Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Lehrstuhl Rechnerarchitektur

Studienarbeit

Evolution eines Algorithmus zur Steuerung eines virtuellen Fahrzeugs in einer Rennsimulation

Thorsten Tiede

Betreuer: PD Dr. Marc Ebner
Wilhelm-Schickard-Institut für Informatik

Begonnen am: 1. Oktober 2008

Beendet am: 8. Juni 2009

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Tübingen am 8. Juni 2009

____

Thorsten Tiede

Inhaltsverzeichnis

1 Einleitung **1**

2 Material (verwendete Software) **2**

2.1 The Open Racing Car Simulator 2
2.2 Streckeneditor 2
2.3 Kommunikationspaket 3
2.3.1 Serverbot 3
2.3.2 Java-Client 3
2.4 Evolutionsbibliothek ECJ 6
2.5 Eclipse 7
2.6 Java Laufzeitumgebung 7

3 Methoden (Ein evolutionärer Algorithmus zur Evolution eines Fahrreglers) **8**

3.1 Algorithmus 8
3.1.1 Initialisierung 8
3.1.2 Evaluation 8
3.1.3 Generationswechsel 10
3.1.4 Terminierung 10
3.2 Pseudocode der Algorithmen 10
3.2.1 Evolution der Lenkung und aller Fahrfunktionen auf einer Strecke . . . 10
3.2.2 Evolution aller Fahrfunktionen auf mehreren Strecken 11
3.3 Implementierungsdetails 11
3.3.1 Initialisierung 11
3.3.2 Fitnessberechnung 12
3.3.3 Generationswechsel 14
3.3.4 Terminierung 16

4 Versuche und Ergebnisse **17**

4.1 Versuch zur Evolution der Lenkung einer Fahrzeugsteuerung 17
4.2 Versuche zur Entwicklung aller Fahrfunktionen auf einzelnen Strecken 21
4.2.1 Versuch zur Entwicklung allgemeiner Fahrfunktionen mit zwei Bäumen 21
4.2.2 Versuche zur Beobachtung des Lernverhaltens 25
4.3 Versuche zur Entwicklung allgemeiner Fahrfunktionen auf mehreren Teststrecken 28
4.3.1 Versuche mit Mengen elementarer Funktionen M_1 und M_2 29
4.3.2 Versuche mit Mengen elementarer Funktionen M_3 und M_4 33

5 Zusammenfassung **42**

Literaturverzeichnis **43**

1 Einleitung

Der Computerspielesektor ist einer der umsatzstärksten Märkte in der Unterhaltunsindustrie [uRS08]. Immer leistungsfähigere Computer und neue Spielekonsolen geben immer mehr Menschen den Zugang zu digitalen Medien. Der "dynamischste Markt", aus [Bri09], ist dabei der Onlinespiele Sektor. Das Spielerlebnis mit oder gegen menschliche Spieler aus aller Welt zieht immer mehr Fans an. Doch auch das beste Mehrspielerspiel kommt selten ohne Nicht-Spieler-Charaktere (NPCs für engl. *Non-Player-Characters*) aus. Ob beim Handeln von Waren in Rollenspielen, dem Simulieren von Sportereignissen oder Einzelspieler-Geschichten in First-Person-Shootern, man kommt nicht ohne ein gewisses Maß an künstlicher Intelligenz aus. Diese muss den wachsenden Ansprüchen der menschlichen Spieler auch in Zukunft gerecht werden, um ein lang anhaltendes Spieleerlebnis zu garantieren. Dabei gilt es das richtige Gleichgewicht im Schwierigkeitsgrad zu finden, um den Spieler nicht zu frustrieren oder zu langweilen. In beiden Fällen würde er schnell das Interesse am Spiel verlieren und im schlimmsten Fall zu einem Konkurrenzprodukt wechseln. Die Entwicklung anpassungsfähiger NPCs stellt somit auch in Zukunft eine wichtige Aufgabe bei der Entwicklung neuer Computerspiele dar. Gerade in Spielen, in denen NPCs ein hohes spielerisches Können abverlangt wird, um mit dem menschlichen Spieler konkurrieren zu können, wird es zunehmend komplexer leistungsfähige künstliche Intelligenzen zu entwickeln. Daher wird versucht die NPCs das erforderliche Verhalten selbst erlernen zu lassen. Evolutionäre Algorithmen scheinen sich daher sehr gut zur Entwicklung neuer Computerspieler zu eignen. "Ein evolutionärer Algorithmus simuliert die natürliche Evolution auf einem Computer zur Lösung eines Optimierungsproblems" aus [Ebn05]. Da es sich bei NPCs um Programme handelt, die in der virtuellen Welt agieren, liegt ein besonderes Augenmerk der Entwickler auf der genetischen Programmierung [Koz92], [Koz94], [BNKF98]. Bereits Siegel et al. entwickelten mit Hilfe genetischer Programmierung Programme die das Spiel Tetris spielen konnten [SC96]. Agapitos et al. [ATL07] entwickelten Steuerungen für eine Rennsimulation unter Verwendung Objekt-orientierter genetischer Programmierung. Dies soll hier aufgegriffen und die Tauglichkeit genetischer Programmierung für das Problem des Steuerns eines Fahrzeugs in einer virtuellen Simulationsumgebung geprüft werden. Anlässlich des IDEE Symposium on Computational Intelligence in Games, CIG (`http://www.ieee-cig.org/`) und des dort stattfindenden Wettbewerbs (`http://cig.dei.polimi.it/`), der "Car Racing Competition", zu selbstlernenden Steuerungen für Fahrzeuge in Rennsimulationen soll die Entwicklung eines möglichen Wettbewerbskandidaten angestoßen werden.

2 Material (verwendete Software)

In dieser Studienarbeit soll untersucht werden, ob sich genetische Programmierung für die Entwicklung einer Steuerung für ein Fahrzeug eignet. Dazu soll entsprechend den Vorgaben des CIG-Wettbewerbs eine Steuerung in der Simulation TORCS mit evolutionären Algorithmen programmiert werden. Der bereitgestellte Serverbot kann dabei ohne Änderung benutzt werden. Es wird die Evolutionary Computation in Java (ECJ) Bibliothek von Luke et al. zur Evolution verwendet (`http://cs.gmu.edu/~eclab/projects/ecj/`). Dazu muss ein Kommunikationspaket programmiert werden, welches mit dem CIG-Robot Daten austauschen kann. Als Entwicklungsumgebung kommt Eclipse zum Einsatz, programmiert wird auf Java Laufzeitumgebung 6.0.

2.1 The Open Racing Car Simulator

The Open Racing Car Simulator, kurz TORCS (`http://torcs.sourceforge.net`) wurde entwickelt von Eric Espié und Christophe Guionnea. Es ist ein Open Source Projekt und wird zur Zeit weiterentwickelt von Bernhard Wymann, Christos Dimitrakakis und anderen. Der Spieler kann dabei selbst das Steuer eines Rennwagens übernehmen und in einer simulierten Umgebung gegen andere Spieler oder Computergegner antreten. TORCS bietet aber auch die Möglichkeit selbst programmierte Fahrer, sogenannte Robots einzubinden. Deshalb hat es sich zu einer beliebten Testumgebung für selbstfahrende Programme etabliert und wird seit einiger Zeit für den CIG-Wettbewerb verwendet. In dieser Arbeit wird ausschließlich der Trainingsmodus verwendet. Dort kann ohne gegnerische Fahrzeuge geübt werden. Es besteht neben der Echtzeitberechnung auch die Möglichkeit das Training ohne graphische Darstellung zu berechnen. Dies hat den Vorteil, dass die Laufzeit nur vom verwendeten Rechnersystem abhängt. Dadurch kann viel Zeit eingespart werden. In dieser Arbeit wird mit TORCS Version 1.3.0 gearbeitet.

2.2 Streckeneditor

Für die Konstruktion der verschiedenen Teststrecken kommt der Streckeneditor *trackeditor-0.62* zum Einsatz (`http://www.trackeditor.sourceforge.net`). Er ermöglicht die einfache Konstruktion eigener Strecken in einer graphischen Benutzeroberfläche. Dieser war in der vorliegenden Version nur unter Windows lauffähig. Zwischenzeitlich wurde Version 0.70 veröffentlicht, welche in dieser Arbeit jedoch nicht mehr zum Einsatz kommt.

2.3 Kommunikationspaket

Die Veranstalter der Car Racing Competiton stellen ein Kommunikationspaket zur Verfügung. Dieses ist in zwei Teile gegliedert. Zunächst wird ein Robot in TORCS installiert. Er fungiert als Server und kommuniziert mit dem Clientteil des Pakets. Dieses umfasst neben den Kommunikationsmethoden auch einen Rahmen zur Entwicklung der Steuerung des Fahrzeugs.

2.3.1 Serverbot

Die Software Anleitung [Dan08] des Wettbewerbspakets erklärt die einfache Installation der beiden Robots *wcci2008trainer* und *wcci2008competition*. Während der *wcci2008competition*-Bot nur zehn Millisekunden auf eine Antwort des Clients nach dem Senden der Sensordaten wartet, akzeptiert der *wcci2008trainer*-Bot auch zehn Sekunden nach Senden der Sensordaten noch eine Antwort. Daher wird in dieser Arbeit ausschließlich der *wcci2008trainer*-Bot verwendet. Dadurch werden mögliche Abbrüche der Kommunikation durch zu lange Berechnungen vermieden. Tabelle 2.1 zeigt alle gesendeten Sensordaten. Diese können vom Client ausgelesen und verarbeitet werden. Die beiden Robots verfügen über keinerlei Intelligenz oder Erfahrung. Sie setzen lediglich die vom Client gesendeten Aktordaten um und steuern dadurch das Fahrzeug.

2.3.2 Java-Client

Der Java-Client des Wettbewerbs enthält neben den Kommunikationsmethoden einen Rahmen zur Evolution einer Fahrzeugsteuerung. In dieser Arbeit wird die Bibliothek ECJ eingesetzt, ein umfangreiches Paket zu evolutionären Algorithmen. Es wurde daher eine Schnittstelle zwischen dem Serverbot des Kommunikationspakets und ECJ programmiert und bereitgestellt. Es orientiert sich dabei an den Kommunikationsmethoden des Clientteils. Folgende Klassen bilden das Kommunikationsmodul.

- Action
- Communicator
- Connection
- InterruptingConnection
- Sender
- MessageParser

Das Action-Objekt besteht aus fünf Feldern, die mit den Aktoren für TORCS übereinstimmen. Diese werden in jedem Zeitschritt der Evaluierung gesetzt. Tabelle 2.2 zeigt alle verfügbaren Aktoren. Die Funktion *sendAction(Action toSend)* der Communicator-Klasse sendet das aktuelle Action-Objekt an TORCS. Die Antwort wird in einem *MessageParser*-Objekt verarbeitet. *MessageParser* speichert die Sensorwerte in einer Hash-Tabelle. Die Funktion *getReading(String key)* liefert den Sensorwert zum gegebenen Schlüssel *key* als Objekt. Aus diesem können die enthaltenen Daten extrahiert und benutzt werden.

Name	Bereich (Einheit)	Beschreibung.
angle	[-π, π](rad)	Winkel zwischen Fahrzeugrichtung und Streckenachse.
curLapTime	[0,-](s)	Verstrichene Zeit auf der aktuellen Runde.
damage	[0,-](point)	aktueller Schaden des Fahrzeugs.
distFromStart	[0,-](m)	Abstand des Fahrzeugs von der Startlinie entlang der Strecke.
distRaced	[0,-](m)	Zurückgelegte Strecke des Fahrzeugs seit Beginn des Rennens.
fuel	[0,-](l)	Spritmenge im Tank.
gear	{-1, 0, 1, .., 6}	Aktueller Gang (-1: Rückwärtsgang, 0: neutral)
lastLapTime	[0,-](s)	Benötigte Zeit für die letzte Runde.
opponents	[0,100](m)	Vektor von 18 Sensoren, die die Distanz zu gegnerischen Fahrzeugen in Metern messen. 18 Sensoren angeordnet alle 10 Grad, die den Bereich $[-\pi/2, +\pi/2]$ vor dem Fahrzeug abdecken.
racePos	1, 2, ...	Position im Rennen.
rpm	[2000-7000](rpm)	Umdrehungszahl des Motors.
speedX	- (km/h)	Geschwindigkeit des Fahrzeugs in Längsrichtung.
speedY	- (km/h)	Geschwindigkeit des Fahrzeugs in Querrichtung.
track	[0,100](m)	Vektor von 19 Streckensensoren vor dem Fahrzeug. Abbildung 2.1 zeigt die Orientierung der Sensoren.
wheelSpinVel	[0,-](rad/s)	Vektor von vier Sensoren, welche die Umdrehungsgeschwindigkeit der Räder messen.
trackPos	[-1, 1]	Position des Fahrzeugs relativ zur Streckenmitte -1 bedeutet linker Streckenrand, 1 bedeutet rechter Streckenrand.

