



Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften  
Department für Informatik

Bachelorstudiengang Informatik

Bachelorarbeit

# **Simulation selbstfahrender Autos mithilfe von künstlichen neuronalen Netzen und evolutionären Algorithmen**

vorgelegt von

**Eike Stein**

Gutachter

**Prof. Dr. H.-J. Appelrath  
Cornelius Ludmann**

Oldenburg, 30. June 2016

---

---

# INHALT

<b>1</b>	<b>Einleitung.....</b>	<b>1</b>
1.1	Motivation.....	1
1.2	Ziele der Arbeit.....	1
1.3	Aufbau der Arbeit.....	1
<b>2</b>	<b>Grundlagen.....</b>	<b>2</b>
2.1	Autonomes Fahren.....	2
2.2	Evolutionäre Algorithmen.....	2
2.3	Künstliche neuronale Netze.....	3
2.4	Simulationsumgebung.....	6
<b>3</b>	<b>Verwandte Arbeiten.....</b>	<b>8</b>
<b>4</b>	<b>Anforderungsdefinition.....</b>	<b>9</b>
4.1	Die Simulation.....	9
4.2	An die Ergebnisse.....	9
<b>5</b>	<b>Entwurf.....</b>	<b>10</b>
5.1	Simulation.....	10
5.1.1	Simulationsablauf.....	10
5.1.2	Streckendaten.....	11
5.1.3	Physik.....	12
5.1.4	Visualisierung.....	14
5.2	Künstliches Neuronales Netz.....	14
<b>6</b>	<b>Implementierung.....</b>	<b>15</b>
6.1	Simulation.....	15
6.2	Künstliches Neuronales Netz.....	15
<b>7</b>	<b>Evaluation.....</b>	<b>16</b>
7.1	Fehlerfrei zurückgelegte Strecke.....	16
7.2	Geschwindigkeit.....	16
7.3	Fahrverhalten.....	16
	<b>Literatur.....</b>	<b>17</b>
	<b>Abbildungen.....</b>	<b>18</b>

---

---

Abkürzungen.....	19
Erklärung.....	20

# 1 Einleitung

Im Jahr 2015 starben alleine in Deutschland 3475 Menschen durch Verkehrsunfälle [CITATION Sta16 \l 1031 ]. Bestehende Sicherheitssysteme beschränken sich in der Regel auf die technische Unterstützung des Fahrers. Dazu zählen Antiblockiersysteme (ABS), Elektronisches Stabilitätsprogramme (ESP), Brake Assist Systems (BAS) und je nach Ausstattungsgrad noch weitere. Dem gegenüber begann sich in den letzten Jahren ein Zweig der Informatik zu entwickeln, mit dem Ziel, die Steuerung von Autos vollständig durch Computer zu realisieren. Zu den führenden Forschungseinrichtungen in diesem Bereich zählen auch große Unternehmen wie Google und Tesla. Für eine erfolgreiche Umsetzung autonomer Autos, müssen eine Reihe von Themen, wie Sensorverarbeitung, Bildverarbeitung und Psychologie, behandelt werden. Die besondere Herausforderung bei autonomen Autos, ist das Zusammenspiel vieler verschiedener Teilbereiche, so zu koordinieren, dass ein sicheres und zuverlässiges Fahren ermöglicht wird.

Aufgrund der verschiedenen Umwelteinflüsse ist es in der Regel nicht möglich auf alle erdenkbaren Situationen eine entsprechende Reaktion fest einzuprogrammieren, sodass Algorithmen zum Einsatz kommen müssen, die auch auf neue Gegebenheiten angemessen reagieren können. Einen solchen Ansatz verfolgen künstliche neuronale Netze. Inspiriert von neuronalen Verbindungen im Gehirn, versuchen diese gewünschtes Verhalten zu erlernen. In dieser Arbeit soll untersucht werden, inwieweit es möglich ist, mithilfe von künstlichen neuronalen Netzen Sensordaten eines simulierten Autos zu verarbeiten und die Steuerung dessen zu übernehmen. Konkret soll eine möglichst schnelle, sichere und *menschenähnliche* Fahrweise erreicht werden.

Zu Beginn der Arbeit soll in die zugrundeliegenden Technologien eingeführt werden, sowie einen ersten Überblick über die Simulationsumgebung gegeben werden. Anschließend werden eine Reihe von verwandten Arbeiten aufgelistet, die alternative Ansätze untersuchen oder ausführlicher Teilthemen dieser Arbeit behandeln. Es folgt die Anforderungsdefinition zum einen an die implementierte Simulation und zum anderen an die, zu erzielenden Ergebnisse. Im darauffolgenden Kapitel wird der Entwurf der Simulation und ihrer Komponenten sowie des künstlichen neuronalen Netzes vorgestellt. Danach wird die Implementierung erläutert. Im Anschluss wird eine Ergebnisevaluation durchgeführt, die das Fahrverhalten auf fehlerfrei zurückgelegte Strecke, Geschwindigkeit und Fahrverhalten hin untersucht. Abschließend folgt Fazit und Ausblick. Außerdem wird auf Grenzen dieser Arbeit hingewiesen.

## 2 Grundlagen

### 2.1 Autonomes Fahren

Unter autonomem Fahren bezeichnet man grundsätzlich Autos, Busse, Lastwagen oder andere Verkehrsteilnehmer, die teilweise oder vollständig durch Computer gesteuert werden. Der erste ernstzunehmende Beitrag wurde 1977 von *Tsukuba Mechanical Engineering Laboratory* in Japan geleistet. Damals konnte ein Auto weißen Straßenmarkierungen auf einem abgesperrten Testgelände folgen[ CITATION Ale07 \l 1031 ]. Mittlerweile beschäftigen sich vor allem große Unternehmen wie Google und Tesla mit der Entwicklung[CITATION Aar15 \l 1031 ] [CITATION www16 \l 1031 ]. Damit autonome Fahrzeuge in der Lage sind, sich in ihrer Umgebung zurecht zu finden, kommen eine Reihe von Sensoren zum Einsatz: Kameras, Ultrasound, GPS, Laser und einige weitere [ CITATION Ale07 \l 1031 ]. Die Aufgabe der Software ist es, die Daten, die die einzelnen Sensoren liefern zu verarbeiten und anschließend das Fahrzeug entsprechend zu kontrollieren. Es gibt eine Reihe verschiedener Lösungsstrategien die genutzt werden können. Sogenannte künstliche neuronale Netze ist einer davon[ CITATION Vij15 \l 1031 ].

