit gegeben ist. Stattdessen wird die gewünschte Funktion durch vorhandene Trainingsdaten impliziert [9]. Diese bestehen aus einem Set $\{(x_1, t_1), ..., (x_p, t_p)\}$ aus sortierten Paaren von n- und m-dimensionalen Vektoren, wobei n die Größe des Eingabe- und m die des Ausgabevektors ist. Wenn ein Eingabevektor x_i in das neuronale Netz gegeben wird, wird anhand der Funktionen und verschiedenen Kantengewichte, welche zu Beginn zufällig initialisiert werden, ein Ausgabevektor o_i berechnet. Ziel ist es nun die Abstände von den verschiedenen berechneten Ausgabevektoren zu den erwarteten Ausgabevektoren des Trainingssets zu minimieren. Dafür wird mit Funktion 3.4 der Gesamtfehler des Netzes errechnet, welcher sich aus der Hälfte der Summe der quadratischen Abstände der Ausgabevektoren ergibt:

$$E = \frac{1}{2}\sum_{i=1}^{p} ||o_i - t_i||^2 \tag{3.4}$$

Nachdem ein Minimum für diese Funktion gefunden wurde, wird erwartet, dass für neue unbekannte Eingabevektoren Ähnlichkeiten zur Mustern der Trainingsmenge erkannt werden und ein entsprechend passender Ausgabevektor berechnet wird. Dabei wird der Backpropagation-Algorithmus benutzt um ein lokales Minimum zu finden. Es wird ein Gradient der Fehlerfunktion ermittelt und dieser benutzt, um die vorhandenen Kantengewichte anzupassen[9].

Im Ersten Schritt wird hierfür das Netz durch Hinzufügen einer weiteren Schicht nach dem Ausgabevektor erweitert. Diese besitzt dieselbe Größe wie die Ausgabeschicht und jedes Neuron erhält dabei den Wert von genau einem Neuron der Ausgabe und wendet auf diesen die Fehlerfunktion an. Auf diese Weise wird komponentenweise der Fehler des Ausgabevektors ermittelt und von allen Neuronen weitergeleitet zu einer Funktion, welche die Summe aller Komponentenfehler für die Eingabe i bildet. Eine Recheneinheit bildet wiederum die Summe der Fehler für alle Eingaben $E_1, ..., E_p$, was den Gesamtfehler des neuronalen Netzes bildet.

Da die einzigen veränderlichen Komponenten des neuronalen Netzes die Kantengewichte sind, gilt es nun anhand dieser das Minimum der Fehlerfunktion zu ermitteln. Diese ist durch die vorher getroffenen Einschränkungen eine stetige und differenzierbare Funktion über die l Gewichte $w_1, w_2, ..., w_l$, was es ermöglicht die Minimierung durch iterativen Gradientenabstieg vorzunehmen. Der Gradient errechnet sich hierbei durch

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l}). \tag{3.5}$$

Die Gewichte werden für eine gegebene Lernkonstante γ angepasst:

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} \text{ für } i = 1, ..., l. \tag{3.6}$$

Diese Gewichtsanpassung erfolgt in jedem Iterationsschritt für alle Gewichte. Der Gradient wird dabei unter Anwendung der Kettenregel (vgl. [9]) vom Ausgang des Netzes über alle Schichten bis zur Eingabeschicht weiterberechnet. Am Ausgangspunkt des Netzes lässt sich der Gradient wie folgt berechnen:

$$\frac{\partial E}{\partial o_i} = o_i - t_i. \tag{3.7}$$

Wenn an dieser Stelle erst einmal von einem einfachen neuronalen Netz ohne versteckte Schichten zwischen der Eingabe- und der Ausgabeschicht ausgegangen wird, kann sich durch die Kettenregel der Gradient eines Eingabevektors $\vec{x_i}$ für die Stelle l nach Gleichung 3.8 berechnen lassen aus dem Gradienten am Ausgabepunkt und dem Gradienten Eingabepunkt.

$$\frac{\partial E}{\partial x_{il}} = \frac{\partial E}{\partial o_i} * \frac{\partial o_i}{\partial x_{il}} * w_l \tag{3.8}$$

Da in diesem Fall für die Funktion am Ausgabepunkt die Sigmoid-Funktion gewählt wurde, kann dessen Ableitung aus Gleichung 3.3 für den entsprechenden Gradienten verwendet werden, sodass gilt:

$$\frac{\partial o_i}{\partial x_{il}} = o_i * (1 - o_i). \tag{3.9}$$

Dies lässt sich nun in die Gleichung 3.8 einsetzen. Da das Ziel ist, das Gewicht w_l anpassen zu können, wird dabei außerdem die Ableitung umgestellt, sodass sich ergibt:

$$\frac{\partial E}{\partial w_l} = (o_i - t_i) * o_i * (1 - o_i) * x_{il}. \tag{3.10}$$

Schließlich kann die Gleichung 3.6 zur Anpassung des Gewichts mit Gleichung 3.9 kombiniert werden um die abschließende Formel

$$\Delta w_l = -\gamma (o_i - t_i) * o_i * (1 - o_i) * x_{il} \tag{3.11}$$

zu erhalten.

Im Falle von neuronalen Netzen mit mehr Schichten lässt sich die Backpropagation auf die gleiche Weise unter Anwendung der Kettenregel verwenden. Außerdem ist es möglich neben der Lernkonstante noch einen zweiten Parameter in Form eines Momentums einfließen zu lassen, wodurch beim Gradientenabstieg eine Art Schwung aufgenommen werden kann, der es zum Beispiel möglich macht, lokale Minima zu überwinden um mit größerer Wahrscheinlichkeit ein globales Minimum zu erreichen. Dafür wird die Gleichung 3.6 der Gewichtsberechnung auf

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} + \alpha \Delta w_{lk}(t-1) \tag{3.12}$$

erweitert, wobei α der Parameter für das Momentum ist (nach [10]).

Kapitel 4

Schwarmintelligenz in Form eines neuronalen Netzes

4.1 Ansatz zur Schwarmintelligenz innerhalb eines neuronalen Netzes

In Kapitel 3.2 wurden die Strukturen und eine mögliche Implementation von neuronalen Netzen erläutert. Werden diese nun noch einmal unter dem Kontext von Schwarmverhalten betrachtet, so fällt auf, dass viele grundlegenden Konzepte wiedergefunden werden können. So wurden beispielsweise als Voraussetzung für Selbstorganisation in Schwärmen positives und negatives Feedback, sowie Interaktionen zwischen Individuen und Fluktuation benannt (siehe Kapitel 2.1). Beide Formen von Feedback lassen sich bei neuronalen Netzen in der Fehlerberechnung des Netzes wiederfinden und sowohl beim Berechnen des Ausgabevektors für eine Eingabe als auch bei der Backpropagation werden Informationen zwischen den Neuronen weitergeleitet was einer Interaktion zwischen ihnen entspricht. Selbst die Fluktuation lässt sich ebenfalls leicht erzielen, durch beispielsweise eine zufällige Auswahl der Eingabevektoren oder die Beeinflussung der Parameter für die Lernkonstante und das Momentum. Eine weitere Parallele ist, dass der Beitrag eines Individuum des Schwarms oder in diesem Fall eines Neurons des Netzes ganz elementare Aktionen wie das Ausführen einer Funktion und die Weitergabe des Ergebnisses sind, das Ergebnis des gesamten Systems jedoch viel komplexere Strukturen wie eine Bilderkennung oder Klassifizierungen sein können.