Tabelle 2.1: Sensoren des Kommunikationspakets, aus [Dan08]

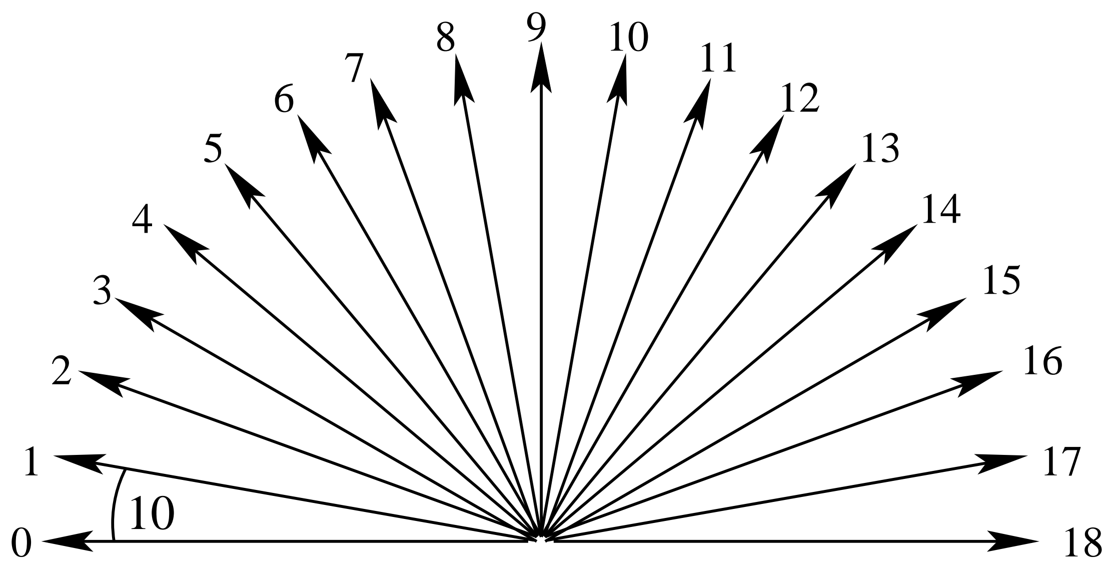


Abbildung 2.1: Streckensensoren

Die Streckensensoren sind alle 10 Grad um die Fahrzeugmitte im Bereich $-\pi/2$ bis $+\pi/2$ angeordnet. Sie geben einen Wert zwischen null und 100 in der Einheit Meter wieder und beschreiben den Abstand zur Streckenbegrenzung. Der Wert 100 Meter wird für alle Abstände größer oder gleich 100 Meter zurückgegeben.

TORCS hat ein Speicherleck, wodurch mit jedem Neustart des Rennens ca. 400 Kilobyte Speicher belegt und nicht wieder freigegeben werden. Aufgrund der hohen Anzahl an Neustarts füllt dieses Leck sehr schnell den Hauptspeicher des Rechners. Dies führt zum einen zu erheblich längeren Rechenzeiten, zum anderen zu Abstürzen des Rechners aufrund überlaufender Speicher. Da eine Analyse des Spiels keinen Aufschluss über die Ursache des Speicherlecks ergeben hat, wurde dem Kommunikationspaket die Funktionalität zum Neustart des Spiels hinzugefügt. Dazu wird mithilfe der Klasse *ProcessBuilder*[1] das Spiel aus Java heraus gestartet. Dadurch kann das Spiel jederzeit automatisch gestartet werden. Das Spiel akzeptiert beim Start keine Kommandozeilenparameter. Um nach dem Start des Spiels automatisch eine Trainingssitzung zu beginnen, muss per Hand durch das graphische Menü navigiert werden. Um eine automatisierte Neustartroutine zu ermöglichen wird ein Java-Robot[2] verwendet. Dieser erlaubt es dem Spiel Tastendruckereignisse zu übergeben. Befindet sich der Fokus auf dem TORCS Fenster, so ist es möglich, das Spiel zu jedem Zeitpunkt neu zu starten. Auch eine Funktion zur Änderung der Strecke während eines Evolutions-Laufes wurde für spätere Versuche implementiert.

Name	Bereich	Beschreibung
accel	[0, 1]	Virtuelles Gaspedal (0: kein Gas, 1: Vollgas).
brake	[0, 1]	Virtuelles Bremspedal (0: keine Bremse, 1: Volle Bremsen).
gear	-1, 0, 1, .., 6	Gangwahl.
steering	[-1, 1]	Lenkwert, -1 und 1 bedeuten vollständig links bzw. rechts entsprechend eines Winkels von 45 Grad.
meta	0, 1	Meta-Kontrolle (0: tue nichts, 1: sende Anfrage an den Server das Rennen neu zu starten).

Tabelle 2.2: Aktoren des Kommunikationspakets, aus [Dan08]

2.4 Evolutionsbibliothek ECJ

ECJ ist ein System für evolutionäre Algorithmen in Java. Es wurde flexibel gestaltet und nahezu alle Klassen werden dynamisch bei Laufzeit bestimmt. Dies geschieht durch eine durch den Nutzer bereitgestellte Parameterdatei. In dieser Arbeit wird ECJ Version 18 verwendet.
In dieser Arbeit soll ein System zur genetischen Programmierung aufgesetzt werden. “Heute versteht man unter dem Begriff genetisches Programmieren allgemein die Generierung von Computer-Programmen mit Hilfe evolutionärer Algorithmen” aus [Ebn05]. Ein Programm besteht intern dabei aus einem oder mehreren Parse-Bäumen und wird auch als Individuum bezeichnet. Mehrere Individuen ergeben eine Population. Diese wird über mehrere Generationen

[1] http://java.sun.com/j2se/1.5.0/docs/api/java/lang/ProcessBuilder.html
[2] http://java.sun.com/j2se/1.5.0/docs/api/java/awt/Robot.html

evolviert. In jeder Generation wird jedes Individuum der Population evaluiert und bewertet. Das Bewertungskriterium, die sogenannte Fitnessfunktion ist dabei maßgeblich für den Verlauf und Erfolg der Evolution. Sie muss vom Programmierer definiert werden. Im Generationswechsel werden aus den Individuen der alten Generation die Individuen der neuen Generation durch genetische Operationen erstellt.
Eine Anwendung zur genetischen Programmierung in ECJ besteht aus einer Evaluationsbeschreibung, dem *GPProblem*, einer Parameterdatei und den verwendeten Knoten und Datenstrukturen. Die Evaluationsbeschreibung beinhaltet die Anweisungen wie ein Individuum der Population evaluiert und bewertet wird. Die Parameterdatei legt die Bedingungen und Eigenschaften des GP-Systems fest. Die ECJ-Hilfe [Luk02] gibt einen Überblick über Parameter und Parameterdateien. In der Parameterdatei werden zunächst einige globale Einstellungen festgelegt. Da wir ein allgemeines GP-System nach Koza[3] aufsetzen möchten, übernehmen wir standardisierte Parameter, die bereits in *gp/koza/koza.params* festgelegt sind. Im weiteren wird unser spezielles System beschrieben. Dazu zählt neben der Populationsgröße, der Generationenanzahl und den Reproduktionsmethoden auch die Anzahl der Bäume, die ein Individuum besitzt. Jeder Baum wird definiert durch ein Set von Beschränkungen, den *TreeConstraints*. Die wichtigsten Beschränkungen sind die Art des Aufbaus der Bäume und die Mengen elementarer Funktionen zum Aufbau der Bäume.

2.5 Eclipse

Als Programmierumgebung kommt Eclipse (`http://www.eclipse.org/`) zum Einsatz.

2.6 Java Laufzeitumgebung

Es wird Java Runtime Environment 6 verwendet.

[3] http://www.genetic-programming.com/johnkoza.html

3 Methoden (Ein evolutionärer Algorithmus zur Evolution eines Fahrreglers)

3.1 Algorithmus

Der hier beschriebene Algorithmus zur Evolution des Fahrreglers ist deterministisch. Ein Versuch zweimal durchgeführt ergibt die gleichen Ergebnisse. Zu Erwähnen ist jedoch die Problematik, dass sich ein Robot in der Simulation TORCS bei Echtzeitberechnung unter Umständen anders verhält als bei beschleunigter Berechnung ohne graphische Ausgabe. Alle durchgeführten Berechnungen wurden unter beschleunigten Bedingungen durchgeführt. Es kann daher nicht gewährleistet werden, dass ein Individuum eines Experiments unter Echtzeitbedingungen die selben Fitnesswerte hervorbringt. Eine Systematik im unterschiedlichen Verhalten konnte jedoch nicht entdeckt werden. Abbildung 3.1 zeigt den dieser Arbeit zugrunde liegenden evolutionären Algorithmus.

3.1.1 Initialisierung

Ausgehend von einem gegebenen Seed wird ein Zufallszahlengenerator initialisiert. Vor Beginn der Evolution wird die Population der ersten Generation initialisiert. Diese wird mit zufällig erstellten Individuen gefüllt. Jedes Individuum besteht aus n Bäumen und einem Fitnesswert. Die Bäume werden mit einer Ramped Half-and-Half Methode durch zufälliges Ziehen von Knoten aus dem Set von Funktionen aufgebaut. Dazu gibt der Nutzer eine minimale und eine maximale Baumtiefe an. Der Algorithmus zieht eine zufällige Zahl d aus diesem Bereich. Es werden dann zur Hälfte Bäume erstellt, die genau die festgelegte Tiefe d besitzen und zur Hälfte Bäume erstellt, die zwischen einem Knoten und d Knoten Tiefe, inklusive, besitzen. Die Fitness der Individiuen wird auf den Wert Null gesetzt.

3.1.2 Evaluation

Evaluation auf einer Strecke

Jedes Individuum der aktuellen Generation wird evaluiert und erhält entsprechend der Leistung einen Fitnesswert zugewiesen. Zur Evaluation muss mit dem Rennspiel TORCS kommuniziert und das Fahrzeug auf der Teststrecke gesteuert werden. Die Zuweisung der Fitness erfolgt nach Abschluss der Testfahrt.