### 2.2 Evolutionäre Algorithmen

Evolutionäre Algorithmen kommen häufig dort zum Einsatz, wo es zwar möglich ist eine potentielle Lösung für ein Problem zu bewerten, es aber sehr schwer ist eine solche Lösung zu konstruieren. Die Idee dabei orientiert sich an der Evolutionstheorie der natürlichen Selektion nach Charles Darwin, wo die besten Individuen überleben und Nachkommen produzieren, die generell etwas besser sind als die Generationen vor ihnen. So werden nach und nach nur solche Individuen existieren, die am besten in ihrer Umgebung zurechtkommen. Wie auch bei den künstlichen neuronalen Netzen ist es nicht möglich die volle Komplexität in ein Computermodell zu übertragen. Es wird vielmehr die grundsätzliche Idee genutzt und angewendet. Die einzelnen Individuen bestehen meist nur aus einem Array an Zahlen. Diese Zahlen symbolisieren Merkmalsausprägungen und sind vergleichbar mit den Genen in der DNS. Ein Evolutionärer Algorithmus läuft in der Regel wie folgt ab:



1. Zunächst wird eine, meist zufällige, Ausgangspopulation generiert. In den meisten Fällen sind die Individuen nicht als Lösungskandidaten einsetzbar. Es gibt allerdings Variationen, bei denen Wissen über den Lösungsraum in die Generierung mit einfließt, was die Qualität der ersten Generation an Lösungen verbessern kann.

2. Nun werden alle Individuen anhand einer sogenannten Fitnessfunktion bewertet. Beispielsweise könnte das Problem sein ein Polynom 2. Grades zu finden, welche eine Nullstelle an der Stelle 2 hat. Dann wären die Gene/Merkmal ausprägungen der Individuen als Vorfaktoren aufzufassen. So ergibt sich eine mögliche

$$h(\text{Individuum}) = \frac{\text{Bewertungsfunktion}}{|a_{22}x^2 + a_{21}x + a_{20}|}$$

3. Anschließend werden anhand verschiedener Auswahlverfahren Individuen anhand ihrer Bewertung selektiert. Meist wird die Auswahl zufällig gewichtet getroffen.

4. Die in Schritt 3 ausgewählten Lösungskandidaten werden dann paarweise (in einigen Fällen auch tripel- oder quadrupelweise) miteinander kombiniert. Dies geschieht indem zum Teil die Gene des einen, dann die Gene des anderen ausgewählt und aneinandergefügt werden. So würde  $[a, b, c]$  und  $[x, y, z]$

Beispielsweise zu  $[a, y, z]$  oder  $[x, y, c]$  werden. Welche Gene von welchem Individuum genommen werden wird i.d.R. zufällig entschieden. Die resultierenden Individuen können als *Kinder* verstanden werden.


5. Damit der erreichbare Lösungsraum nicht ausschließlich von den Kombinationsmöglichkeiten der Ausgangsindividuen abhängt, werden die *Kinder* nun mit einer gewissen Wahrscheinlichkeit mutiert. Dabei ändert sich meistens ein Gen um einen zufälligen Wert. So wird versucht den gesamten Lösungsraum erreichbar zu machen.
6. Die *Kinder* werden jetzt anhand der Fitnessfunktion bewertet und gespeichert. Es werden solange Kinder erzeugt bis eine festgelegte Anzahl erreicht wurde.
7. Aus allen erzeugten *Kindern* und *Eltern* werden jetzt wieder mit einem (häufig stochastischen) Auswahlverfahren diejenigen Individuen ausgewählt, die in die nächste Generation übernommen werden sollen. Es wird wieder mit Schritt 3 fortgefahren und der Ablauf wiederholt sich.

Ein Evolutionärer Algorithmus läuft prinzipiell unbegrenzt lange, allerdings gibt es eine Reihe möglicher Abbruchkriterien, die entscheiden wann eine weitere Ausführung keinen Sinn mehr macht oder zu *teuer* wird. *Teuer* in dem Sinne, als dass angenommen wird, dass die Ausführung Ressourcen wie Zeit oder Geld kostet. Ein mögliches Abbruchkriterium wäre das eine bestimmte Anzahl an Generationen durchlaufen wurden, oder dass über längere Zeit kein besseres Individuum erzeugt werden konnte. Die gewählten Abbruchkriterien sind domainspezifisch.

Ein Problem der Evolutionären Algorithmen ist die Parameterbestimmung. Es müssen eine Reihe von Werten im Voraus festgelegt werden, die jeweils nicht trivial von dem Lösungsraum, der Größe der Population und anderen Faktoren abhängen können. Beispielsweise ist es nicht möglich eine einheitliche optimale Mutationswahrscheinlichkeit zu empfehlen. In vielen Fällen bietet es sich auch an die Werte im Laufe des Algorithmus anzupassen was die Festlegung weiter erschwert. Es bleibt also festzuhalten, dass EAs zwar flexibel in der Anwendung sind, jedoch nicht einfach ohne Anpassung zu jeder Problemstellung eine Lösung finden.