Doch stellt sich die Frage, ob neuronale Netze auch dabei helfen können, ein schwarm-spezifisches Problem zu lösen, wie die positionelle Einordnung eines Individuums zwischen anderen Mitgliedern des Schwarms, beziehungsweise im Kontext dieser Arbeit das Einordnen eines autonomen Fahrzeugs zwischen anderen Fahrzeugen im Straßenverkehr. Notwendig dazu wäre das Erlernen der elementaren Verhaltensregeln, die dafür sorgen, das Verkehrsteilnehmer sich angemessen einordnen. Es wäre möglich zu versuchen diese zu formulieren und zu testen, aber in dieser Arbeit wurde als Ansatz versucht durch ein neuronales Netz diese elementaren Regeln zu erlernen, basierend auf der Verhaltensweise anderer menschlicher Teilnehmer des Straßenverkehrs. Dabei sollen die elementaren Verhaltensregeln nicht direkt formuliert werden, sondern stattdessen in Form der Gewichte des neuronalen Netzes erlernt und auf neue Eingaben angewandt werden. Der Vorteil ist, dass ein gut trainiertes Netz in der Lage wäre, mit geringem Aufwand für jede Verkehrssituation eine Position angeben zu können, da lediglich für einen einzelnen Eingabevektor eine Ausgabe berechnet werden muss. Im Vergleich dazu müsste zum Beispiel der Partikelschwarm für jede Situation erneut über viele Iterationen nach einer

optimalen Lösung suchen und müsste dabei auch die elementaren Verhaltensregeln kennen um eine gute Evaluierungsfunktion definieren zu können.

4.1.1 Die Trainingsdaten

Wie in 4.1 beschrieben, soll für die Trainingsmenge die Verhaltensweise von menschlichen Autofahrern dienen. So sollen aus Verkehrssituationen mit anderen Autos Bilder als Eingangsdaten zum Trainieren erzeugt werden, aus denen jeweils ein Auto entfernt wurde und dessen Position als gesuchtes Ergebnis (auch Label genannt) genommen wird. Die daraus resultierende Vorgehensweise weist parallelen zu den Grundschritten des Partikelschwarm-Algorithmus auf. So finden die Evaluierung und das Vergleichen mit anderen Individuen dadurch statt, dass die Position der anderen Verkehrsteilnehmer die Label für die Trainingsdaten bilden. Somit lassen sich beide Schritte in jedem Berechnen der Fehlerfunktion des Netzes wiederfinden. Das Imitieren findet dabei in der Anpassung der Gewichte in Abhängigkeit zum Fehler statt, da dieser wiederum auf den Positionsdaten der anderen Autos basiert. Somit werden in jeder Iteration Änderungen der Gewichte vorgenommen, um das Ergebnis der Positionierung näher an das der anderen Autos heranzubringen. Die Basis für diesen Ansatz liefern zahlreiche ROS Bag Dateien von Fahrten des MadeInGermany Autos durch Berlin. Für diese Arbeit wurden dabei eine Fahrt vom Flughafengelände Tempelhof zur Freien Universität Berlin sowie eine weitere Fahrt vom Bundesplatz zum Flughafengelände verwendet. In diesen Dateien befinden sich neben anderen Daten, auch die Scandaten der Ibeo Lux-Laserscanner, welche Hindernisse in der Umgebung aufzeichnen und somit auch die Position anderer Autos. Weiterhin ließen sich bereits vorhandene Pakete aus der Arbeitsgruppe der Autonomous Cars verwenden, welche Nachrichten über Hindernisse (eine obstacle ROS message) basierend auf den Ibeo Scandaten versenden und Informationen wie Positionsdaten, Ausrichtung, Geschwindigkeit und Art des Hindernisses enthalten. Ein weiteres Paket bot einen Subscriber zu dem entsprechenden ROS Topic der Hindernisdaten und transferierte diese in ein CSV Dateiformat (siehe Anhang 1). Außerdem gab es bereits versuchsweise eine Anwendung, die aus dieser CSV Datei Hindernisse ausliest und für jeden Zeitpunkt in dem als Auto klassifizierte Hindernisse vorhanden sind, Bilder generiert. Dabei wird ein als Label bezeichnetes Bild mit allen Hindernissen erzeugt, sowie für jedes vorhandene Auto, dessen relative Geschwindigkeit in einem festgelegten Rahmen liegt, ein weiteres Bild auf dem genau dieses eine Auto weggelassen wurde. Durch die Geschwindigkeitseinschränkung wird dabei verhindert, dass Autos aus dem Gegenverkehr als Label genommen werden können. Da dies bereits genau die Trainingsbilder lieferte, die als Ansatz für diese Arbeit genommen werden sollten, waren nur noch Modifikationen an dem Format der Bilder notwendig um letztendlich 201x201 Pixel große schwarzweiß Bilder (mit ausschließlich den Pixelwerten 0 und 255) zu erzeugen.

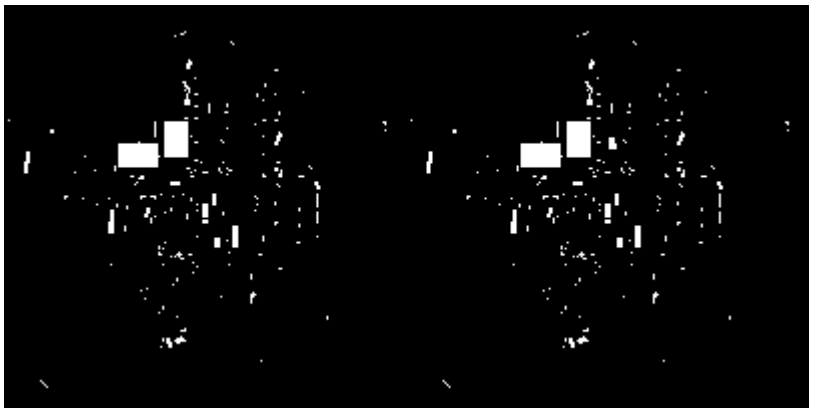

ABBILDUNG 4.1: Erzeugtes Trainingsbild mit einem weggelassenen Auto (links) und zugehöriges Labelbild mit allen Autos (rechts)

In Abbildung 4.1 sind ein Trainingsbild mit dem dazugehörigen Labelbild zur Veranschaulichung dargestellt. Dabei wurde im Trainingsbild das oberste Auto im Bild weggelassen. Zusätzlich zu dem Labelbild wurde aus Gründen der Performanz des neuronalen Netzes als alternative Label noch die Erzeugung einer CSV Datei zu dem Code hinzugefügt, in die für jedes Trainingsbild der Name sowie die X- und Y-Position des Pixels der unteren rechten Ecke des weggelassenen Autos als Label eingetragen wird, was es ermöglicht als Label mit zwei Werten zu arbeiten im Gegensatz zu den 40401 Werten des 201x201 Pixel großen Labelbilds.