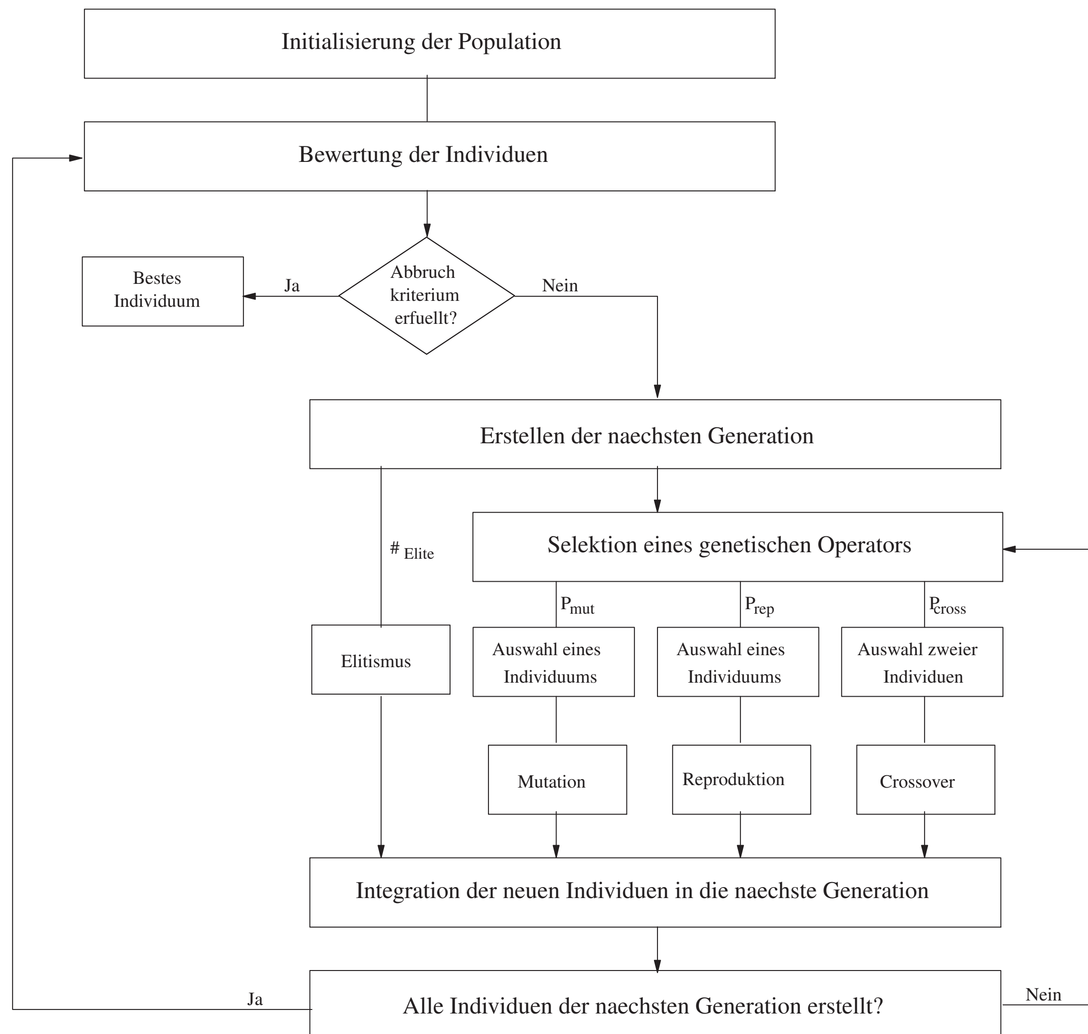


Abbildung 3.1: Flowchart des evolutionären Algorithmus

Evaluation auf mehreren Strecken

Die Population der aktuellen Generation wird s mal evaluiert. s ist die Anzahl der zu testenden Strecken. Auf jeder Strecke wird jedes Individuum evaluiert und erhält entsprechend der Leistung eine temporäre streckenbezogene Fitness zugewiesen. Zur Evaluation muss mit dem Rennspiel TORCS kommuniziert und das Fahrzeug auf der Teststrecke gesteuert werden. Nach s Evaluierungen wird für jedes Individuum der Mittelwert über alle streckenbezogenen Fitnesswerte gebildet und als Gesamtfitness dem Individuum zugeordnet.

3.1.3 Generationswechsel

Im Generationswechsel wird die aktuelle Population durch eine Neue ersetzt. Die Individuen der neuen Generation gehen durch genetische Operationen aus der alten Population hervor. Die Auswahl der Individuen für die genetischen Operationen erfolgt durch Turnier-Selektion. Die Größe der Selektion ist variabel und wird vom Nutzer festgelegt. Im Anschluss wird die neue Generation evaluiert.

3.1.4 Terminierung

Die Evolution wird beendet wenn entweder die angegebene Anzahl an Generationen evaluiert oder ein ideales Individuum gefunden wurde.

3.2 Pseudocode der Algorithmen

3.2.1 Evolution der Lenkung und aller Fahrfunktionen auf einer Strecke

Algorithmus 1 Ein evolutionärer Algorithmus zur Evolution eines Fahrreglers

Erfordert: List of Nodes, Starting-Seed, TreeConstraints
Erfordert: RandomNumberGenerator, StartingGeneration

```
 1: Initialize Evolutionary System
 2: Create Initial Population
 3: Populate Initial Population
 4: for i ← 0 upto maxGenerations do
 5:    for j ← 0 upto NumberOfIndividuals do
 6:       for k ← 0 upto MaxTimeSteps do
 7:          Send Action to TORCS and receive Sensordata
 8:       end for
 9:       Assign Fitness to Individual j
10:    end for
11:    Breed next Generation
12: end for
```

3.2.2 Evolution aller Fahrfunktionen auf mehreren Strecken

Algorithmus 2 Ein evolutionärer Algorithmus zur Evolution eines Fahrreglers

```
Erfordert: List of Nodes, Starting-Seed, TreeConstraints
Erfordert: RandomNumberGenerator, StartingGeneration
 1: Initialize Evolutionary System
 2: Create Initial Population
 3: Populate Initial Population
 4: for i ← 0 upto maxGenerations do
 5:    for each Track do
 6:       for j ← 0 upto NumberOfIndividuals do
 7:          for k ← 0 upto MaxTimeSteps do
 8:             Send Action to TORCS and receive Sensordata
 9:          end for
10:          Assign temporary Fitness to Individual j
11:       end for
12:       Calculate resulting Fitness
13:    end for
14:    Breed next Generation
15: end for
```

3.3 Implementierungsdetails

Im Folgenden wird auf einige Punkte der Implementierung eingegangen. Dabei wird sowohl die bereits vorhandene Implementierung von ECJ als auch die in dieser Arbeit geleistete Implementierung der Evaluationsbeschreibung und der Fitnessfunktion betrachtet.

3.3.1 Initialisierung

Aufgrund der langen Evaluationsdauer eines Individuums wird die Populationsgröße in allen Versuchen auf 200 Individuen beschränkt. Die maximale Generationenzahl wird auf 100 festgelegt. Die Individuen bestehen bei Entwicklung der Lenkung aus einem Baum, welcher den Lenkwert berechnet. In den folgenden Versuchen wird ein weiterer Baum für die Beschleunigung und Verzögerung (Gas und Bremse) kombiniert hinzugefügt.

Seed

Auf dem Gebiet der evolutionären Algorithmen gibt es viele Mechanismen, die auf Zufall beruhen. Zum Beispiel die Auswahl der Individuen beim Generationenwechsel oder der Knoten für den Aufbau von Bäumen. Daher bieten Systeme für evolutionäre Algorithmen Methoden um Zufallszahlen für diese Mechanismen zu ermitteln. In ECJ basieren diese Methoden auf sogenannten *Seeds*. Dies sind kurze Folgen von Zahlen, von denen ausgehend alle weiteren Zufalls-

zahlen bestimmt werden. Der Nutzer muss daher in der Parameterdatei einen Startseed angeben. Als Zufallszahlengenerator kommt ein MarsenneTwister[4] zum Einsatz.

Erzeugen der Bäume

Die von ECJ implementierte HalfBuilder-Methode orientiert sich an lil-gp[5], was im Gegensatz zu Kozas Algorithmus auch zu Bäumen der Tiefe Eins führen kann. Die minimale Baumtiefe wird auf 4, die Maximale auf 30 festgelegt.

Menge der elementaren Funktionen

Die Menge elementarer Funktionen gibt an, aus welchen Bestandteilen die Bäume aufgebaut werden können. Dabei wird unterschieden zwischen elementaren Funktionen und Terminalsymbolen. Elementare Funktionen besitzen ein oder mehrere Kinder und bilden die inneren Knoten der Bäume. Terminalsymbole besitzen keine Kinder und führen bei Einbau in den Baum zum Abschluss des aktuellen Astes. Die jeweils verwendeten Mengen elementarer Funktionen werden in Kapitel 4 bei den Versuchen angegeben. Bäume, die verschiedene Mengen und damit verschiedene Baumbeschränkungen benutzen, können in genetischen Operationen nicht miteinander kombiniert werden. Wird dagegen für alle Bäume eines Individuums dieselbe Beschränkung verwendet, so kann genetisches Material von einer Fahrzeugfunktion in den Baum einer anderen Fahrzeugfunktion transferiert werden.

Datentyp

Da die Knoten zum Aufbau der Bäume zufällig aus dem Set von Funktionen ausgewählt werden, muss sichergestellt sein, dass jeder Knoten mit jedem anderen Knoten kombinierbar ist. Dazu wird eine Datenstruktur definiert, die alle Knoten zurückgeben müssen und alle Funktionen als Eingabe für Ihre Parameter erwarten. In dieser Arbeit wird als Datentyp ein *Double* verwendet, welches in die Struktur *DoubleData* eingebettet ist. Beim Durchschreiten eines Baumes wird ein *DoubleData*-Objekt angelegt und dieses bei Evaluierung eines Knotens an die Kinder übergeben. Dadurch wird immer auf demselben Objekt gearbeitet und so Speicherplatz gespart.

3.3.2 Fitnessberechnung

Jedes Individuum erhält nach erfolgter Evaluation einen Fitnesswert zugewiesen. Die Berechnung dieses Fitnesswertes wird im folgenden erläutert. Zur Auswahl elitärer Individuen und Selektion geeigneter Individuen für genetische Operationen werden die Fitnesswerte miteinander verglichen. Dabei ist ein Fitnesswert besser als ein anderer, wenn er kleiner als dieser ist. Es wird also nach einem Individuum mit möglichst kleinem Fitnesswert gesucht und der evolutionäre Algorithmus löst ein Minimierungsproblem.

[4] http://de.wikipedia.org/wiki/Mersenne-Twister
[5] http://garage.cse.msu.edu/software/lil-gp/

Evolution der Lenkung

Jedes Individuum wird auf einer Strecke evaluiert. Die Evaluation ist dabei zeitlich nicht begrenzt. Die Simulation läuft bis das Individuum die Strecke einmal umrundet hat oder die Strecke seitlich verlässt. Die Fitness F eines Individuums berechnet sich nach Formel 3.1.

$$Fitness\ F = \begin{cases} L - D + P_S + P_T & \text{falls Strecke verlassen} \\ |L - D + P_S| & \text{falls Strecke umrundet} \end{cases} \tag{3.1}$$

Von der Streckenlänge L wird die gefahrene Distanz D abgezogen. Je weiter ein Individuum auf der Strecke fährt, desto kleiner wird somit dessen Fitness. Der Wert P_S bestraft ein Individuum, welches Werte außerhalb des benötigten Intervalls [-1, 1] produziert. Verlässt das Individuum die Strecke wird der Strafwert P_T addiert. Die Distanz D gibt die absolut gefahrene Strecke an. Wenn das Individuum Schlangenlinien fährt, kann diese größer als die Streckenlänge L werden. Da die Fitness eines Individuums nicht negativ werden darf, wird für den zweiten Fall der Absolutbetrag der Rechnung herangezogen. Dies birgt somit auch den Vorteil, dass starkes Schlangenlinienfahren bei Umrunden der Strecke zu schlechteren Werten führt als weniger starkes Schlingern.

Evolution aller Fahrfunktionen auf einer Strecke

Jedes Individuum wird auf einer Strecke evaluiert. Die Evaluation ist dabei zeitlich nicht begrenzt. Die Simulation läuft bis das Individuum die Strecke einmal umrundet hat oder die Strecke seitlich verlässt. Die Fitness F eines Individuums berechnet sich nach Formel 3.2 (Version 1) oder Formel 3.3 (Version 2).

$$Fitness\ F = \begin{cases} B - s_z & \text{falls Strecke verlassen} \\ \text{lastLapTime} & \text{falls Strecke umrundet} \end{cases} \tag{3.2}$$

$$Fitness\ F = \begin{cases} (B - s_z) * 1000 + \text{LapTime} & \text{falls Strecke verlassen} \\ \text{lastLapTime} & \text{falls Strecke umrundet} \end{cases} \tag{3.3}$$

Von einem Basiswert B wird die zurückgelegte Strecke s_z in Metern abgezogen, wenn die Strecke seitlich verlassen wurde. Umrundet das Individuum die Strecke einmal vollständig dient als Fitnesswert die benötigte Rundenzeit in Sekunden. Ein großer Nachteil dieser Methode ist, dass solange die Strecke noch nicht umrundet wurde, ein Fahrer der Schlangenlinien fährt eine größere Distanz zurücklegt als ein Fahrer der dies nicht tut. Somit wird das Schlangenlinienfahren begünstigt. In Version 2 wird daher die berechnete Differenz mit 1000 multipliziert und die bisher benötigte Rundenzeit addiert. Dadurch wird ein Individuum das zwar die Strecke an der gleichen Stelle wie ein weiteres Individuum verlassen hat begünstigt, wenn es bis zu diesem Punkt weniger Zeit benötigt hat. Desweiteren stellt sich die Frage, ob bei Evaluierung aller Generationen auf derselben Teststrecke die Beschaffenheit dieser Strecke in den evolvierten Programmen auftaucht. Es ist nicht sichergestellt, dass die Individuen generelle Fahrfunktionen erlernen.