## 2.3 Künstliche neuronale Netze

*“The computer is incredibly fast, accurate, and stupid. Man is unbelievably slow, inaccurate, and brilliant. The marriage of the two is a challenge and opportunity beyond imagination.” – Stuart Walesh*

Künstliche neuronale Netze versuchen die Brücke zu schlagen, zwischen dem Intellekt von Menschen und der Rechengeschwindigkeit von Computern. Diese Netze sind inspiriert von den neuronalen Verbindungen im Gehirn. Es werden jedoch nur die grundsätzlichen Eigenschaften übernommen und viele biologische Facetten ignoriert. Der Aufbau jedes neuronalen Netzes ist grundsätzlich gleich. Es gibt eine Eingabeebene (*input layer*) die einen Eingabevektor akzeptiert. Über diesen Weg werden (Umgebungs-)Daten oder ähnliches  das neuronale Netz übergeben. Das biologische Äquivalent wären zum Beispiel die Augen, die Farb- und Helligkeitsinformationen wahrnehmen und an Neuronen im Gehirn weiterleiten. Die Daten des Eingabevektors werden nun an die nächste Ebene im Netz propagieren, die erste, sogenannte, *hidden layer*. Diese verarbeitet die Daten und leitet sie weiter an die nächste *hidden layer*, bis schließlich die letzte Ebene erreicht wird und die Ergebnisse ausgelesen werden können (die *output layer*). Jedes Element des Eingabevektors wird genau an ein *Neuron* der Eingabeebene geleitet. Jedes dieser Neuronen ist üblicherweise mit jedem Neuron der nächsten Ebene verbunden. Die Ausgabe eines Neurons, also welcher Wert an die nächste Ebene weitergeleitet wird, errechnet sich mithilfe einer Aktivierungsfunktion. Die Funktion hat das Ziel die Ausgabe immer im gleichen Intervall zu halten. So könnte ein Neuron aufgrund der Eingangskonfiguration einen Wert annehmen, der unproportional groß oder klein ist. Mithilfe der Funktion wird der Wert jedoch wieder in das Intervall  $[0,1]$  oder  $[-1,1]$  projiziert. Eine Aktivierungsfunktion die häufig zum Einsatz kommt, ist die sogenannte Sigmoid-Funktion. Sie hat die Form:

$$\frac{1}{1 + e^{-t}}$$

, wobei  $t$  dabei der eigentliche Wert ist. Ihr Verlauf skizziert sich so:

Desweiteren sind die Verbindungen gewichtet. Das bedeutet das jeder Wert der von der Aktivierungsfunktion berechnet wurde mit einer bestimmten Gewichtung mit in den Eingabewert eines Neurons eine Ebene weiter fließt. Der letztendliche Eingabewert ergibt sich aus der Summe aller gewichteten Ausgabewerte der Neuronen der vorherigen Ebene. Somit ergibt sich folgende Formel für ein Neuron mit dem Index  $i$  in Ebene  $k$ :

$$\text{Ausgabe}(\text{Neuron}_{i,k}) = \text{Sigmoid}\left(\sum_{j=0}^n \text{Ausgabe}(\text{Neuron}_{j,k-1}) * \text{Gewicht}_{[j,k-1],[i,k]}\right)$$

So errechnet sich der Ausgabewert jedes Neurons, mit Ausnahme denen der ersten Ebene, die ihren Wert explizit gesetzt bekommen. Wie sich die Anzahl der *hidden layers* festlegt und wie viele Neuronen sich jeweils in ihnen befinden, ist nicht genau definiert und hängt von der Komplexität des Einsatzgebietes ab. Ein einfaches *Und-Gatter* lässt sich beispielsweise mit 2 Eingabeneuronen und einem Ausgabeneuron realisieren und somit gar keine *hidden layers* benötigt werden. Möchte man ein *Exklusiv-Oder-Gatter* nachstellen benötigt man hingegen schon eine *hidden layer*. Intuitiv lässt sich das damit erklären, dass die Eingabedaten in Verbindung zueinander gesetzt werden müssen. Desweiteren werden in der Regel *Bias-Neuronen* implementiert. Diese speziellen Neuronen haben als Ausgabewert immer 1 und ermöglichen so auch Informationen aus einem Eingabevektor zu gewinnen, bei dem alle Werte 0 sind. Ein Beispiel dafür wäre ein *NOR-Gatter*. Die Ausgabe soll unter anderem dann 1 sein, wenn alle Eingangswerte 0 sind. Ohne *Bias-Neuronen* ist es nicht möglich die Verbindungen im Netz so zu gewichten, dass aus einem 0-Eingabevektor ein *nicht-0-Ausgabewert* wird.

$$I_0=0, I_1=0, \text{dann } I_0 * w_0 + I_1 * w_1 = 0$$

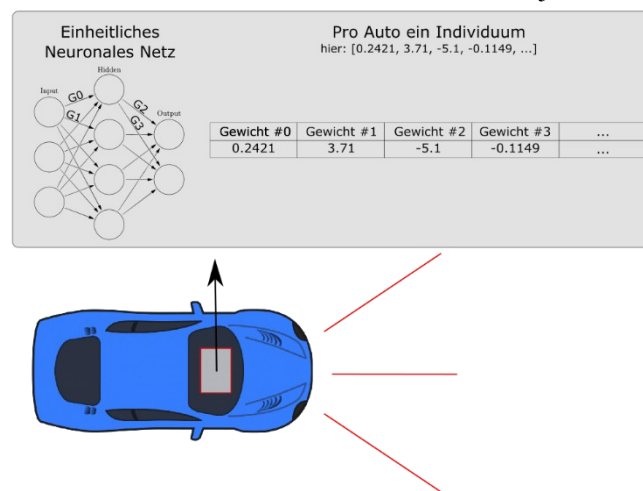
Damit die Ausgabe eines neuronalen Netzwerks überhaupt sinnvoll verwendet werden kann, muss das Netz zunächst trainiert werden. Dazu werden Trainingsdaten erzeugt, an denen das Netz trainiert werden kann. Wurde ein vorher festgelegtes Trainingsziel erreicht, kann das Netz nun an Daten außerhalb des Trainingsdatensatzes getestet werden. So könnte zum Beispiel die Börsendaten der letzten Wochen und Monate als Trainingsdatensatz genutzt werden, da hierfür bekannt ist, wie sich die Aktienkurse tatsächlich verändert haben. Im Trainingsprozess wird jetzt versucht zu erreichen, dass das neuronale Netz die Aktienkurse möglichst präzise vorhersagt. Ist das Training abgeschlossen, kann das Netz für Vorhersagen aktueller Aktienkurse eingesetzt werden. In der Realität gibt es in diesem Prozess einige Hürden, deren Bewältigung nicht ganz einfach ist. In fast allen Fällen bleibt der Aufbau des Netzwerks über den gesamten Trainingszeitraum gleich, nur die einzelnen Gewichte ändern sich. Für die Bestimmung der Gewichte gibt es eine Reihe verschiedener Ansätze, die unterschiedliche Vor- und Nachteile mit sich bringen. Häufig kommt der *Back-Propagation-Algorithmus* zum Einsatz. Die Idee bei diesem Algorithmus ist es, die Ausgabe des Netzwerks mit einer vorher festgelegten Ausgabe zu vergleichen. Je größer der Abstand zur gewünschten Ausgabe je größer der Fehler. Dieser Fehlerwert wird dann von der Ausgabebene durch die Ebenen zurück propagiert und die Gewichte werden dabei korrigiert. Der Vorteil liegt vor allem in der einfachen Implementierung, allerdings ist das Trainieren sehr zeitaufwendig. Ein weiteres Problem ist, dass man zunächst Trainingsdaten benötigt, die jedem gegebenen Eingabevektor des Datensatzes genau ein Ausgabevektor zuordnen. Das ist zwar häufig kein Problem, wie an dem Beispiel mit den Aktienkursen gezeigt, aber nichtsdestotrotz gibt es Situationen in denen die Erzeugung der gewünschten Ausgabedaten nicht ohne weiteres möglich ist. Angenommen das Ziel ist es ein künstliches neuronales Netz als Steuerungseinheit für ein Weltraumfahrzeug zu trainieren, dass später im Falle eines Verbindungsabbruchs selbstständig die Umgebung auf einem Himmelskörper erkundet. Zwar ist es durchaus denkbar, dass mithilfe von Simulationen ein Datensatz mit verschiedenen Sensorwerten generiert werden kann, allerdings stellt das Festlegen der gewünschten Ausgabewerte ein großes Problem dar. In solchen Fällen können häufig evolutionäre Algorithmen eingesetzt werden. Die einzelnen Individuen stellen dabei jeweils eine Gewichtungskonfiguration dar. Jedes *Gen* entspricht einem Verbindungsgewicht. Die