4.1.2 Das Neuronale Netz

Als Ansatz für den Versuch wurde ein simples neuronales Netz basierend auf dem Backpropagation-Algorithmus unter Verwendung der Sigmoid-Funktion gewählt um die Funktionalität von neuronalen Netzen für diese Aufgabe zu testen. Dafür wurde eine bereits vorhandene Implementation eines neuronalen Netzes verwendet [12] und so angepasst, dass die Trainingsdaten und dazugehörigen Label importiert und verwendet werden. Dabei werden die Trainingsbilder in ein 2-dimensionales Array mit den Größen 40401 und der Anzahl der zu verwendenden Trainingsbilder umgewandelt. Das gleiche wurde anfangs auch mit den Labelbildern getan, jedoch zeigte sich nach einigen Testversuchen, dass die benötigte Anzahl von Neuronen dadurch deutlich zu hoch sein würde um den Rechenaufwand in einer annehmbaren Zeitspanne bewältigen zu können. Als Folge dessen wurde sich für die Label stattdessen auf eine Positionskoordinate beschränkt, wodurch die Größe der Ausgabeschicht des Netzes von 40401 auf 2 reduzieren ließ. Für eine leichtere Verwendung wurden außerdem alle Trainingsdaten auf float Werte zwischen null und eins normiert.

4.2 Daten und Auswertung

Für den ersten Versuch mit dem neuronalen Netz wurde eine sehr kleine Trainingsmenge verwendet. So wurden lediglich drei Zeitpunkte der Fahrt zufällig ausgewählt und alle Trainingsbilder dieser Zeitpunkte verwendet, so dass das Trainingsset aus ingesamt zwölf Bildern bestand. Dabei wurden im Netz 500 Neuronen in der Eingangsschicht sowie 250 in einer versteckten Schicht verwendet. Das Training durchlief dabei 1000 Iterationen und verwendete als Lernkonstante 0.01 sowie ein

Momentum von 0.005.

TABELLE 4.1: Erwartete und tatsächliche Ergebniswerte des ersten Trainingssets

Trainingsbild	**Zeitpunkt**	**Erwartete X- / Y-Koordinate**	**Berechnete X- / Y-Koordinate**
1	1	101/76	103/77
2	1	101/122	107/112
3	1	104/117	107/112
4	2	100/73	94/73
5	2	104/70	93/68
6	2	101/113	106/110
7	2	104/99	107/99
8	2	104/112	106/110
9	3	101/105	104/102
10	3	104/93	99/92
11	3	104/108	106/109
12	3	102/68	91/65

Ein Vergleich der Werte in Tabelle 4.1 zeigt, dass in etwas über der Hälfte der Trainingsbilder das weggelassene Auto in geringem Abstand zu der gewünschten Position gesetzt worden wäre. Dies lässt bereits vermuten, dass das neuronale Netz noch nicht optimal trainiert ist und die Parameter besser angepasst werden könnten. Doch für eine genauere Interpretation lässt sich ein Blick auf die Abbildungen 4.1, 4.2 und 4.3 werfen.

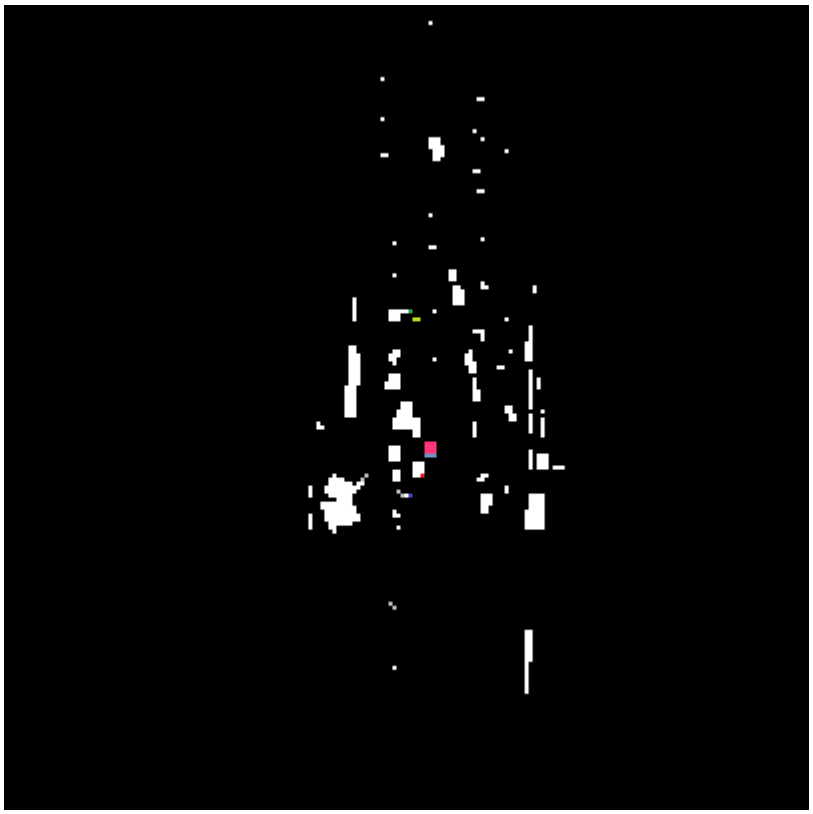

ABBILDUNG 4.2: Graphische Darstellung von Trainingsset 1 zum Zeitpunkt 1. Die vollständig farbigen Autos kennzeichnen die Positionen aus der Berechnung des neuronalen Netzes und die einzelnen farbigen Pixel in den weißen Autos die exakte Position des Labels. Die berechnete Position für die Autos blau und rot ist in diesem Fall nahezu identisch.