Evolution aller Fahrfunktionen auf mehreren Strecken

Um den Nachteil des Schlangenlinienfahrens ohne Betrachtung der Rundenzeit ausgleichen zu können wird nicht die zurückgelegte Strecke gemessen, sondern die Distanz zur Startlinie entlang der Strecke betrachtet. Zusätzlich wird jede Generation auf fünf verschiedenen Strecken evaluiert. In jedem Durchlauf fährt jedes Individuum auf der aktuellen Strecke S_x $maxTimeSteps$ Zeitschritte. Das Fahrzeug startet zu Beginn des Trainings einige Meter vor der Ziellinie. Der Sensor $distFromStart$ hält zu diesem Zeitpunkt einen Wert der fast der gesamten Streckenlänge entspricht. Auf der Start-/Ziellinie springt der Wert des Sensors auf Null. Zur Ermittlung der zurückgelegten Distanz entlang der Strecke wird folgendes Verfahren eingesetzt.
Für jede Teststrecke werden drei Werte zwischengespeichert.

- Die Anfangsposition P_A des Fahrzeugs als Distanz zur Startlinie entlang der Strecke in Meter.
- Die Endposition P_E des Fahrzeugs als Distanz zur Startlinie entlang der Strecke in Meter.
- Die maximale Entfernung P_{max} des Fahrzeugs zur Startlinie entlang der Strecke in Meter.

Aus diesen drei Werten wird nach Formel 3.4 die zurückgelegte Strecke D berechnet. Die maximal zurücklegbare Distanz des Versuchsfahrzeugs D_{max} berechnet sich nach Formel 3.5. Die Anzahl der Zeitschritte eines Versuchslaufs *maxTimeSteps* dividiert durch die Anzahl der Zeitschritte in einer Sekunde *timeStepsPerSecond* ergibt die Dauer eines Laufes in Sekunden. Diese wird multipliziert mit der Maximalgeschwindigkeit des Fahrzeugs v_{max} (hier: $76.3889\frac{m}{s}$).

$$Distanz\ D = \begin{cases} -1 & \text{falls in falsche Richtung gefahren} \\ P_E - P_A & \text{falls nicht über Ziellinie gefahren} \\ P_{max} - P_A + P_E & \text{falls Ziellinie überquert} \end{cases} \tag{3.4}$$

$$Maximale\ Distanz\ D_{max} = (maxTimeSteps/timeStepsPerSecond) * v_{max} \tag{3.5}$$

Die Fitness F_{S_x} eines Individuums auf der aktuellen Teststrecke S_x ergibt sich durch Subtraktion ($F_{S_x} = D_{max} - D$). Wurden alle Individuen auf allen Teststrecken evaluiert, wird für jedes Individuum der Mittelwert über alle Fitnesswerte F_{S_x} gebildet und dieser jeweils als Gesamtfitness für dieses Individuum gesetzt.

3.3.3 Generationswechsel

Sind alle Individuen einer Generation evaluiert und ihre Fitness berechnet, werden beim Generationenwechsel die Individuen der nächsten Generation aus den Individuen der aktuellen Generation erstellt.

Genetische Operationen

Die Auswahl der Individuen für diese genetischen Operationen erfolgt durch Turnier-Selektion. Dabei werden n Individuen zufällig aus der Population gewählt. Das beste Individuum dieser

Auswahl wird für den folgendenen Prozess herangezogen. Dabei kommen je nach Versuch folgende Mechanismen, auch *Pipelines* genannt zum Einsatz. Jede Pipeline erhält vom Anwender eine Wahrscheinlichkeit mit der diese Operation ausgewählt wird. Die Summe der Wahrscheinlichkeiten aller in einem Experiment eingesetzen Pipelines muss eins ergeben.

- Crossover
  Bei dieser Methode werden zwei Individuen der Population ausgewählt und miteinander zu einem neuen Individuum kombiniert. Dazu wird in beiden Individuen zufällig je ein Baum ausgewählt. Haben die so ausgewählten Bäume die gleichen Beschränkungen, kann der *Crossover*-Vorgang eingeleitet werden. In jedem Baum wird zufällig ein Knoten ausgewählt. Die darunter liegenden Unterbäume werden dann miteinander vertauscht und das erste der beiden Individuen wird in die neue Population übernommen, das Andere wird verworfen.

- Mutation
  Hier wird ein Individuum der Population ausgewählt. Nach zufälliger Wahl eines Baumes wird ebenfalls zufällig ein Knoten des Baumes gewählt und durch einen neuen, zufällig gezogenen Knoten ersetzt. Handelt es sich bei dem gewählten Knoten um eine Funktion mit Kindern, so wird der Unterbaum durch zufälliges Wählen weiterer Knoten vervollständigt.

- Reproduktion
  Diese Methode übernimmt das durch die Turnier-Selektion gewählte Individuum in die folgende Generation.

Elitismus

Unter Elitismus versteht man den Prozess der Auswahl einer bestimmten Anzahl elitärer Individuen einer Generation um diese unverändert in die nächste Generation zu übernehmen. Dadurch wird sichergestellt, dass die am besten bewerteten Individuen nicht ausselektiert werden.

Checkpoints

ECJ bietet die Möglichkeit Zwischenstände der aktuellen Evolution, sogenannte *Checkpoints* zu speichern. Es ist dann möglich den Evolutionsprozess von diesem Checkpoint aus erneut zu starten, selbst wenn zwischenzeitlich die Problembeschreibung geändert wurde. Alle Generationen beginnend ab dem Zwischenstand werden dann nach der neuen Evaluationsbeschreibung bewertet. Um die Speicherung der Daten zu ermöglichen müssen diese serialisierbar sein. Dazu ist es nötig das Java-Interface *Serializable*[6] in allen involvierten Klassen zu implementieren. Klassen die nicht serialisiert werden können, z.B. Threads wie die Klasse *Sender* des Kommunikationspakets müssen dann mit dem Zusatz *transient* benutzt werden. Sie werden dann nach dem Start von einem Zwischenstand neu erstellt und müssen vom Nutzer zu diesem Zeitpunkt neu initialisiert werden.

[6] http://java.sun.com/javase/6/docs/api/java/io/Serializable.html

3.3.4 Terminierung

Ein ideales Individuum kann hier nicht festgelegt werden. Daher wird der Evolutionsprozess so lange fortgesetzt bis keine wesentliche Verbesserung mehr eintritt und dann von Hand beendet.

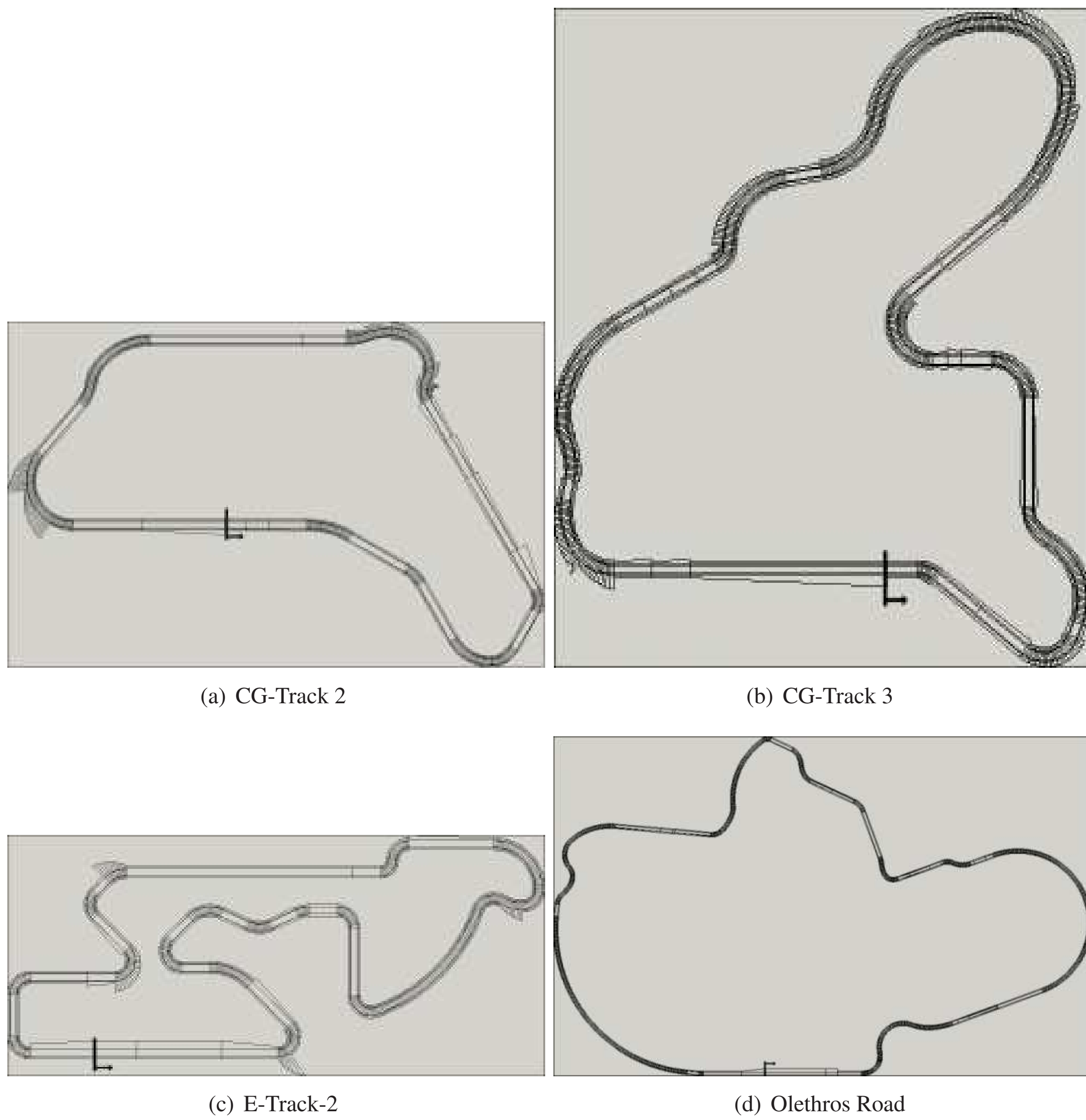
4 Versuche und Ergebnisse

Um die generelle Funktionalität des Ansatzes zu zeigen, wurde in einem ersten Versuch mit einem Baum zunächst die Lenkung für ein Fahrzeug evolviert. Dabei wurde eine konstante Beschleunigung von 20% eingestellt und das Fahrzeug im ersten Gang belassen. Dies führte zu einer konstanten Geschwindigkeit von ca. 78km/h. In weiteren Versuchen wurde ein weiterer Baum hinzugefügt, welcher die Aktoren Gas und Bremse berechnet. Da die Fahrzeuge auch schneller fahren sollen, wurde eine Automatikschaltung implementiert. Abhängig von der Motordrehzahl wird hoch- oder runtergeschalten.
Für die Versuche wurden verschiedene Teststrecken verwendet. Dieser Versuch wurde auf der einfachen dem Spiel beiliegenden Strecke CG-Track2 durchgeführt. Die Versuche zur Evolution aller Fahrfunktionen auf je einer Strecke fanden auf verschiedenen dem Spiel mitgelieferten Strecken statt. Abbildung 4.1 zeigt die Strecken aus der Vogelperspektive. Bei den Versuchen wird mit Hilfe des Namens auf die verwendete Strecke verwiesen. Für die Experimente zur Entwicklung allgemeiner Fahrfunktionen auf mehreren Strecken wurden mit Hilfe des Streckeneditors fünf eigene Strecken erstellt. Es werden dadurch auf jeder Strecke andere Fahrmanöver gefordert und die Evolution gezwungen generelle Fahrzeugfunktionen zu entwickeln. Das Erlernen einer einzelnen Strecke ist somit ausgeschlossen. Abbildung 4.6 zeigt die fünf Strecken aus der Vogelperspektive mit Startpunkt und Fahrtrichtung.

Die im folgenden gezeigten Kurven zeigen jeweils das beste Individuum einer Generation. In jeder Generation finden sich auch Individuen, die schlechter bewertet werden. Um den Verlauf der Evolution zu zeigen wird hier aber nur auf das jeweils beste Individuum verwiesen. Allen Versuchen liegt eine Populationsgröße von 200 Individuen zugrunde. Desweiteren werden die Individuen für genetische Operationen mit einer Turnier-Selektion mit Turniergröße sieben ausgewählt.