Bewertungsfunktion ist dabei nicht direkt an die Ausgabe gekoppelt, sondern vielmehr an das Verhalten welches von den Ausgabewerten ausgeht. So könnte bei dem Weltraumfahrzeug die zurückgelegte Strecke ein Faktor für die Bewertung eines Individuums sein; ein Wert der ebenfalls in Simulationen bestimmt werden kann.

## 2.4 Simulationsumgebung

Nachdem in EAs und KNNs eingeführt wurde, soll nun deren Rolle in der Simulation erläutert werden. Die Simulation benutzt GPS-Koordinaten bekannter Rennstrecken um einen virtuellen Rundkurs zu erzeugen. Auf diesem Rundkurs werden dann Autos platziert, die mithilfe von Abstandssensoren ihre Umgebung wahrnehmen können. Die Autos besitzen jeweils ein künstliches neuronales Netz um die Sensorwerte in Geschwindigkeit und Lenkrichtung zu übertragen. Der Aufbau der KNNs unterscheidet sich dabei nicht zwischen den Autos nur die Gewichte der einzelnen Verbindungen. Die Gewichte werden mithilfe von einem EA trainiert. Jedes Individuum in dem EA entspricht einer Gewichtungskombination. Dabei gibt jedes Gen das Gewicht genau einer Verbindung im neuronalen Netz an. Das bedeutet, dass sich die Autos nur in der Gewichtungskonfiguration der Verbindung ihrer neuronalen Netze unterscheiden und somit die Individuen es sind, die das eigentliche Fahrverhalten der Autos bestimmen.



Es wird zunächst eine zufällige Startpopulation generiert. Nun wird für jedes Individuum ein Auto mit der entsprechenden Gewichtungskonfiguration auf einem ausgewählten Rundkurs platziert. Anschließend wird die Simulation gestartet und die neuronalen Netze reagieren auf die Sensordaten und steuern das Auto durch den Rundkurs. Kollidiert das Auto dabei mit der Streckenbegrenzung bricht die Simulation sofort ab und die Bewertung findet statt. Alternativ beendet sich die Simulation auch nach einer festgelegten Zeit. Wie genau die Bewertung stattfindet wird im Kapitel *Simulation* näher beschrieben. Grundsätzlich folgt aber aus großer zurückgelegter Strecke mit hoher Geschwindigkeit eine hohe Bewertung. Nachdem das Fahrverhalten jedes Individuums der Population bewertet wurde, wird die neue Generation

erzeugt und die Simulation startet erneut. Nach einigen Generationen hat sich dann hoffentlich ein erfolgreiches Fahrverhalten entwickelt.

### **3 Verwandte Arbeiten**

## 4 Anforderungsdefinition


### 4.1 Die Simulation

Ziel ist es eine Fahrphysik zu simulieren, die es ermöglicht eine realistische Einschätzung von einem gegebenem Fahrverhalten machen zu können, ohne Rücksicht auf eventuelle Einschränkungen durch die Simulation nehmen zu müssen. Das bedeutet, dass die Simulation nicht vollständig die realen Gegebenheiten widerspiegeln muss (Gangschaltung, Reifentemperatur, Fahrtwind oder ähnliches), allerdings eine Übertragung des Fahrverhaltens auf reale Autos denkbar wäre. Außerdem ist es notwendig den Fortschritt auf einer Strecke messen zu können, sodass die Bewertung einzelner Fahrer, die zurückgelegte Strecke berücksichtigen kann. Des Weiteren, muss eine Kollisionserkennung zwischen Auto und Streckenbegrenzung erfolgen, da das einem unakzeptablen Fahrverhalten entspricht und entsprechend mit in die Evaluation fließen muss. Die Messwerte der Simulation müssen realitätsnah sein. Insbesondere die Geschwindigkeit, Entfernung und Zeit sollten mit echten Autos vergleichbar sein.