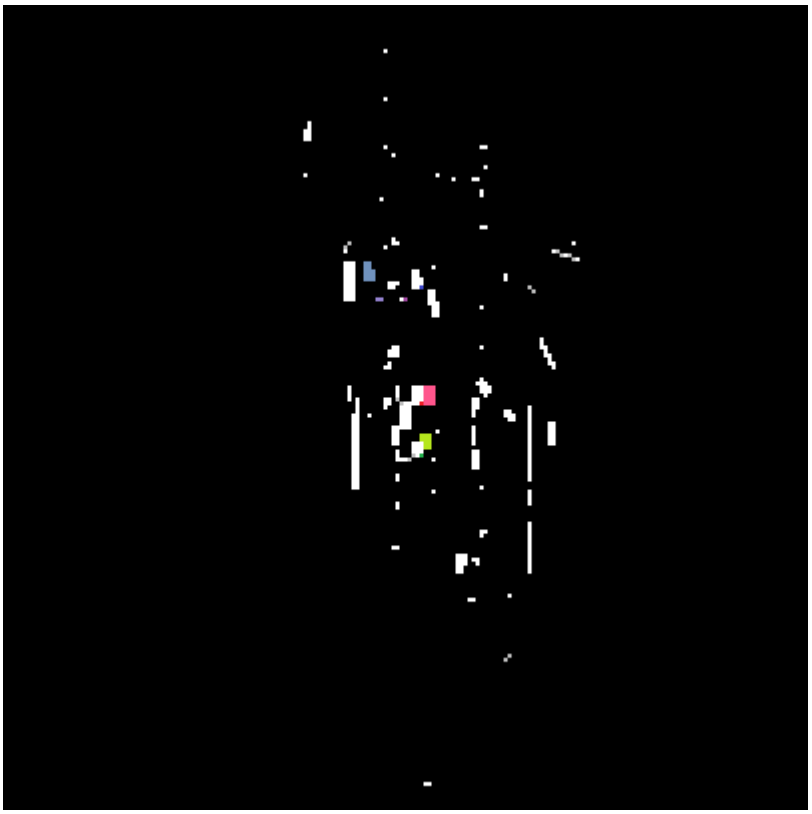

ABBILDUNG 4.3: Graphische Darstellung von Trainingsset 1 zum Zeitpunkt 2. Die vollständig farbigen Autos kennzeichnen die Positionen aus der Berechnung des neuronalen Netzes und die einzelnen farbigen Pixel in den weißen Autos die exakte Position des Labels.

Bei der Betrachtung der graphischen Auswertung lässt sich als allererstes feststellen, das in keinem einzigen Fall eine Position errechnet wurde, durch die ein Auto in ein anderes Hindernis hinein versetzt werden würde, was bedeutet, dass das neuronale Netz erst einmal die naheliegendste und wichtigste elementare Regel gelernt hat. Die nächste Auffälligkeit ist jedoch, das jedes Auto in der Spur beziehungsweise auf der X-Achse versetzt wurde. Bei den zentral gelegenen Autos in Abbildung 4.3 sieht dies erst einmal nach einer sinnvollen Positionierung aus, auch wenn das grüne Auto einen Pixel neben die Spur geraten ist. In Abbildung 4.4 wurde das grüne Auto zwar ordentlich zwischen die anderen beiden eingeordnet, jedoch drängt sich hierbei der Verdacht auf, das in dem Trainingsbild, in dem das grüne Auto entfernt wurde, das neuronale Netz nicht mehr in der Lage war, eine zweite Spur zu erkennen und deswegen eine Position zwischen den zwei anderen Autos berechnet hat. In Abbildung 4.1 konnten alle Autos genau in eine Spur eingeordnet werden, wobei auffällig ist, dass auch hier eine Tendenz zu Spurwechseln in die rechte Spur (aus Fahrtsicht linke) vorhanden ist. Die wahrscheinlichsten Gründe für diese Auffälligkeit sind aufgrund der geringen Trainingsmenge dessen Beschaffenheit und das Fehlen von klar erkennbaren Fahrspuren abseits der Positionierung der anderen Autos. Jedoch gibt es auch drei Fälle von falsch positionierten Autos. In Abbildung 4.3 wurden sowohl das blaue als auch das lilafarbene Auto nach links versetzt, obwohl diese Position anseits der Fahrspur liegt. Der wahrscheinlichste Grund

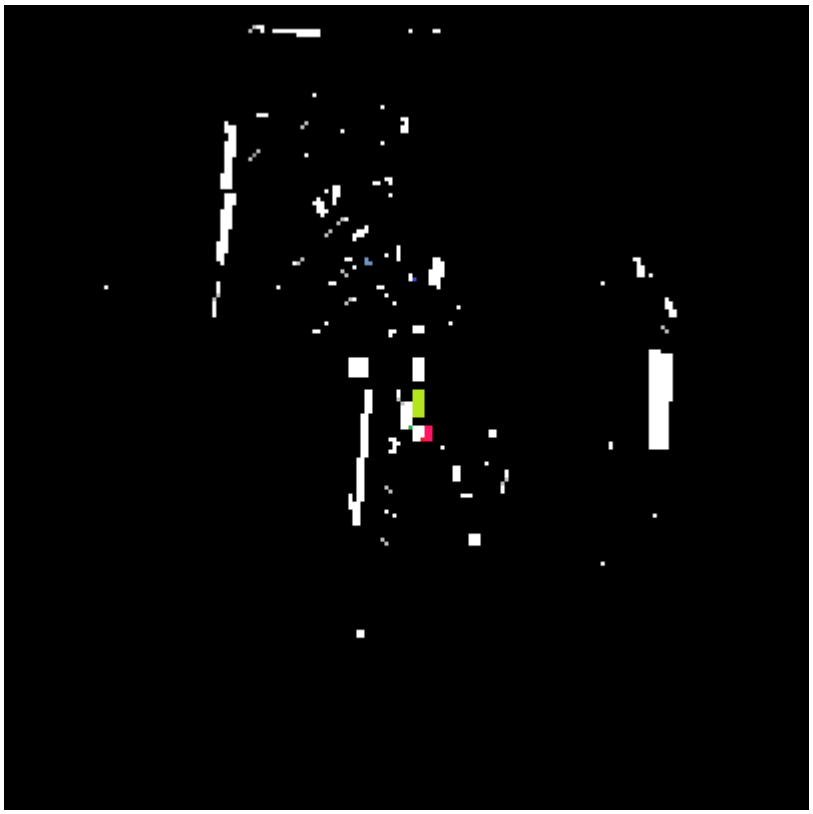

ABBILDUNG 4.4: Graphische Darstellung von Trainingsset 1 zum Zeitpunkt 3. Die vollständig farbigen Autos kennzeichnen die Positionen aus der Berechnung des neuronalen Netzes und die einzelnen farbigen Pixel in den weißen Autos die exakte Position des Labels.

dafür ist das angrenzende Hindernis, dass die für die Autos in den Trainingsdaten typische Rechteckform aufweist und dadurch als ein anderes Auto vom neuronalen Netz interpretiert werden könnte. Bei dem dritten falsch positionierten Auto handelt es sich um das blaue aus Abbildung 4.4 welches nach links zwischen eine Ansammlung von Hindernissen versetzt wurde. Eine mögliche Interpretation der Ursache ist die untypische Form des Autos durch drei Pixel, welche sich von denen aller anderen (selbst wenn das Rechteck nur aus zwei Pixeln besteht) unterscheidet.