4.1 Versuch zur Evolution der Lenkung einer Fahrzeugsteuerung

Evolviert wurde hier ein Baum, dessen Knoten und Blätter aus dem in Tabelle 4.1 aufgeführten Set von Funktionen aufgebaut werden. Bereits nach wenigen Generationen entwickelten sich Individuen, die eine ganze Runde auf der einfachen Teststrecke absolvieren konnten. Dabei war zu beobachten, dass die Qualität der evolvierten Lenkung stark von der gefahrenen Strecke abhängt. Beginnt eine Teststrecke mit einer Rechtskurve (CG-Track2), entwickeln die Fahrer eine gute Rechtslenkung, scheitern jedoch am gleichmäßigen Lenken nach Links. Beginnt eine Strecke mit einer Linkskurve (Olethros-Road) ist das entgegengesetzte Verhalten zu beobachten.
Folgende Punkte beschreiben das Experiment:



(a) CG-Track 2

(b) CG-Track 3

(c) E-Track-2

(d) Olethros Road

Abbildung 4.1: Strecken der Versuche mit großer Menge elementarer Funktionen

- Anzahl Bäume: 1
- Anzahl Mengen elementarer Funktionen: 1
- Menge elementarer Funktionen: siehe Tabelle 4.1
- Genetische Operationen: Crossover (90%), Reproduction (10%)
- Berechnung des Aktors: Werte die außerhalb des benötigten Intervalls [-1, 1] liegen, werden auf die nächstliegende Intervallgrenze gesetzt.
- Fitnessberechnung: $L = 3185.828$; $P_S = 4000$; $P_T = 10000$
- Teststrecke: CG-track2 (15 Meter breit, einfache mittellange Strecke; erste Kurve: lange offene rechtskurve 45 Grad)

Abbildung 4.2 zeigt den Verlauf eines Experiments auf der Strecke CG-Track2. Es ist zu sehen, dass in den ersten 24 Generationen die Bewertung der Individuen relativ konstant hoch bleibt. Die Individuen fahren geradeaus bis zur ersten Kurve und verlassen die Strecke. Reagiert ein Individuum auf die nahende Kurve, verbessert sich dessen Fitness, was zu einer bevorzugten Auswahl während des Generationswechsels führt. Ab Generation 25 finden sich solche Individuen in der Population, welche die Strecke umrunden und daher die Rundenzeit in Sekunden als Fitness erhalten. In den nächsten 8 Generationen tritt keine weitere Verbesserung ein, weshalb das Experiment an dieser Stelle beendet wurde. Die in diesem nach Koza's Vorbild aufgesetzten Experiment benutzte ReproductionPipeline wurde in den folgenden Versuchen entfernt und stattdessen eine MutationPipeline eingeführt. In den finalen Experimenten wurde durch Elitismus ein ähnlicher Mechanismus wieder in die Evolution eingebracht.

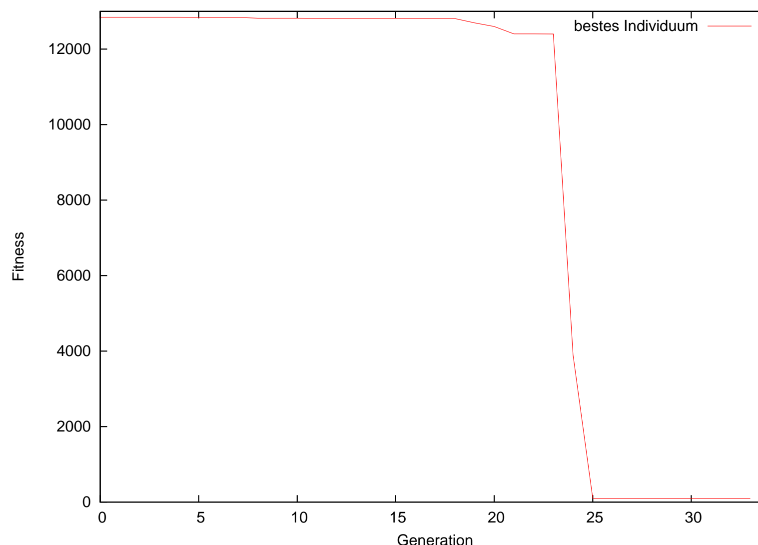


Abbildung 4.2: Versuchsverlauf des Experiments zur Evolution einer Lenkung

Name	Parameter	Beschreibung.
PlusOne	0	Gibt die Zahl +1.0 zurück.
PlusZeroPointFive	0	Gibt die Zahl +0.5 zurück.
Zero	0	Gibt die Zahl 0 zurück.
MinusZeroPointFive	0	Gibt die Zahl -0.5 zurück.
MinusOne	0	Gibt die Zahl -1.0 zurück.
TrackLeft	0	Gibt den Mittelwert der sechs linken Streckensensoren zurück.
TrackFront	0	Gibt den Mittelwert der sieben nach vorne gerichteten Streckensensoren zurück.
TrackRight	0	Gibt den Mittelwert der sechs rechten Streckensensoren zurück.
Abs	1	Gibt den Absolutbetrag des Parameters zurück.
Sqrt	1	Gibt die Quadratwurzel des Absolutbetrages des Parameters zurück.
Add	2	Gibt die Summe der beiden Parameter zurück.
Sub	2	Gibt den Wert der Differenz der beiden Parameter zurück.
Mul	2	Gibt das Produkt der beiden Parameter zurück.
Div	2	Gibt den Quotient der beiden Parameter zurück (sichere Division durch null).
IfThen	4	Gibt den dritten Parameter zurück, wenn der Erste kleiner als der Zweite ist, und sonst den vierten Parameter.

Tabelle 4.1: Menge elementarer Funktionen für das Experiment zur Evolution einer Lenkung

Die schnell erzielten guten Ergebnisse zeigen eine gute Machbarkeit der Aufgabenstellung bei langsamer Fahrt. Es gilt zu untersuchen, inwieweit auch die Steuerung der anderen Fahrzeugfunktionen, Gas und Bremse, durch genetische Programmierung realisiert werden können.

4.2 Versuche zur Entwicklung aller Fahrfunktionen auf einzelnen Strecken

Im nächsten Schritt wird dem obigen Experiment daher ein weiterer Baum hinzugefügt, dessen Rückgabewert für die Aktoren Gas und Bremse benutzt wird. Dabei bestimmt der Betrag des Wertes die Stärke, das Vorzeichen die Art der Aktion. Ein Positiver Wert wird dem Gaspedal, ein negativer Wert dem Bremspedal zugeführt, wobei der jeweils andere Aktor auf den Wert Null gesetzt wird, was keiner Aktion entspricht. Die Rennsimulation TORCS erwartet dabei für Gas und Bremse Werte im Bereich [0, 1]. In den frühen Versionen der Experimente wurden dazu alle Rückgabewerte des Baums außerhalb des Bereichs [-1, 1] auf die jeweils nächste Intervallgrenze gesetzt und dem Individuum ein Strafwert zu seiner Bewertung hinzuaddiert. Dies hat sich im Verlauf der Experimente als wenig nützliches Instrumentarium erwiesen. Daher werden in späteren Experimenten die Rückgabewerte in sigmoiden Funktionen verrechnet und der Ausgabebereich dieser Funktion auf den benötigten Bereich normiert. Dadurch können unabhängig der Rückgabewerte die Intervalle eingehalten und Unterschiede in höheren Wertebereichen berücksichtigt werden.
Zusätzlich wird die Menge der elementaren Funktionen erweitert. Tabelle 4.2 zeigt die verfügbaren Knoten. Dabei werden die drei Streckensensoren durch fünf neue ersetzt, welche eine feinere Abtastung der Streckengrenzen ermöglichen. Desweiteren werden die fünf konstanten Werte {-1.0, -0.5, 0, 0.5 1.0} durch sogenannte *Ephemeral Random Constants* ersetzt. Diese Zahlen werden bei Initialisierung des Individuums zufällig aus dem gegebenen Intervall gewählt und bleiben für die gesamte Lebensdauer des Individuums konstant. Daneben werden die mathematischen Funktionen Sinus und Cosinus eingeführt, sowie eine weitere drei-parametrige If-Abfrage hinzugefügt um einen einfachen Mechanismus zur Erkennung negativer Zahlen bereitzustellen. Auch die Sensoren für die Eigengeschwindigkeit des Fahrzeugs in Fahrtrichtung und Querrichtung stehen zur Wahl, werden jedoch nicht in allen Experimenten verwendet. Diese Änderungen sollen der Evolution bessere Möglichkeiten zur Entwicklung potenter Individuen geben. Die in Tabelle 4.2 gezeigte Menge von Funktionen beschreibt alle verfügbaren Knoten. Bei den Experimenten werden die Namen der Knoten aufgeführt die für den jeweiligen Versuchslauf verwendet wurden. Besitzt ein Experiment zwei Mengen elementarer Funktionen, so wird Menge eins dem ersten Baum (Lenkung) und Menge zwei dem zweiten Baum (Gas/Bremse) zugeordnet. Das verwenden mehrerer Mengen bringt den Vorteil, dass bei genetischen Operationen nur Bäume gleicher Fahrfunktionen miteinander überkreuzt und Unterbäume ausgetauscht werden können.

4.2.1 Versuch zur Entwicklung allgemeiner Fahrfunktionen mit zwei Bäumen

Der erste Test mit zwei Bäumen fand auf der einfachen Strecke CG-Track2 statt. Es zeigen sich auch hier sehr schnell sehr gute Ergebnisse. Folgende Liste zeigt wieder die Eigenschaften des Versuchs:

Name	Parameter	Beschreibung
Const1_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-1,1] zurück.
Const500_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-500,500] zurück.
TrackLeft	0	Gibt den Mittelwert der Streckensensoren [0,1,2,3] zurück.
TrackLeftFront	0	Gibt den Mittelwert der Streckensensoren [4,5,6,7] zurück.
TrackFront	0	Gibt den Mittelwert der Streckensensoren [8,9,10] zurück.
TrackRightFront	0	Gibt den Mittelwert der Streckensensoren [11,12,13,14] zurück.
TrackRight	0	Gibt den Mittelwert der Streckensensoren [15,16,17,18] zurück.
Angle	0	Gibt den Winkel des Fahrzeugs zur Streckenachse in Radiant zurück.
SpeedX	0	Gibt die Eigengeschwindigkeit des Fahrzeugs in Fahrtrichtung zurück.
SpeedY	0	Gibt die Eigengeschwindigkeit des Fahrzeugs quer zur Fahrtrichtung zurück.
Abs	1	Gibt den Absolutbetrag des Parameters zurück.
Sqrt	1	Gibt die Quadratwurzel des Absolutbetrages des Parameters zurück.
Sin	1	Gibt den Sinus des Parameters zurück.
Cos	1	Gibt den Cosinus des Parameters zurück.
Add	2	Gibt die Summe der beiden Parameter zurück.
Sub	2	Gibt den Wert der Differenz der beiden Parameter zurück.
Mul	2	Gibt das Produkt der beiden Parameter zurück.
Div	2	Gibt den Quotient der beiden Parameter zurück (sichere Division durch Null).
IfThen3	3	Gibt den zweiten Parameter zurück, wenn der Erste kleiner ist als Null, und sonst den dritten Parameter.
IfThen4	4	Gibt den dritten Parameter zurück, wenn der Erste kleiner als der Zweite ist, und sonst den vierten Parameter.