### 4.2 An die Ergebnisse

## 5 Entwurf

### 5.1 Simulation

Als erstes werden Streckendaten, die vorher hinterlegt wurden, eingelesen und analysiert. Anschließend bekommt der Benutzer die Möglichkeit eine Strecke auszuwählen, sowie zu entscheiden wie die Startpopulation aufgebaut sein soll.  kann auswählen zwischen zufällig generierten oder vorher gespeicherten Individuen. So kann die Simulation nach einer Unterbrechung fortgesetzt werden. Des Weiteren soll es möglich sein die Einstellungen der Simulation anzupassen. Dazu zählen unter anderem maximale Geschwindigkeit und Beschleunigung der Fahrzeuge, aber auch Mutationswahrscheinlichkeiten, Anzahl der *hidden layers* und, bis auf wenige Ausnahmen, alle anderen Werte die an irgendeiner Stelle in der Simulation benötigt werden. Nachdem die Konfiguration abgeschlossen ist, kann die Simulation gestartet werden. Es kann entweder ein Einzelschritt der Simulation ausgeführt oder aber fortlaufend neue Generationen errechnet werden. Es ist außerdem möglich Individuen auszuwählen, sich das neuronale Netzwerk anzeigen zu lassen und gegebenenfalls zu speichern. Ferner kann das ausgewählte Individuum visualisiert werden, indem sein Verhalten auf der Rennstrecke dargestellt wird. Dazu wird die Simulation in Echtzeit ausgeführt und der aktuelle Zustand aller Objekte gerendert.

#### 5.1.1 Simulationsablauf

Der Simulationsablauf läuft zyklisch ab. Je nachdem wie viele Individuen es zu simulieren gilt, werden *Teilsimulationen* initialisiert, sodass jedes Individuum genau einer Teilsimulation zugeordnet wird. Jede dieser Teilsimulation laufen parallel ab, sodass der Computer auf dem die Simulation läuft möglichst voll ausgelastet wird. Die Koordination übernimmt dabei ein *Simulationsmanage*. Dieser startet die Teilsimulationen und wartet anschließend bis alle beendet wurden. Dies kann über zwei Wege passieren: Zum einen kann die Zeit abgelaufen sein. Jedes Individuum hat nur eine begrenzte Zeit zur Verfügung eine möglichst große Strecke zurückzulegen. Zum anderen kann das Fahrzeug mit der Streckenbegrenzung kollidiert sein. Da dies ein nicht akzeptables Fahrverhalten darstellt kommt es zum vorzeitigen Abbruch. Sind alle Teilsimulationen beendet beginnt die Auswertung. Dabei wird jedem Individuum ein Wert zugeordnet. Dieser spiegelt wider, wie weit das Fahrzeug auf dem Rundkurs gekommen ist. Anschließend beginnt die Selektion und Mutation der Individuen, basierend auf ihrer errechneten Bewertung.

Die einzelnen Teilsimulationen laufen ebenfalls zyklisch ab. Zunächst wird die Simulation jedoch initialisiert. Dazu werden die errechneten Werte der Rennstrecke ausgelesen und in entsprechende Datenstrukturen übertragen. So wird jede Kante der beiden Polygone (*innere* und *äußere* Begrenzung) einmalig in die Physikengine übertragen. So wird gewährleistet, dass die Kollisionen korrekt erkannt werden und die Sensoren korrekte Werte liefern. Außerdem wird das Fahrzeug auf der Startposition platziert und alle Kräfte und Bewegungen werden zurückgesetzt. Dies ist wichtig, da die Simulationen wiederverwendet werden können. So muss nicht nach jeder Teilsimulation die Rennstrecke erneut in die Physikengine eingespeist werden,

sondern nur das Fahrzeug angepasst werden. Nachdem die Initialisierung abgeschlossen ist, beginnt die eigentliche Simulation. Mithilfe von Raycasting wird in bestimmten Winkeln von der Front des Fahrzeuges aus die Entfernung zum nächsten Hindernis gemessen. Wie viele und welche Winkel genau, kann beliebig eingestellt werden. Es bietet sich an einen großen Bereich vor dem Auto abzudecken, sodass das künstliche neuronale Netz genug Informationen besitzt, um eine angemessene Entscheidung treffen zu können. Bedacht werden muss jedoch, dass eine höhere Anzahl an Raycasts auch mehr Rechenzeit in Anspruch nehmen und sich so die Simulation insgesamt verlangsamt. Außerdem steigt mit der Anzahl der Sensorwerte die Anzahl der Eingabeneuronen im Netzwerk und so auch die Menge an Verbindungsgewichten, welche wiederum entsprechend trainiert werden müssen. So dauert nicht nur jede einzelne Generation länger, weil die Teilsimulation mehr Zeit benötigen, sondern auch die Zahl der Generationen, bis sich ein brauchbares Fahrverhalten entwickelt hat. Zwar bedeuten mehr Gewichte, auch mehr Möglichkeiten das Fahrverhalten zu verbessern, aber auch eben mehr Freiheitsgrade, die es zu trainieren gilt. Auf der anderen Seite darf eine bestimmte Anzahl an Sensoren auch nicht unterschritten werden, da dann das neuronale Netz nicht einmal theoretisch genug Informationen besitzt, um angemessen agieren zu können. So entsteht ein schmaler Grat zwischen Informationsmenge und Simulationsdauer, dessen Ausloten vor allem durch Ausprobieren erreicht werden kann.