Als erstes Zwischenfazit lässt sich an dieser Stelle bereits festhalten, dass das Konzept als solches im Rahmen dieser kleinen Trainingsmenge funktioniert hat. In den meisten Fällen wurden die Autos innerhalb von Spuren platziert und in den offensichtlich falschen Ergebnissen konnten wahrscheinliche Ursachen ausgemacht werden, denen sich entgegenwirken lässt (siehe Abschnitt 4.3).

In einem weiteren Versuch wurde die Größe der Trainingsdaten angepasst. So wurden nun insgesamt 100 Trainingsbilder aus der Fahrt vom Flughafengelände Tempelhof zur Freien Universität Berlin verwendet, sowie 50 Bilder aus der Fahrt vom Bundesplatz zum Flughafengelände als Validierungsset. Als Parameter wurden dabei 250 Neuronen in der Eingangsschicht, 100 Neuronen in der versteckten Schicht, 5000 Iterationen, eine Lernkonstante von 0.2 sowie ein Momentum von 0.1 verwendet. Bilder wurden dabei aus verschiedenen Zeitpunkten der Fahrten ausgewählt, wobei immer alle Bilder eines Zeitpunktes verwendet wurden um jede Position eines Autos aus dieser Situation einfließen zu lassen.

TABELLE 4.2: Erwartete und tatsächliche Ergebniswerte des zweiten Trainingssets

Trainingsbild	Zeitpunkt	Erwartete X- / Y-Koordinate	Berechnete X- / Y-Koordinate
1	1	104/83	101/118
2	1	102/34	99/96
3	1	106/136	99/84
4	1	100/116	98/107
5	1	103/152	100/82
6	2	105/147	102/145
7	2	100/143	102/147
8	2	105/163	103/146
9	2	105/122	102/144
10	2	103/64	101/140
11	2	98/158	103/144
12	2	98/123	102/146
13	2	100/44	102/146
14	3	102/132	102/117
15	3	99/46	102/104
16	3	101/18	102/122
17	4	104/93	98/118
18	4	104/82	100/100
19	4	104/104	99/116
20	4	101/89	99/116
21	4	104/107	99/113
22	4	104/118	98/115
23	5	104/96	100/124
24	5	104/86	96/94
25	5	104/109	99/92
26	5	101/85	103/108
27	5	104/122	100/98
28	5	101/111	104/99
29	6	106/96	103/151
30	6	105/109	105/129
31	6	101/106	98/84
32	7	103/93	103/133
33	7	103/73	102/116
34	7	101/66	102/128
35	7	101/118	102/111
36	7	104/118	102/113
37	7	104/130	102/126
38	7	103/54	103/153
39	8	103/93	98/136
40	8	103/76	108/113
41	8	103/61	98/134
42	8	101/120	99/125
43	8	103/124	100/128
44	9	101/85	101/153
45	9	99/111	103/136
46	9	101/129	105/142
47	10	110/28	102/141
48	10	106/85	100/140
49	10	107/76	101/126
50	10	108/56	100/134

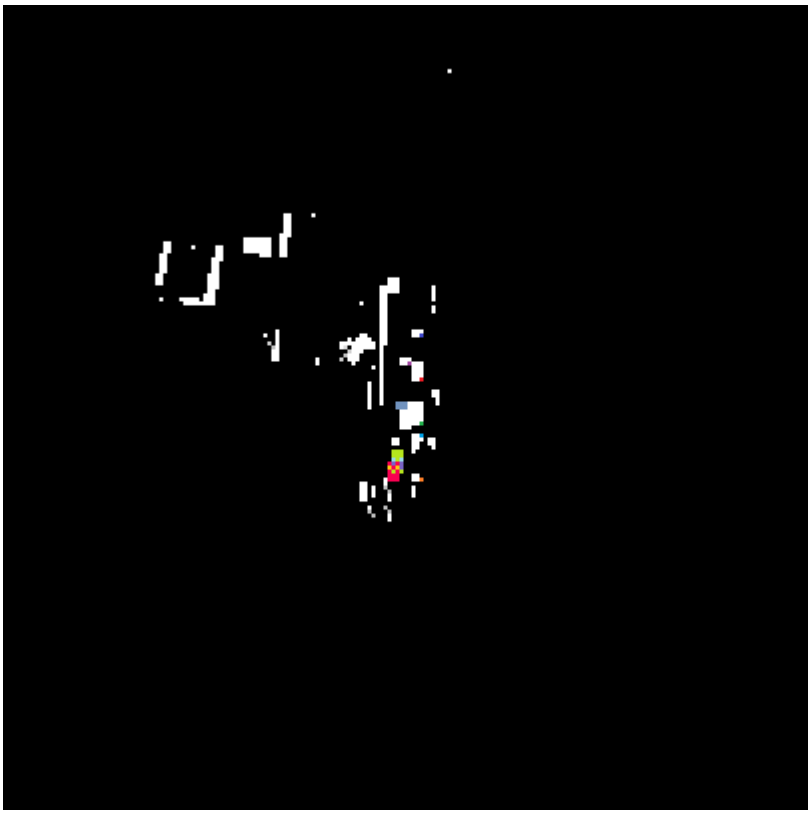

ABBILDUNG 4.5: Graphische Darstellung von Trainingsset 2 zum Zeitpunkt 4. Die vollständig farbigen Autos kennzeichnen die Positionen aus der Berechnung des neuronalen Netzes und die einzelnen farbigen Pixel in den weißen Autos die exakte Position des Labels. Verschiedenfarbig karierte Stellen zeigen Überschneidungen bei der Positionierung der Autos.