Tabelle 4.2: Verfügbare Knoten der Experimente zur Entwicklung aller Fahrfunktionen mit großer Menge elementarer Funktionen

- Anzahl Bäume: 2
- Anzahl Mengen elementarer Funktionen: 2
- Mengen elementarer Funktionen:
  Menge 1: CONST1_ERC, TrackLeft, TrackLeftFront, TrackFront, TrackRightFront, TrackRight, Angle, IfThen3, IfThen4, Add, Sub, Mul, Div, Abs, Sqrt, Sin, Cos
  Menge 2: CONST1_ERC, CONST500_ERC, TrackLeft, TrackLeftFront, TrackFront, TrackRightFront, TrackRight, Angle, SpeedX, SpeedY, IfThen3, IfThen4, Add, Sub, Mul, Div, Abs, Sqrt, Sin, Cos
- Genetische Operationen: Crossover (90%), Reproduction (10%)
- Berechnung Aktoren: Werte die größer oder kleiner als benötigtes Intervall sind, werden auf nächstliegende Intervallgrenze gesetzt.
- Fitnessberechnung: Version 1; $B = 1000000$
- Teststrecke: zunächst CG-Track2 (15 Meter breit, einfache mittellange strecke; erste Kurve: lange offene rechtskurve 45 Grad), ab Generation 91 Olethros-Road (schmale, kurvige und lange Strecke; erste Kurve: 90 Grad links)

Abbildung 4.3 zeigt nur das Fitness-Intervall [0, 400] um die Entwicklung der Individuen herauszustellen, welche die Strecke umrundet haben. In den Generationen Null bis Drei und 91 bis 99 liegen die Fitnesswerte im Bereich 995001.4 bis 999966.3 und werden hier nicht näher betrachtet.
Es ist zu sehen, dass auf der einfachen Strecke bereits in der vierten Generation Fahrzeuge die Strecke umrunden können. Ab Generation 20 findet keine weitere Verbesserung statt. In Generation 91 wurde die Strecke auf Olethros Road geändert um zu untersuchen, ob die Individuen eine generelle Fähigkeit zum Fahren eines Fahrzeugs entwickelt haben, oder lediglich die gewählte Teststrecke CG-Track2 erlernt haben. Da die Strecke Olethros Road schmaler ist als CG-Track2 bleiben die meisten der Individuen am Start stehen und bremsen. Sie reagieren offensichtlich auf die näher liegende Streckenbegrenzung. Bereits 9 Generationen später können einige Individuen die Strecke wieder umrunden. Die erzielten Fitness-Werte sind größer, da zum Umrunden der längeren Strecke Olethros Road mehr Zeit benötigt wird. Es gilt zu untersuchen ob die Individuen nun die neue Strecke gelernt haben, oder lediglich die schmalere Strecke die Ursache für die 9 schlechten Generationen war. Dazu müssten die Streckensensoren normiert werden in Bezug auf die Streckenbreite. Dies ist Aufgrund der Beschaffenheit der Sensoren jedoch nicht möglich. Der Wertebereich der Sensoren umfasst das Intervall [0,100]. Das bedeutet bei einem Sensorwert von 100 nicht zwingend, dass die Streckenbegrenzung in 100 Metern Entfernung liegt. Jede Entfernung größer oder gleich 100 Meter gibt den maximalen Sensorwert 100 zurück.

Weitere Versuche auf anderen Strecken sollen die allgemeine Tauglichkeit des Ansatzes überprüfen. Desweiteren waren die in obigen Experiment beobachteten Rundenzeiten recht langsam. Eine Analyse der Fahrweise ergab, dass die Individuen keinen Gebrauch von der Bremse machen. Die folgenden Experimente sollen zeigen, ob dies ein einmaliges Phänomen darstellt, oder ein allgemeines Problem dieses Ansatzes sein könnte.

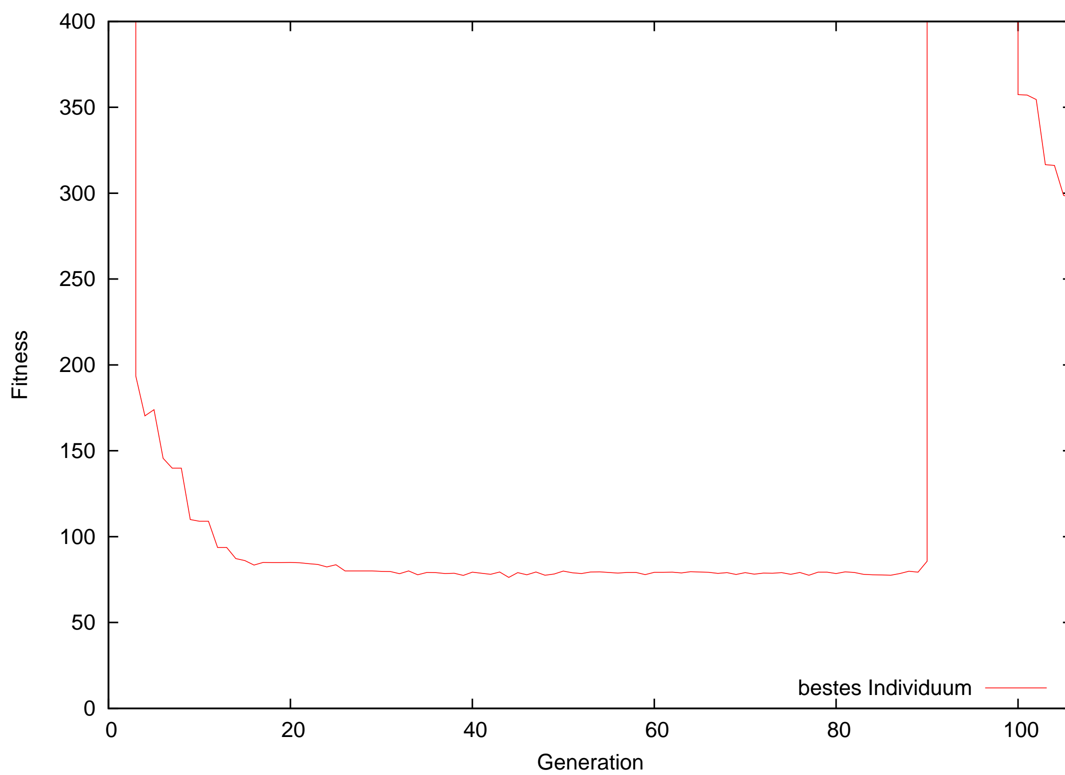


Abbildung 4.3: Fitness des jeweils besten Individuums einer Generation im Experiment zur Entwicklung allgemeiner Fahrfunktionen mit zwei Bäumen

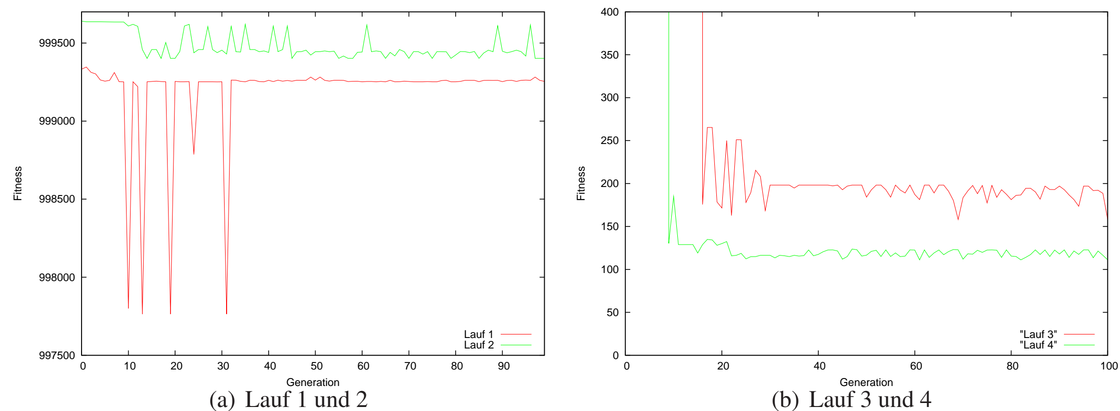
4.2.2 Versuche zur Beobachtung des Lernverhaltens

Die im Folgenden aufgeführten Experimente unterscheiden sich lediglich im Startseed und der Teststrecke. Lauf 1 und Lauf 2 fanden auf der schweren Olethros-Road statt. Die Individuen aus obigem Experiment konnten die Strecke umrunden nachdem sie auf der einfachen Strecke CG-Track 2 einfache Fahrfunktionen scheinbar erlernen konnten. Lauf 3 und Lauf 4 fanden auf der Strecke CG-Track 3 statt. Diese hat eine geringere Streckenbreite als CG-Track 2 und engere Kurven. Es soll untersucht werden, ob die Fahrfunktionen auch direkt auf den schwereren Strecken erworben werden können, oder die Strecken zu anspruchsvoll für diese Aufgabe sind.

Erlernen genereller Fahrfunktionen auf unterschiedlichen Strecken

- Anzahl Bäume: 2
- Anzahl Mengen elementarer Funktionen: 2
- Mengen elementarer Funktionen:
  Menge 1: CONST1_ERC, CONST500_ERC, TrackLeft, TrackLeftFront, TrackFront, TrackRightFront, TrackRight, Angle, IfThen3, IfThen4, Add, Sub, Mul, Div, Abs, Sqrt, Sin, Cos
  Menge 2: CONST1_ERC, CONST500_ERC, TrackLeft, TrackLeftFront, TrackFront, TrackRightFront, TrackRight, Angle, SpeedX, SpeedY, IfThen3, IfThen4, Add, Sub, Mul, Div, Abs, Sqrt, Sin, Cos
- Genetische Operationen: Crossover (90%), Reproduction (10%)
- Berechnung Aktoren: Werte die größer oder kleiner als benötigtes Intervall sind, werden auf nächstliegende Intervallgrenze gesetzt.
- Fitnessberechnung: Version 1; $B = 1000000$
- Teststrecke: Lauf 1: Olethros-Road (schmale, kurvige und lange Strecke; erste Kurve: 90 Grad links)
  Lauf 2: Olethros-Road
  Lauf 3: CG-Track 3 (schmale, kurvige, mittellange Strecke; erste Kurve: 80 Grad rechts)
  Lauf 4: CG-Track 3

In Abbildung 4.4(a) ist zu sehen, dass die Läufe eins und zwei zu keinem positiven Ergebnis führen. Keines der Individuen kann die Strecke umrunden. Die starke Schwankung der Fitness ist auf fehlenden Elitismus zurückzuführen. Gute Individuen werden nicht sicher in die nächste Generation übernommen. Dagegen bringen die Läufe drei und vier (Abbildung 4.4(b) bereits nach weniger als 20 Generationen potente Individuen hervor welche die Strecke umrunden können. Hier ist wieder nur der relevante Wertebereich der Fitnesswerte gezeigt, um den Fitnessverlauf der Individuen zu zeigen, welche die Strecke umrunden können. Es kann zunächst festgehalten werden, dass sich die Strecke Olethros Road nicht zum Erlernen der Grundfunktionen eignet, eine einfachere Strecke hingegen durchaus potente Individuen hervorbringt.



(a) Lauf 1 und 2 (b) Lauf 3 und 4

Abbildung 4.4: Fitness des jeweils besten Individuums zum Erlernen genereller Fahrfunktionen auf unterschiedlichen Strecken; 4.4(a): Lauf 1 und 2; 4.4(b): Lauf 3 und 4

Eine Analyse des Fahrverhaltens der Individuen aller vier Läufe zeigt, dass keines der Individuen die Funktionalität der Bremse nutzt. Dies resultiert aus der Tatsache, dass in den ersten Generationen diejenigen Individuen einen Vorteil erlangen, die nicht Bremsen, sondern das Gaspedal betätigen. Die Funktionalität des Bremsens führt hier also zu einem klaren Evolutionsnachteil und stirbt aus. Die Individuen beschleunigen dann gerade so stark, dass sie die Kurven der Strecke durchfahren können, ohne das Fahrzeug abbremsen zu müssen.

Fliegender Start

Daraufhin wurde in den Läufen fünf und sechs das Fahrzeug in den ersten sechs Sekunden des Experiments voll geradeaus beschleunigt. Die gefahrene Teststrecke E-Track 2 besitzt eine lange Start/Ziel Gerade mit einer anschließenden 90 Grad Kurve nach links. Damit die Individuen die erste Kurve durchfahren können, muss die Bremse betätigt werden. Es sollte damit die Funktionalität des Bremsens zu einem Evolutionsvorteil werden. Zusätzlich wurde die sigmoide Verarbeitung der Rückgabewerte aller Bäume eingeführt und die Fitnessfunktion auf Version 2 geändert.