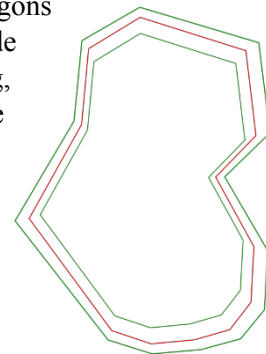
### 5.1.2 Streckendaten

Die Streckendaten stammen von der Webseite GPSies. Dort können Benutzer Strecken erstellen, bearbeiten und veröffentlichen. Einige Nutzer haben die GPS-Koordinaten von bekannten Rennstecken recherchiert und auf die Webseite hochgeladen. Über einen kostenlosen Downloadbereich können die Strecken in einem XML-Format heruntergeladen werden. Die Streckenpunkte liegen allerdings als GPS-Koordinaten vor und müssen für die Verwendung in der Simulation zunächst in ein Format umgewandelt werden, dass auf Metern basiert. Dazu kam zunächst die Webseite <http://www.uwgb.edu/dutchs/usefuldata/ConvertUTMNoOZ.HTM> zum Einsatz, da es jedoch recht mühsam ist, jeden Streckenpunkt manuell umzuwandeln, kommt ein selbstprogrammierter Algorithmus zum Einsatz, der die Konvertierung automatisiert. Die Grundlage dafür lieferte der JavaScript-Quellcode auf der Webseite, allerdings wurden noch Teile abgeändert und optimiert um direkt mit den XML-Dateien zu funktionieren. Was jedoch zusätzlich nach wie vor manuell eingefügt werden muss, ist der Name der jeweiligen Strecke. Dies wird über ein Attribut im Kopf der XML-Datei realisiert. Die XML-Datei des *Antree Circuit* sieht beispielsweise wie folgt aus:

```
<?xml version="1.0" encoding="utf-8" ?>
<track name="Antree Circuit">
  <trkpt lat="53.47596610" lon="-2.95051574" />
  <trkpt lat="53.47973330" lon="-2.94613838" />
  <trkpt lat="53.48023130" lon="-2.94553756" />
  <trkpt lat="53.48038450" lon="-2.94510841" />
  <trkpt lat="53.48049950" lon="-2.94455051" />
  <trkpt lat="53.48053780" lon="-2.94360637" />
  <trkpt lat="53.47935020" lon="-2.93869256" />
  <trkpt lat="53.47912030" lon="-2.93860673" />
  <trkpt lat="53.47568510" lon="-2.94276952" />
  <trkpt lat="53.47539140" lon="-2.94266223" />
  <trkpt lat="53.47527650" lon="-2.94246912" />
  <trkpt lat="53.47521260" lon="-2.94214725" />
  <trkpt lat="53.47532750" lon="-2.93828487" />
  <trkpt lat="53.47545520" lon="-2.93794155" />
  <trkpt lat="53.47563400" lon="-2.93776988" />
  <trkpt lat="53.47617040" lon="-2.93779134" />
  <trkpt lat="53.47683450" lon="-2.93776988" />
  <trkpt lat="53.47760070" lon="-2.93772697" />
  <trkpt lat="53.47785610" lon="-2.93738365" />
  <trkpt lat="53.47800930" lon="-2.93691158" />
  <trkpt lat="53.47807320" lon="-2.93648242" />
  <trkpt lat="53.47642580" lon="-2.92980909" />
  <trkpt lat="53.47615760" lon="-2.92957305" />
  <trkpt lat="53.47588950" lon="-2.92948722" />
  <trkpt lat="53.47557020" lon="-2.92946577" />
  <trkpt lat="53.47530200" lon="-2.92965888" />
  <trkpt lat="53.47499550" lon="-2.93002367" />
  <trkpt lat="53.47459960" lon="-2.93090343" />
  <trkpt lat="53.47442080" lon="-2.93159008" />
  <trkpt lat="53.47389720" lon="-2.94412136" />
  <trkpt lat="53.47383330" lon="-2.94609546" />
  <trkpt lat="53.47382050" lon="-2.94742584" />
  <trkpt lat="53.47353960" lon="-2.94871330" />
  <trkpt lat="53.47345020" lon="-2.94939994" />
  <trkpt lat="53.47336080" lon="-2.95055866" />
  <trkpt lat="53.47352680" lon="-2.95163154" />
  <trkpt lat="53.47373110" lon="-2.95199632" />
  <trkpt lat="53.47389720" lon="-2.95218944" />
  <trkpt lat="53.47420370" lon="-2.95221090" />
  <trkpt lat="53.47451020" lon="-2.95195341" />
  <trkpt lat="53.47499550" lon="-2.95156717" />
  <trkpt lat="53.47526370" lon="-2.95124530" />
  <trkpt lat="53.47586390" lon="-2.95066595" />
</track>
```

Beim Programmstart wird in einem speziellen *Racetrack* Ordner nach dieser Art von Dateien gesucht. Die Koordinaten werden umgerechnet und ein *Racetrack*-Objekt wird erstellt. Da es sich bei den Strecken um Rundkurse handelt, können die errechneten Koordinaten als Eckpunkte eines Polygons aufgefasst werden. Verkleinert man das resultierende Polygon, erhält man die *innere* Begrenzung, vergrößert man es die *äußere* Begrenzung. Diese Begrenzungen dienen als Grenzen die von den Fahrzeugen zu keinem Zeitpunkt überfahren werden dürfen. Die Ursprungskoordinaten, sowie die errechneten Eckpunkte werden in dem

*Racetrack*-Objekt gespeichert und können jetzt für die Simulation eingesetzt werden.



### 5.1.3 Physik

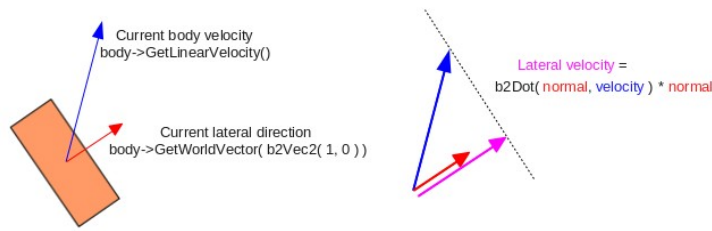
Die physikalische Berechnung übernimmt die *Farseer Physics Engine*. Bei ihr handelt es sich um eine Open-Source 2D Engine, die sich stark an der bekannten *Box2D* Engine orientiert. Sie ist vollständig in C# geschrieben und so ist es sehr einfach möglich sie in das Projekt zu integrieren. Die Handhabung ist sehr einfach. Jeden Simulationsschritt, also jeden *Frame*, wird die *Step*-Funktion aufgerufen. Als Parameter wird die vergangene Zeit seit dem letzten Aufruf

übergeben. Die Engine errechnet dann die neuen Positionen und Rotationen der Objekte, die zuvor zur Engine hinzugefügt wurden. Dabei werden Kollisionen auf- und entsprechende Ereignisse ausgelöst, auf die wiederum reagiert werden kann. Mithilfe von *Joints*, oder Gelenken, können verschiedene Objekte miteinander verbunden werden. Ein *Slider-Joint* beispielsweise sorgt dafür, dass zwei Objekte sich nur entlang einer bestimmten Achse zueinander bewegen können. So ist es möglich komplexe Modelle aus simplen Figuren zu erzeugen. Da vom Benutzer erwartet wird, dass er die benötigten Modelle selbst erstellt, liefert die Engine keine vorgefertigten Fahrzeug-, Flugzeug- oder etwaige andere Modelle. Aus diesem Grund ist es notwendig ein eigenes Fahrzeugmodell zu erstellen. Zunächst wird das Fahrgestell modelliert. Da nur zwei Dimensionen berücksichtigt werden und die Simulation aus die Sicht von oben auf die Strecke durchgeführt wird, muss auch nur der Umriss des Fahrgestells aus dieser Perspektive angegeben werden. Die Physikengine errechnet dann aus den gegebenen Eckpunkten des Fahrgestells-Polygon den Schwerpunkt und das Gewicht, um eine möglichst realistische Simulation zu ermöglichen. Ein mögliches Fahrgestell könnte zum Beispiel die Eckpunkte:

$(-0.45, -1.825), (-0.9, -1.325), (-0.85, 0.275), (-0.25, 2.125),$   
 $(0.25, 2.125), (0.85, 0.275), (0.9, -1.325), (0.45, -1.825)$  haben, wobei der erste Wert den x-Wert und der zweite den y-Wert angibt. Reifen könnten an den Positionen:  
 $(-0.9, 1.5), (0.9, 1.5), (-0.9, -1.8), (0.9, -1.8)$  Das Fahrzeug hätte damit eine Breite von 1,80m und eine Länge von 3,95m. Grob visualisiert stellt es sich wie folgt dar:

Die Reifen werden über sogenannte *Revolute-Joints* mit dem Fahrgestell verbunden. Diese *Joints* sorgen dafür, dass die Reifen in einer konstanten relativen Position zum Chassis gehalten werden. Zusätzlich ermöglichen sie, das Einstellen von minimaler und maximaler Winkeländerung in Relation zum Fahrgestell. So kann gewährleistet werden, dass die vorderen Reifen nur in einem bestimmten Winkel bewegt werden können. Zusätzlich werden die hinteren Reifen parallel zur Fahrtrichtung fixiert. Für die Beschleunigung bietet die *Farseer Physics Engine* die Möglichkeit Kräfte von Objekten der Simulation ausgehen zu lassen. So ist es möglich Vorder-, Heck- oder sogar Allradantrieb umzusetzen. Dabei wird die Motorkraft auf die entsprechenden Reifen verteilt. Wie sich daraufhin der Rest des Fahrzeuges bewegt, wird von der Engine berechnet. Etwas was allerdings nicht ohne zusätzlichen Programmieraufwand berücksichtigt wird ist die Reibung der Reifen. Würde man die Simulation ohne diese starten, würde sich das Auto problemlos orthogonal zur Reifenstellung bewegen können. Das eine Bewegung grundsätzlich nur die die Richtung möglich ist, in die auch die Reifen ausgerichtet sind (abgesehen von Rutschen auf nasser oder vereister Straße), ist eine Eigenschaft von Autos, die der Engine nicht bekannt ist. Somit ist es notwendig die erforderlichen Kräfte und Impulse manuell zu berechnen und auf das Fahrzeug zu übertragen. Da dieses Problem sich nicht nur auf die *Farseer Physics Engine* beschränkt, gibt es eine Reihe von Anleitungen, die mögliche Lösungsansätze erläutern. Eine solche Anleitung findet sich unter dem Link <http://www.iforce2d.net/b2dtut/top-down-car>. Die grundsätzliche Idee ist, die laterale Geschwindigkeit mit gezielten Impulsen entgegenzuwirken, sodass ausschließlich die Geschwindigkeit parallel zu den Reifen übrigbleibt. Der Impuls errechnet sich dann aus der Masse des jeweiligen Reifen multipliziert mit der negativen lateralen Geschwindigkeit. Wendet man diesen Impuls in jedem Simulationsschritt an, wird eine orthogonale Bewegung ausgeschlossen. Die eigentliche Reibung wird allerdings nach wie vor nicht berechnet. So kann





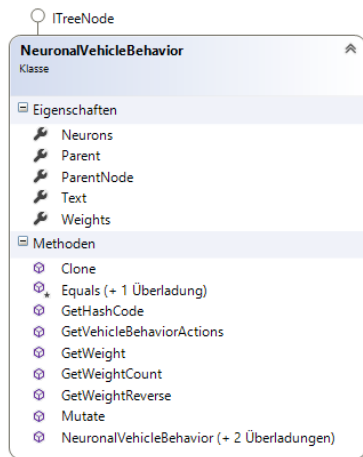
das Fahrzeug, sofern es einmal beschleunigt wurde, ungehindert weiter rollen. In der Realität würde Luftwiderstand und Reibung dafür sorgen, dass das Fahrzeug gebremst wird. Dieses

Verhalten lässt sich Umsetzen, indem eine Kraft berechnet wird, die entgegen der aktuellen Fahrtrichtung wirkt. Für die Realisierung des Lenkens kommen die bereits erwähnten Begrenzungen der *Joints* zum Einsatz. Es ist prinzipiell möglich mit ihnen dem Reifen einen gewissen Spielraum zu geben, allerdings können die Begrenzungen auch so gesetzt werden, dass der Reifen in einen ganz bestimmten Winkel gezwungen wird. Die Angabe erfolgt über einen minimalen und einen maximalen Winkel. Um einen Reifen auf einen Winkel festzulegen wird der minimale und maximale Winkel auf denselben Wert festgelegt. So hat der Reifen keinen Spielraum mehr. Dies kann bei den Vorderreifen genutzt werden, um die Lenkrichtung festzulegen. Dazu werden die beiden Winkel auf den entsprechenden Wert gesetzt. Es bedarf jedoch einiger Einschränkungen. Zum einen darf die Lenkrichtung nicht von einem Extrem zum anderen in zu kurzer Zeit gesetzt werden. Um möglichst realitätsnah zu bleiben, sollte das Umlenken eine gewisse Zeit in Anspruch nehmen. Des Weiteren, muss die maximale Lenkrichtung in Abhängigkeit von der Geschwindigkeit geändert werden. Ansonsten wäre es möglich Kurven mit einer unrealistisch hohen Geschwindigkeit zu fahren. Ein alternativer Ansatz wäre in einem solchen Fall Haftungsverlust der Reifen zu simulieren, allerdings müssten davor eine Reihe zusätzlicher Mechanismen berücksichtigt werden. Zum Beispiel die Gewichtsverlagerung beim Bremsen und Beschleunigen, aber auch das Fahrwerk und die Federung. Aufgrund dieser Komplexität bietet es sich an schlicht die maximale Auslenkung zu verringern und den Verlust an Realismus in Kauf zu nehmen.