Beim Betrachten der Daten in Tabelle 4.2 ist auffällig, dass häufiger größere Abstände zwischen den erwarteten und den tatsächlichen Koordinaten liegt. Vor allem entlang der Y-Koordinate wurden in manchen Fällen Autos von dem oberen Bereich des Bildes in den unteren versetzt, was in der Praxis recht untauglich ist, da es bedeutet, dass das Auto sich durch eine Gruppe von anderen Autos hindurchzwängen müsste, um sich vor ihnen anstelle hinter ihnen einordnen zu können. Dies lässt bereits auf ein schlechter trainiertes neuronales Netz als im ersten Testdurchlauf schließen. Außerdem werden für viele Zeitpunkte der Fahrt die meisten oder sogar alle Autos in etwa der gleichen Position untergebracht. Die wahrscheinlichste Ursache dafür ist ein zu kleines Netz beziehungsweise zu wenig Neuronen um das Verhalten ausreichend detailliert genug modellieren zu können. Bei einer genaueren Betrachtung der graphischen Darstellung der Daten lassen sich zudem weitere Probleme erkennen. So wurden zu Zeitpunkt vier wie in Abbildung 4.5 zu erkennen ist die meisten Autos an dieselbe Stelle positioniert, unabhängig davon wo ihre ursprüngliche Position im Verkehr lag. Doch außerdem wurde das blaue Auto in die Position eines Hindernisses gesetzt, was die wichtigste Regel bei der Ermittlung einer Position verletzt. Vergleichbares lässt sich auch in Abbildung 4.6 erkennen, wo alle berechneten Positionen zwischen zwei der unteren Autos gelegt werden und dabei teilweise das auf dem Bild linke Auto überschneiden. Weiterhin ist zu sehen, dass neben dem eigenen autonomen Fahrzeug in der Bildmitte noch ein weiteres nicht

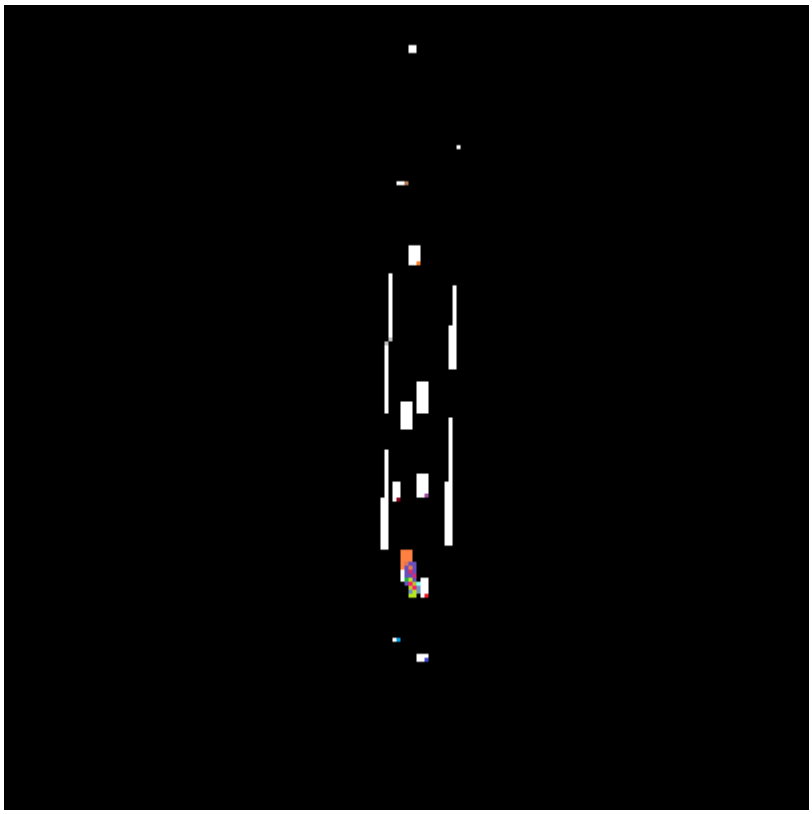

ABBILDUNG 4.6: Graphische Darstellung von Trainingsset 2 zum Zeitpunkt 2. Die vollständig farbigen Autos kennzeichnen die Positionen aus der Berechnung des neuronalen Netzes und die einzelnen farbigen Pixel in den weißen Autos die exakte Position des Labels. Verschiedenfarbig karierte Stellen zeigen Überschneidungen bei der Positionierung der Autos.

zum Lernen verwendet wurde. Dies ist ein möglicher Indikator dafür, dass die Restriktion der relativen Geschwindigkeit beim Erstellen der Trainingsbilder zu streng ist und potenzielle Trainingsdaten verloren gehen.

Eine weitere Auffälligkeit zeigt sich in Abbildung 4.7. Hier ist zu sehen, dass das neuronale Netz größere Probleme hat, die Fahrzeuge korrekt in eine Spur einzuordnen. So konnte für dieses Trainingsbild kein einziges Auto erkennbar in eine korrekte Spur eingeordnet werden. Jedoch zeigt Abbildung 4.8 auch, dass sich für ein vergleichsweise einfaches Trainingsbild mit gut erkennbaren Spuren noch sinnvolle Ergebnisse erhalten lassen, auch wenn selbst hier unnötig erscheinende Verschiebungen von den voran fahrenden Autos stattfinden. Davon abgesehen ist für dieses Trainingsbild jedoch jedes Auto kollisionsfrei und in Relation zu den anderen Autos betrachtet in eine Spur eingeordnet worden. Als Erkenntnisse aus dem zweiten Testdurchlauf lassen sich dabei festhalten, dass das neuronale Netz den Anforderungen der Trainingsdaten nicht gerecht werden konnte, um eine gute Modellierung zu erzielen. Während für einfachere Trainingsbilder noch annehmbare Ergebnisse berechnet wurden, sind für die komplexeren Trainingsbilder mit vielen Hindernissen, ohne sehr deutlich erkennbare Spuren, Positionen ermittelt worden, welche im realen Straßenverkehr zu Unfällen geführt hätten. Somit wäre für die Berechnung

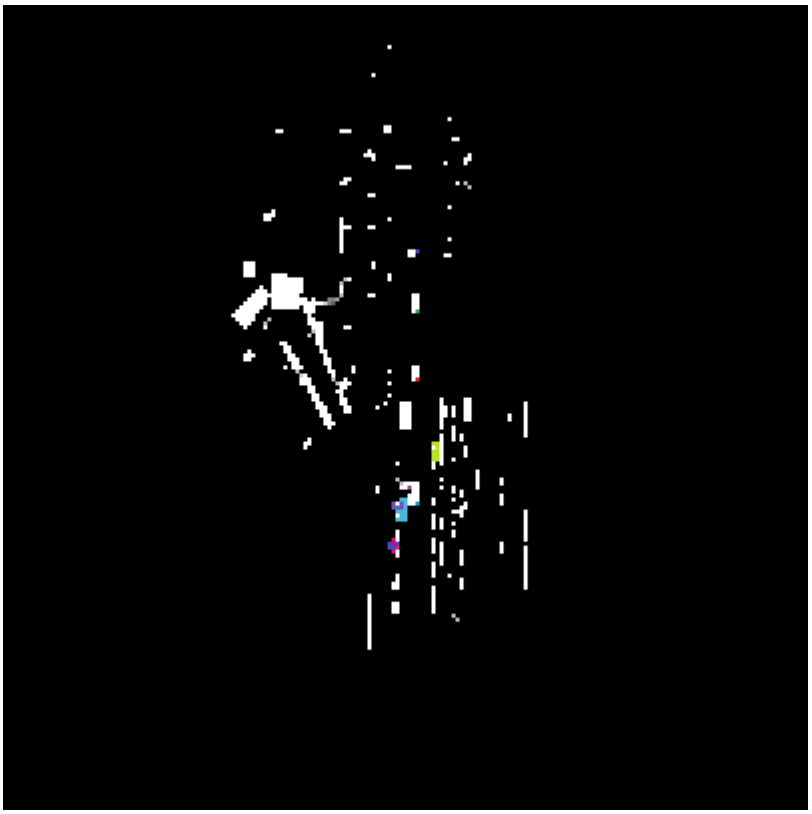

ABBILDUNG 4.7: Graphische Darstellung von Trainingsset 2 zum Zeitpunkt 8. Die vollständig farbigen Autos kennzeichnen die Positionen aus der Berechnung des neuronalen Netzes und die einzelnen farbigen Pixel in den weißen Autos die exakte Position des Labels. Verschiedenfarbig karierte Stellen zeigen Überschneidungen bei der Positionierung der Autos.