- Anzahl Bäume: 2
- Anzahl Mengen elementarer Funktionen: 1
- Menge elementarer Funktionen:
  CONST1_ERC, CONST500_ERC, TrackLeft, TrackLeftFront, TrackFront, TrackRightFront, TrackRight, Angle, SpeedX, SpeedY, IfThen3, IfThen4, Add, Sub, Mul, Div, Abs, Sqrt, Sin, Cos
- Genetische Operationen: Crossover (90%), Reproduction (10%)

- Berechnung Aktoren: Verrechnung der Rückgabewerte in sigmoiden Funktionen und Skalierung auf benötigte Intervalle.
- Fitnessberechnung: Version2; $B = 3150$
- Teststrecke: Lauf 5: E-Track 2 (schmale, kurvige und mittellange Strecke; erste Kurve: 80 Grad links)
  Lauf 6: E-Track 2

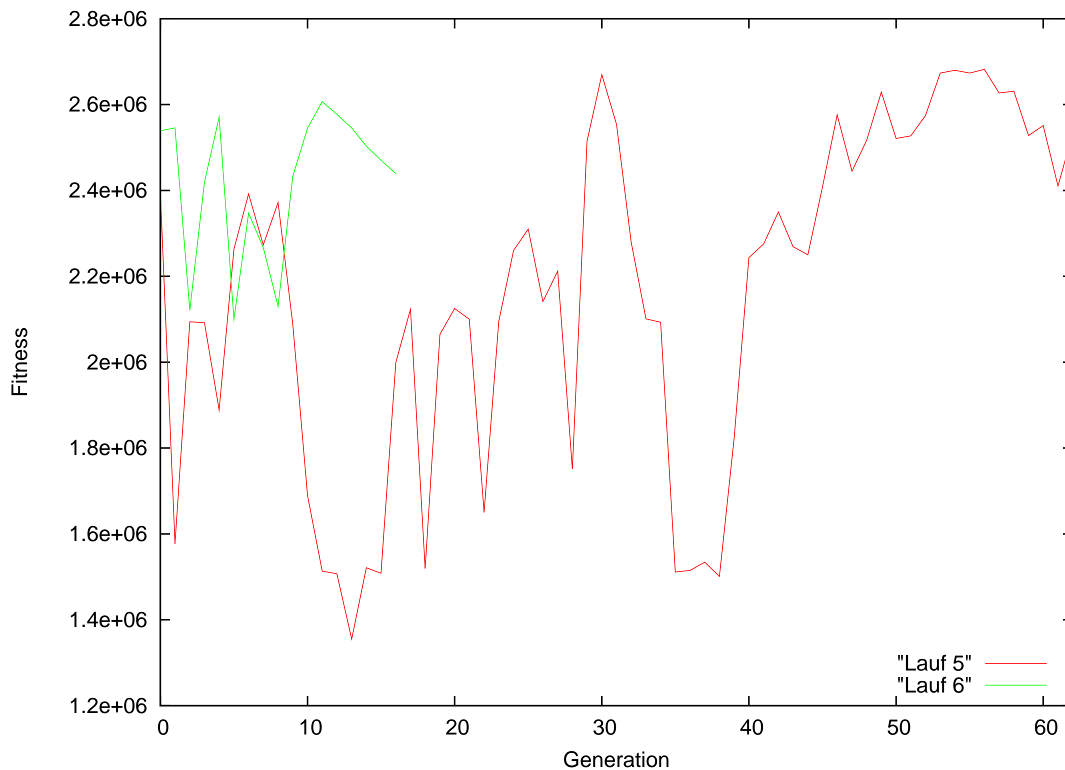


Abbildung 4.5: Fitness des jeweils besten Individuums einer Generation bei fliegendem Start

Abbildung 4.5 zeigt, dass der gewählte Ansatz keine verwertbaren Ergebnisse hervorbringt. Es findet keine Evolution statt. Lauf 6 wurde vorzeitig abgebrochen, da auch hier keine guten Ergebnisse zu erwarten waren. Die Analyse zeigt, dass die Beschleunigung in den ersten sechs Simulationssekunden keine gleichbleibenden Bedingungen schafft. Tabelle 4.3 zeigt die Abweichungen der Startbedingungen sobald das Individuum die Kontrolle über das Fahrzeug erhält. Durch verschiedene Startbedingungen sind die Individuen nicht miteinander vergleichbar und eine Selektion potenter Individuen nicht möglich.

Es konnte mit den vorliegenden Mengen elementarer Funktionen kein Individuum gefunden werden, welches generelle Fahrfunktionen erlernt. Es liegt die Vermutung nahe, dass durch die groß gewählten Mengen elementarer Funktionen der Suchraum für mögliche Lösungen zu groß

Bedingung	Einheit	minimaler Wert	maximaler Wert
Streckenposition	[-]	$2,441 * 10^{-5}$	$3,784 * 10^{-4}$
Distanz zur Startlinie	[m]	$54,281$	$55,409$
Eigengeschwindigkeit in Fahrtrichtung	[km/h]	$160,899$	$161,637$

Tabelle 4.3: Abweichungen relevanter Startbedingungen bei fliegendem Start nach 6 Sekunden konstanter Beschleunigung

ist. Das Finden einer Lösung ist damit zu unwahrscheinlich. Daher wird in einem zweiten Ansatz das Problem von Grund auf neu programmiert und die Erkenntnisse der bisherigen Versuche eingearbeitet. In diesem zweiten Ansatz soll der Evolution auf die Sprünge geholfen werden.

4.3 Versuche zur Entwicklung allgemeiner Fahrfunktionen auf mehreren Teststrecken

In jeder Generation wird die Population auf mehreren Strecken evaluiert (siehe Abbildung 4.6). Für jede Strecke erhält das Individuum einen Fitnesswert. Vor dem Generationenwechsel wird aus den einzelnen Fitnesswerten ein Gesamtwert errechnet. Es ist der Mittelwert über alle erhaltenen Fitnesswerte. So wird sichergestellt, dass ein Individuum mit mehreren Situationen zurechtkommt und nicht nur eine Strecke auswendig lernt. Die folgenden Versuche basieren auf neuen Mengen elementarer Funktionen. Diese wurden so gewählt, dass eine einfache Steuerung mit ihnen zu realisieren ist. Desweiteren wird in einigen Versuchen der initialen Population ein Individuum mit dieser einfachen Steuerung vorgegeben. Abbildung 4.7 zeigt die beiden vorgegebenen Bäume. Die Lenkung des Initialen Individuums (Abbildung 4.7(a)) besteht aus einem Proportional-Regler (Softwaregleichung siehe Gleichung 4.1 nach [RN-]) dessen Regelabweichung e durch den Sensor *LR0* berechnet wird. K_p ist der Verstärkungsfaktor des Reglers.

$$y = K_p * e \tag{4.1}$$

Der Baum für Gas und Bremse besteht ebenfalls aus einem P-Regler (Abbildung 4.7(b)), mit Verstärkungsfaktor *C_1*. Die Regelgröße ist hier die Eigengeschwindigkeit des Fahrzeugs in Fahrtrichtung (*SpeedX*), die Führungsgröße berechnet sich aus den zwei Streckensensoren *LR1* und *SensorF* sowie einer Skalierungsvariable *C_2*. Die Führungsgröße beschreibt die maximal mögliche Geschwindkeit auf dem aktuellen Streckenabschnitt. Durch Elitismus soll dann sichergestellt werden, dass dieses Individuum einen Generationenwechsel überlebt, bis die Evolution ein besseres Individuum hervor bringt. Die Rückgabewerte der Bäume werden direkt ohne sigmoide Verarbeitung oder Zuschneidung auf Wertebereiche an die Rennsimulation übermittelt. TORCS interpretiert Werte außerhalb der benötigten Intervalle als jeweiligen Maximalausschlag, weshalb eine vorherige Verrechnung der Werte nicht nötig ist. Die Tabellen 4.4 und 4.5 zeigen die zur Verfügung stehenden Terminalsymbole und elementaren Funktionen. Die Konstanten K_p, C_1 und C_2 wurden in einem externen Versuch ermittelt und dienen zur Konstrukti-

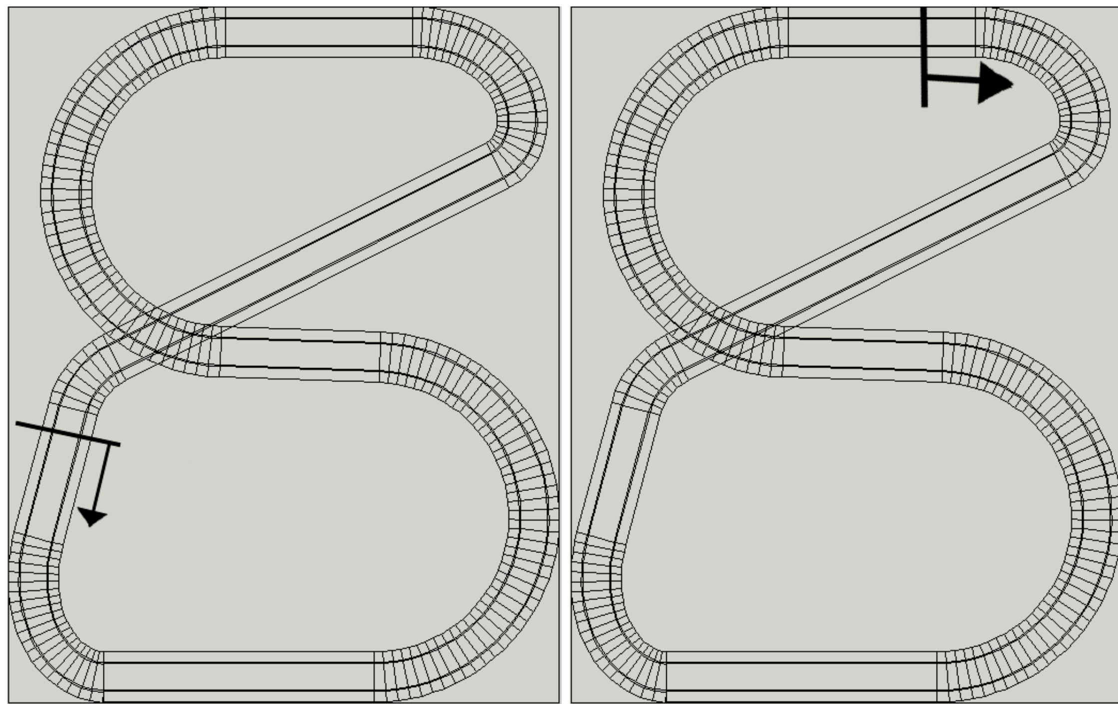
on des initialen Individuums. Die Terminalsymbole *LR*0, *LR*1 und *SensorF* ersetzen die Streckensensoren aus den Vorgängerexperimenten und wurden ebenfalls in externen Experimenten ermittelt. Sie liefern die zuverlässigsten Werte für die jeweiligen Aufgaben. Der Definitionsbereich der Ephemeral Random Constants wurde von [-500;500] verkleinert auf [-150;150], da Werte größer als 150 bzw. kleiner als -150 für die vorliegenden Versuche wenig sinnvoll sind. Zieht man zum Beispiel die Zahl 400 zum Vergleich mit der momentanen Eigengeschwindigkeit des Fahrzeugs heran, so liegt diese in jedem Fall unter der gewählten Vergleichszahl, da das Fahrzeug die Geschwindigkeit von 400 km/h nicht erreichen kann. Ein durch diesen Vergleich möglicherweise eingeleitetes Bremsmanöver findet nie statt.

Name	Parameter	Beschreibung
Const1_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-1,1] zurück.
Const150_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-150,150] zurück.
K_p	0	Konstante des Proportionalteils des Reglers. Gibt den Wert -0.0234 zurück.
LR0	0	Streckensensor; Gibt den Wert folgender Rechnung zurück: $((track[15]+track[14])-(track[3]+track[4]))/2$
Add	2	Gibt die Summe der beiden Parameter zurück.
Sub	2	Gibt den Wert der Differenz der beiden Parameter zurück.
Mul	2	Gibt das Produkt der beiden Parameter zurück.
Div	2	Gibt den Quotient der beiden Parameter zurück. (sichere Division durch Null)
Abs	1	Gibt den Absolutwert des Eingabeparameters zurück.