#### 5.1.4 Visualisierung

### 5.2 Künstliches Neuronales Netz

Das implementierte künstliche neuronale Netz ist fest mit der Implementierung des Fahrzeugverhaltens verknüpft. So gibt es eine Methode, die als Parameter ein *VehicleBehaviorInput* akzeptiert, welches die Sensordaten und gegebenenfalls die Fahrzeuggeschwindigkeit beinhaltet. Die Daten werden in das Netzwerk an die Eingabeneuronen angelegt, die Ausgabe wird berechnet und in der *VehicleBehaviorActions*-Struktur gespeichert und zurückgegeben. So konvertiert die Methode die Sensordaten in entsprechendes Fahrverhalten ausschließlich auf Basis der Konfiguration des neuronalen Netzwerks.



Die einzelnen Neuronen sind dabei als ein 2-dimensionalen *double*-Array und die Verbindungsgewichte als 1-dimensionalen *double*-Array implementiert. Das *Parent* und *ParentNode* Attribut dienen allein dazu in der Oberfläche anzuzeigen welche Vererbungsstrukturen sich im Laufe der Simulation durch den evolutionären Algorithmus ergeben. Dies wird realisiert, indem bei dem Klonen von Netzwerken gespeichert wird, welches Netzwerk die Grundlage lieferte. So entsteht eine hierarchische Struktur, die visuell dargestellt werden kann.

Wenn die Ausgabe des Netzwerkes berechnet werden soll, werden die Neuronen der ersten Ebene auf die entsprechenden Werte des *VehicleBehaviorInput* gesetzt.

Anschließend wird die erste *hidden layer* berechnet, indem die Ausgabewerte der Neuronen aus der Eingabeebene mit den Gewichten aus dem *Weight*-Array multipliziert und jeweils pro Neuron aufaddiert werden. Nach jedem Zugriff auf den *Weight*-Array wird der Index des Arrays um 1 erhöht. So wird gewährleistet, dass jede Verbindung eindeutig einem Element aus dem *Weight*-Array zugeordnet ist. Ist die letzte Ebene berechnet, werden die Werte an den Ausgabeneuronen in einer *VehicleBehavior*-Struktur gespeichert. Sie besteht im Wesentlichen aus einem Geschwindigkeits- und Richtungsfaktor. Der Geschwindigkeitsfaktor wird mit der maximalen Geschwindigkeit des Fahrzeugs multipliziert und ergibt so, die Zielgeschwindigkeit. Bei dem Richtungsfaktor wird äquivalent verfahren, nur liegt hier Wert zwischen -1 und +1. Wobei -1 einem maximalen Einlenken nach links und +1 einem maximalen Einlenken nach rechts entspricht. Diese Werte werden nicht direkt auf das Fahrzeug übertragen, sondern zunächst von der Physikengine verarbeitet (siehe 5.1.2 *Physik*).

## **6 Implementierung**

### **6.1 Simulation**

### **6.2 Künstliches Neuronales Netz**

## **7 Evaluation**

### **7.1 Fehlerfrei zurückgelegte Strecke**

### **7.2 Geschwindigkeit**

### **7.3 Fahrverhalten**

## Literatur

- [1] „statista.com,“ [Online]. Available: <http://de.statista.com/statistik/daten/studie/185/umfrage/todesfaelle-im-strassenverkehr/>. [Zugriff am 08 05 2016].
- [2] M. K. Alex Forrest, „wpi.edu,“ 1 Mai 2007. [Online]. Available: <http://www.wpi.edu/Pubs/E-project/Available/E-project-043007-205701/unrestricted/IQPOVP06B1.pdf>. [Zugriff am 31 Mai 2016].
- [3] A. M. Kessler, „nytimes.com,“ 19 März 2015. [Online]. Available: [http://www.nytimes.com/2015/03/20/business/elon-musk-says-self-driving-tesla-cars-will-be-in-the-us-by-summer.html?\\_r=0](http://www.nytimes.com/2015/03/20/business/elon-musk-says-self-driving-tesla-cars-will-be-in-the-us-by-summer.html?_r=0). [Zugriff am 31 Mai 2016].
- [4] „google.com,“ [Online]. Available: <https://www.google.com/selfdrivingcar/>. [Zugriff am 31 Mai 2016].
- [5] Vijay John, Toyota Technological Institute, „Pedestrian detection in thermal images using adaptive fuzzy C-means clustering and convolutional neural networks,“ in *IAPR International Conference*, Tokyo, 2015.
- [6] M. Majurnder, „Artificial Neural Network,“ in *Impact of Urbanization on Water Shortage in Face of Climatic Aberrations*, Springer Singapore, 2015, pp. 49-54.

## ABBILDUNGEN

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

## ABKÜRZUNGEN

a. A.	anderer Ansicht
ABl.	Amtsblatt
A.C.	Law Report, Appeal Cases
AGD	Allianz Deutscher Designer
Bd.	Band
BGH	Bundesgerichtshof
BPatG	Bundespatentgericht
BReg	Bundesregierung
BT	Bundestag
CA	Court of Appeal
CDPA	Copyright, Designs and Patents Act
DPMA	Deutsches Patent- und Markenamt
EG	Europäische Gemeinschaft
EU	Europäische Union
EGV	Vertrag zur Gründung der Europäischen Gemeinschaft
EIPR	European Intellectual Property Review
EuG	Gericht Erster Instanz
EuGH	Europäischer Gerichtshof
ff.	fortfolgende
Fn.	Fußnote
FS	Festschrift
GeschmMG	Geschmacksmustergesetz
GGVO	Gemeinschaftsgeschmacksmusterverordnung
GRUR	Gewerblicher Rechtsschutz und Urheberrecht
GRUR Int.	Gewerblicher Rechtsschutz und Urheberrecht, internationaler Teil
HABM	Harmonisierungsamt für den Binnenmarkt
HfG	Hochschule für Gestaltung
h.L.	herrschende Lehre
h.M.	herrschende Meinung
HMA	Haager Musterabkommen

# ERKLÄRUNG

Hiermit erkläre ich, dass