besserer Ergebnisse ein komplexeres neuronales Netz notwendig, welches jedoch aufgrund der Größe des Eingabevektors lange Berechnungszeiten mit sich bringen würde und somit durch den zeitlichen Rahmen dieser Arbeit nicht mehr umsetzbar war.

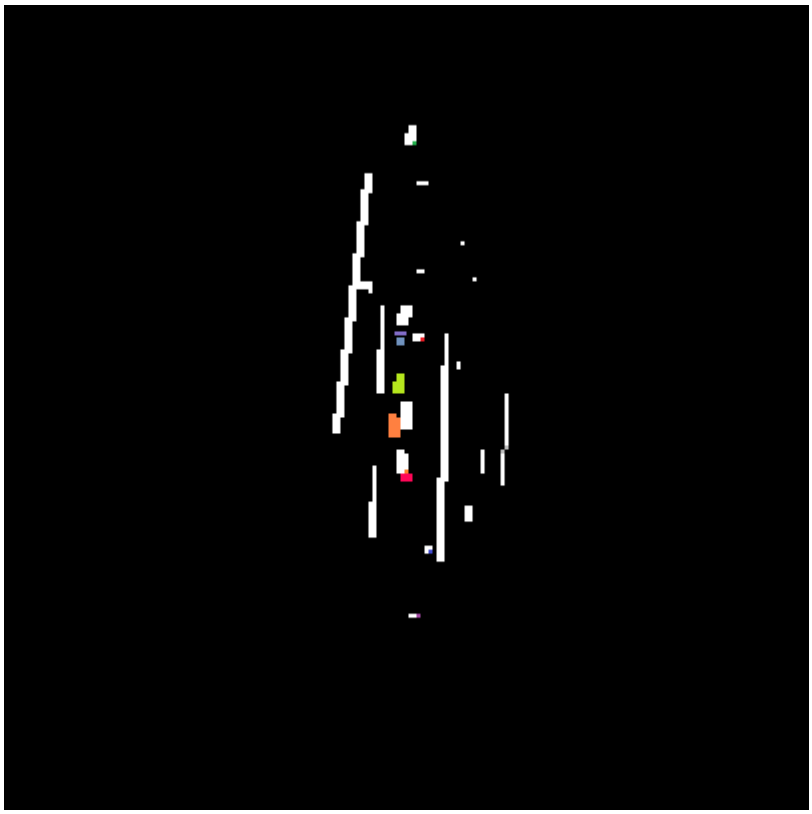

ABBILDUNG 4.8: Graphische Darstellung von Trainingsset 2 zum Zeitpunkt 1. Die vollständig farbigen Autos kennzeichnen die Positionen aus der Berechnung des neuronalen Netzes und die einzelnen farbigen Pixel in den weißen Autos die exakte Position des Labels.

4.2.1 Fazit und Ausblick

In Betrachtung der gesamten Trainingsergebnisse konnte gezeigt werden, dass es möglich ist, mit Hilfe von neuronalen Netzen, Positionen zum Einordnen von Autos in den Straßenverkehr zu bestimmen. Besonders für kleine Trainingsmengen oder Trainingsbilder mit wenig Hindernissen wurden dabei Einordnungen in Fahrspuren und Ausrichtungen an anderen Autos vorgenommen. Jedoch war auch zu Erkennen, dass die Rahmenbedingen des neuronalen Netzes für größere Mengen von Trainingsdaten nicht ausreichend waren. So ist ein einfaches neuronales Netz für die Anforderungen der Trainingsdaten und der Modellierung nicht ausreichend. Stattdessen ist die Verwendung von weiteren Optimierungsmöglichkeiten wie neuronalen Netzen mit Konvolution (convolutional neural networks oder kurz CNN) oder von Frameworks für eine effizientere Formatierung der Daten wie Tensorflow von Google erforderlich um zufriedenstellende Modellierungen für große Trainingsmengen erhalten zu können.

Auch ließen sich bei der Auswertung der Daten Verbesserungsmöglichkeiten für die Trainingsdaten erkennen. So führte es beispielsweise zu Problemen, wenn ein Auto nicht mit allen Punkten erfasst wurde und dies in dem Trainingsbild in einer untypischen Form resultierte (siehe Abbildung 4.4). Ein einfacher Ansatz wäre hierbei, jedes Polygon für ein Auto immer auf eine Rechteckform aufzufüllen, da dies der normalen Form von Autos entspricht und dadurch zu keinen überschüssigen

Pixeln für das Auto führen kann. Ebenfalls ließ sich erkennen, dass eine einfache Ausgrenzung von Autos in Abhängigkeit der relativen Geschwindigkeit unter Umständen nicht ausreichend ist. So kann beispielsweise in der Stadt beim Stehen an einer Ampel, ein auf die Gegenspur abbiegendes Auto mit schätzungsweise 20-30 km/h vorbeifahren. Damit zu solch einer Momentaufnahme dieses Auto nicht als ein der Fahrtrichtung zugehöriges Auto verwendet wird, muss die Schwelle für die relative Geschwindigkeit entsprechend niedrig gesetzt werden. Wenn nun im Gegensatz dazu jedoch sich das autonome Auto auf der Autobahn befindet, sind auch Geschwindigkeitsunterschiede von 40km/h oder mehr zwischen Autos auf der selben Fahrbahn keine Seltenheit. Unter Betrachtung dieser Umstände erscheint daher eine Verwendung von absoluten Geschwindigkeiten als eine bessere Lösung. Insbesondere würde dies die Möglichkeit eröffnen, die Trainingsdaten selbst um die Informationen über die Geschwindigkeiten der Fahrzeuge zu erweitern. Dies ist in sofern relevant, da die absolute Geschwindigkeit von Autos bei der Einordnung in den Verkehr eine Rolle spielt. Das beste Beispiel ist hierbei der Mindestabstand zum voran fahrenden Auto.

Eines der häufigsten auftauchenden Probleme bei der Auswertung der Trainingsdurchläufe war das Problem des neuronalen Netzes Fahrspuren zu erkennen. So waren oft die einzigen Hinweise im Trainingsbild die Position der anderen Fahrzeuge. Wenn sich nun beispielsweise auf einer Fahrspur kein Auto befand, weil es zum Beispiel für dieses Trainingsbild weggelassen wurde (siehe Abbildung 4.4), gab es oft Probleme bei der richtigen Einordnung des Autos in eine Fahrspur, oder es wurde stattdessen in eine andere Spur mit vorhandenen Autos eingeordnet. An dieser Stelle bieten die Laserscandaten nur wenige Möglichkeiten zur Verbesserung, aber wenn von einer anderen Quelle Daten über die Fahrspuren hinzugezogen werden könnten und diese in die Trainingsbilder integriert würden, wäre eine deutliche Verbesserung der genauen Positionierung in Fahrspuren vom neuronalen Netz zu erwarten.

Abschließend sei angemerkt, dass der Ansatz dieser Arbeit darauf basiert, dass die optimale Positionierung im Straßenverkehr eine dem menschlichen Autofahrer möglichst Ähnliche ist. So wäre es zwar auch möglich durch mathematische Berechnungen eine optimal Platz ausnutzende Position mit exakt vorgeschriebenen Abständen zu allen Hindernissen zu finden. Jedoch entspricht dies nicht dem Verhalten von normalen Autofahrern. Wenn somit im Straßenverkehr nahezu ausschließlich menschliche Autofahrer unterwegs sind und dazwischen ein autonomes Fahrzeug mit dem Ziel der mathematisch optimalen Positionierung fahren würde, ist anzunehmen, dass es auf diese Weise eine größere Störung des Verkehrsflusses darstellen würde, als wenn es sich auf eine imitiert natürliche Verhaltensweise in den Verkehr einordnen würde. Der Grund dafür ist, dass die normalen Autofahrer nicht auf Basis solcher mathematischer Berechnungen handeln, sondern auf elementaren Verhaltensregeln basierend, wie bei Individuen eines Schwarms und somit unter Umständen eine ganz andere Vorstellung von den notwendigen Abständen haben. Sollte jedoch die Entwicklung dazu führen, dass die Zahl der autonomen Fahrzeuge im Straßenverkehr stark zunimmt oder diese sogar vorrangig werden, würde dieser Ansatz an Grundlage verlieren und eine mathematische Positionsberechnung zu vergleichsweise besseren Ergebnissen führen.

Anhang A

Verwendeter Quellcode

A.1 Hinzugefügte Codeabschnitte

LISTING A.1: CSV Export der Positionskoordinaten der Autos zur Verwendung als Label

```
std::ofstream mOutFile;
mOutFile.open(mPath);
mOutFile << "input, pos_x, pos_y \n";
for (int i=0; i<cars.size(); ++i) {
...
mOutFile << cars[i][0] << "-" <<i<<  "," << corners[0]<< ","
    << "\n";
\}
```

LISTING A.2: Import der Trainingsdaten und Transformation in Ein- und Ausgabevektoren für das neuronale Netz

```
//loading the images
void inputRead(float (&input) [PATTERN_COUNT][PATTERN_SIZE],
   float (&output) [PATTERN_COUNT][NETWORK_OUTPUT], string
   dirName){
    cv::Mat1f img[2];
    DIR *dir;
    dir = opendir(dirName.c_str());
    struct dirent *ent;
    int i =0;
    if (dir != NULL) {
        while ((ent = readdir (dir)) != NULL) {
            string imgPath(dirName + ent->d_name);
            if ((std::string::npos == imgPath.find("label"))
               && (std::string::npos != imgPath.find("png"))
                )
            {
                img[0] = imread(imgPath, cv::IMREAD_GRAYSCALE
                    );
                img[1] = imread((imgPath.substr(0,imgPath.
                    size()-5) + "label.png"), cv::
                    IMREAD_GRAYSCALE);
                img[0].convertTo(img[0],CV_32F);
```

```
                img[1].convertTo(img[1],CV_32F);
                img[0].setTo(0, img[0] !=255);
                img[1].setTo(0, img[1] !=255);
                img[0].setTo(1, img[0] ==255);
                img[1].setTo(1, img[1] ==255);
                std::vector<float> array1((float*)img[0].data
                    , (float*)img[0].data + img[0].rows * img
                    [0].cols);
                std::vector<float> array2((float*)img[0].data
                    , (float*)img[0].data + img[0].rows * img
                    [0].cols);
                std::copy(array1.begin(), array1.end(), input
                    [i]);
                cout << i << imgPath<< endl;

                string vname, x, y, rest;
                ifstream csvread;
                csvread.open(dirName + "labels.csv", ios::in)
                    ;
                if(!csvread.is_open()) cerr << "Fehler beim
                    Lesen!" << endl;
                else{
                     while(!csvread.eof()){
                         getline(csvread, vname, ',');
                         getline(csvread, x, ',');
                         getline(csvread, y, ']');
                         getline(csvread, rest, '\n');
                         if (imgPath.find(vname) !=std::string
                             ::npos ){
                             array2[0]=stof(x.substr(1))/201;
                             array2[1]=stof(y.substr(1))/201;
                             std::copy(array2.begin(), array2.
                                 end(), output[i]);
                             cout << array2[0]*201 << array2
                                 [1]*201<< endl;
                         }
                     }
                csvread.close();
                }

                i++;
            }
        }
        closedir (dir);
    } else {
        cout<<"not present"<<endl;
    }

}
```

Literatur

[1] AutoNOMOS History
http://autonomos-labs.com/history/

[2] AutoNOMOS MadeInGermany
http://autonomos-labs.com/vehicles/made-in-germany/

[3] AutoNOMOS e-Instein
http://autonomos-labs.com/vehicles/e-instein/

[4] AutoNOMOS FAQ
http://autonomos-labs.com/faq/

[5] ROS History
http://www.ros.org/history/

[6] ROS Wiki Concepts
http://wiki.ros.org/ROS/Concepts

[7] Simon Garnier, Jacques Gautrais, Guy Theraulaz: *The biological principles of swarm intelligence*. Swarm Intell (2007) 1: 3–31.

[8] James Kennedy, Russell C. Eberhart, Yuhui Shi: *Swarm Intelligence*. Swarm Intelligence, Morgan Kaufmann, 2001.

[9] Rojas, Raul: *Neural networks: a systematic introduction* Springer Science & Business Media, 1996

[10] Kornfeld, Nils: *Optimierung eines neuronalen Netzes zur Objekterkennung unter Verwendung evolutionärer Algorithmen* , 2017

[11] Rosenblatt, F.: *The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain.*, Psychological Review (1958), S. 65-386

[12] Rios, Daniel:
http://www.learnartificialneuralnetworks.com/neural-network-software/backpropagation-source-code/