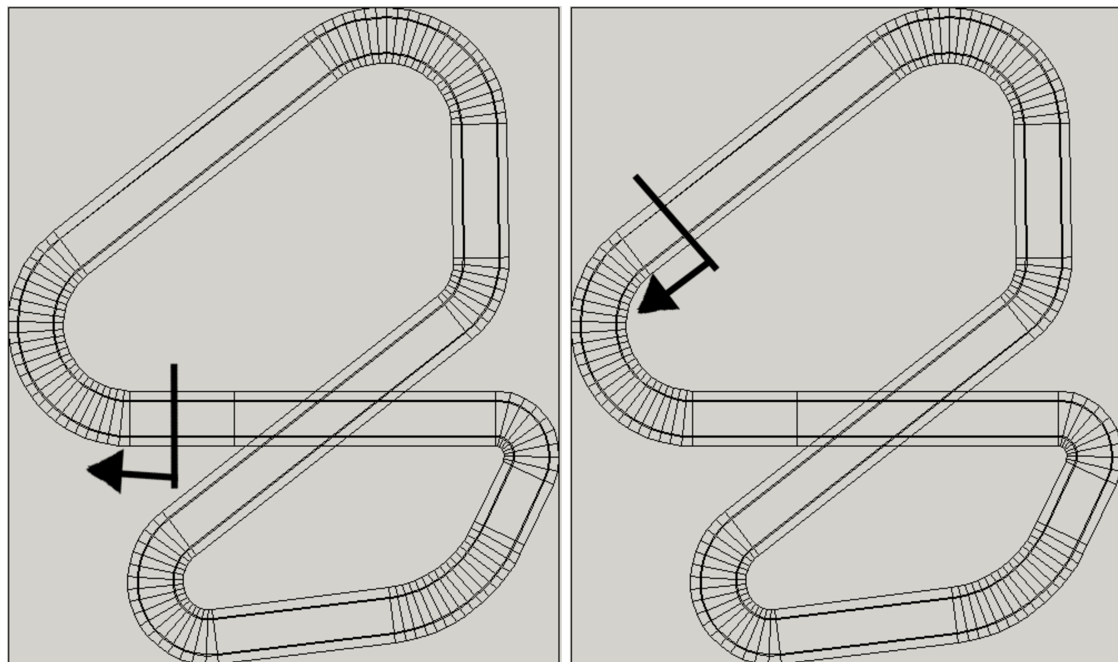
Tabelle 4.4: Menge elementarer Funktionen M_1 des Lenkbaumes

4.3.1 Versuche mit Mengen elementarer Funktionen M_1 und M_2

- Anzahl Bäume: 2
- Anzahl Mengen elementarer Funktionen: 2
- Mengen elementarer Funktionen: siehe Tabellen 4.4 und 4.5
- genetische Operationen: Crossover (50%), Mutation (50%)
- Elitismus: 3
- Berechnung Aktoren: Direkte Anbindung der Baumwerte an die Aktoren.
- Fitnessberechnung: Ansatz 2: maxTimeSteps = 1000; timeStepsPerSecond = 50
- Teststrecken: siehe Abbildung 4.6



(a) Strecke 1 (b) Strecke 2



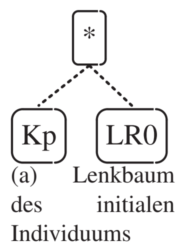
(c) Strecke 3 (d) Strecke 4

(e) Strecke 5

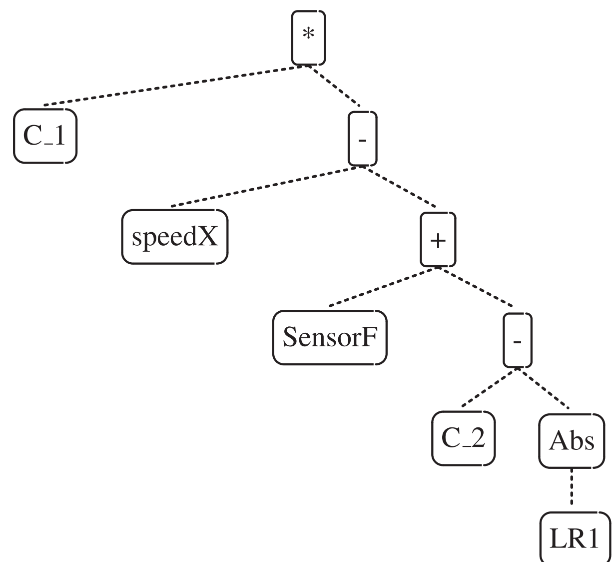
Abbildung 4.6: Strecken des 2. Ansatzes

Name	Parameter	Beschreibung
Const1_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-1,1] zurück.
Const150_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-150,150] zurück.
C_1	0	Konstante des Anfangsindividuums. Gibt den Wert -0.022 zurück.
C_2	0	Konstante des Anfangsindividuums. Gibt den Wert 100 zurück.
LR1	0	Streckensensor; Gibt den Wert folgender Rechnung zurück: $track[8] - track[10]$
SensorF	0	Streckensensor; Gibt den Wert des Sensors track[9] zurück.
SpeedX	0	Gibt die aktuelle Geschwindigkeit des Fahrzeugs entlang der Fahrtrichtung zurück.
Add	2	Gibt die Summe der beiden Parameter zurück.
Sub	2	Gibt den Wert der Differenz der beiden Parameter zurück.
Mul	2	Gibt das Produkt der beiden Parameter zurück.
Div	2	Gibt den Quotient der beiden Parameter zurück. (sichere Division durch Null)
Abs	1	Gibt den Absolutwert des Eingabeparameters zurück.

Tabelle 4.5: Menge elementarer Funktionen M_2 des Gas/Bremse Baumes

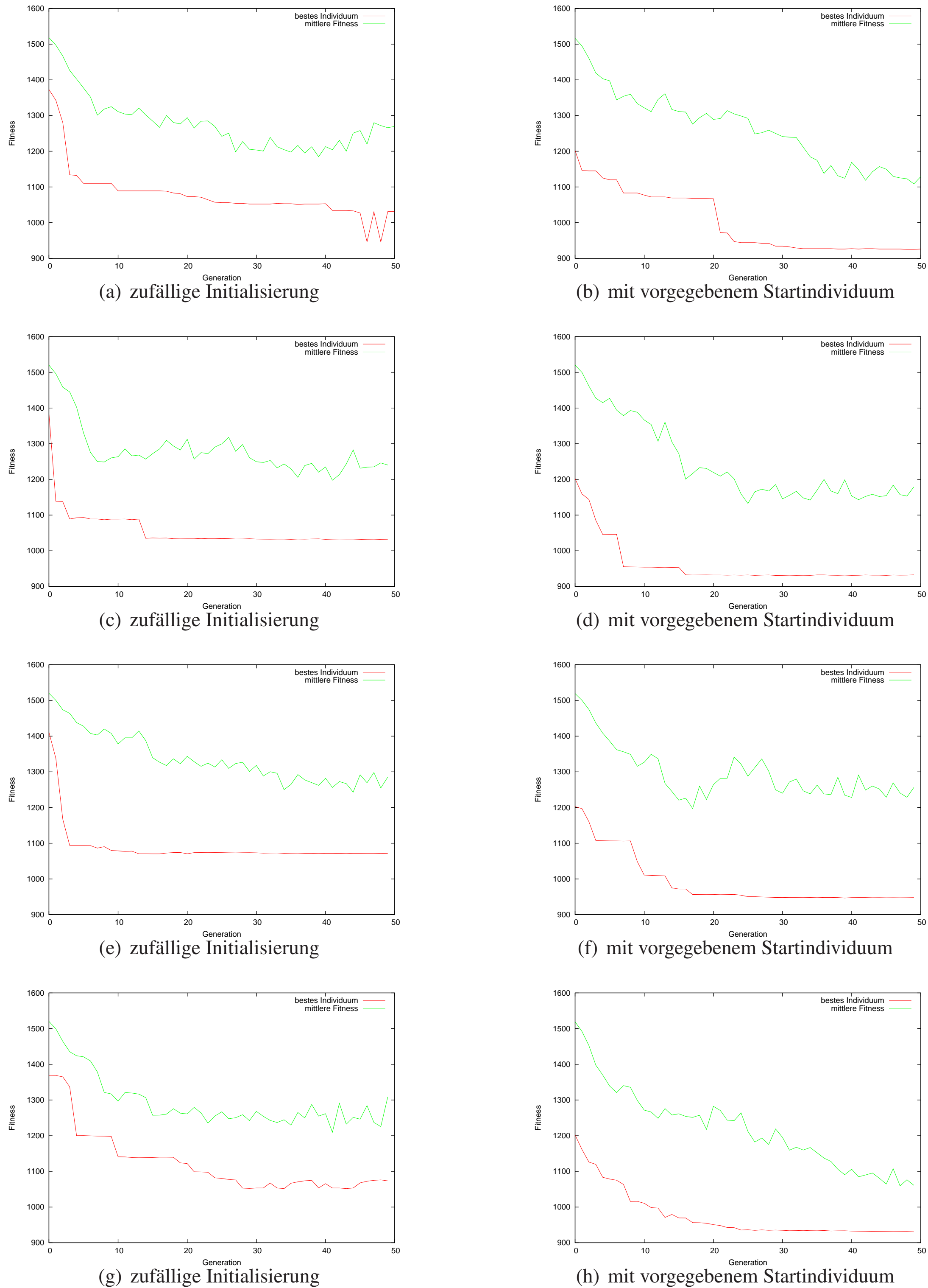


(a) Lenkbaum des initialen Individuums



(b) Parsebaum für Beschleunigung und Verzögerung des initialen Individuums

Abbildung 4.7: Parsebäume des Initialen Individuums



(a) zufällige Initialisierung
(b) mit vorgegebenem Startindividuum
(c) zufällige Initialisierung
(d) mit vorgegebenem Startindividuum
(e) zufällige Initialisierung
(f) mit vorgegebenem Startindividuum
(g) zufällige Initialisierung
(h) mit vorgegebenem Startindividuum

Abbildung 4.8: Fitness des jeweils besten Individuums und die mittlere Fitness der Generation unter Verwendung der Mengen elementarer Funktionen M_1 und M_2 und verschiedener Startseeds

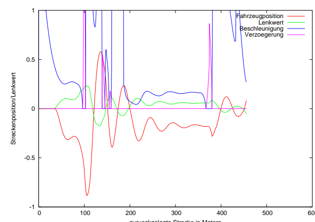
Abbildung 4.8 zeigt die Fitnessverläufe der Versuche im Vergleich. Es fällt auf, dass die Versuche mit zufälliger Initialisierung nicht die Fitnesswerte erreichen, die bei den Versuchen mit Startindividuum erzielt werden. Die Konstruktion eines guten Baumes für die Beschleunigung/-Verzögerung des Fahrzeugs scheint der Evolution unter den gegebenen Bedingungen nicht zu gelingen. Das vorgegebene Individuum der ersten Generation wird erfolgreich weiterentwickelt. Die beiden Sprünge in der Fitness des besten Individuums in Abbildung 4.8(a) resultieren daraus, dass das Fahrzeug auf der Strecke wendet und in Gegenrichtung erneut die Start/Ziellinie überquert. Dadurch steigt der Wert der zurückgelegten Distanz sprunghaft an. Da dies kein wiederkehrendes Verhalten darstellt und das Fahrzeug in der jeweils nächsten Generation beim Wendemanöver die Strecke verlässt, vererbt sich diese Eigenschaft durch Elitismus nicht. Das erfolgreiche Wendemanöver gelingt aufgrund nicht-deterministischer Simulation der Fahrten. Es handelt sich hierbei um Einzelfälle.

Abbildungen 4.9 und 4.10 zeigen die Fahrfunktionen des jeweils besten Individuums der ersten Generation im Vergleich zur Generation 49. Dargestellt sind jeweils die Position des Fahrzeugs auf der Strecke mit Bezug auf die Streckenmitte sowie die drei Aktoren Lenkausschlag, Beschleunigung (Gas) und Verzögerung (Bremse). In Versuchen, bei denen die Kurve für die Beschleunigung nicht zu sehen ist und in den Fällen wo diese den Wert eins überschreitet, wird dauerhaft Vollgas gegeben. Um die Übersicht der anderen Kurven zu wahren wurde die y-Achse jedoch nicht entsprechend umskaliert. Beim Vergleich von Generation Null und Generation 49 fällt auf, dass die Individuen ausnahmslos eine größere Strecke zurücklegen können. Die Individuen betätigen vor nahenden Kurven wie gefordert die Bremse und Verzögern das Fahrzeug um die Kurve zu durchfahren. Desweiteren gelingt es der Evolution Steuerungen hervorzubringen, die das Fahrzeug stabiler in der Streckenmitte halten. Beim Abweichen von dieser wird differenzierter gegengelenkt, das typische Schwingungsverhalten des Proportionalreglers bleibt jedoch erhalten. Um dieses zu minimieren soll die Evolution in den nächsten Versuchen die Möglichkeit erhalten komplexere Reglereinheiten umzusetzen.

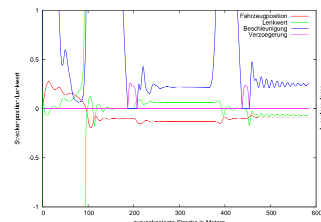
4.3.2 Versuche mit Mengen elementarer Funktionen M_3 und M_4

- Anzahl Bäume: 2
- Anzahl Mengen elementarer Funktionen: 2
- Mengen elementarer Funktionen: siehe Tabellen 4.6 und 4.7
- genetische Operationen: Crossover (50%), Mutation (50%)
- Elitismus: 3
- Berechnung Aktoren: Direkte Anbindung der Baumwerte an die Aktoren.
- Fitnessberechnung: Ansatz 2: maxTimeSteps = 1000; timeStepsPerSecond = 50
- Teststrecken: siehe Abbildung 4.6

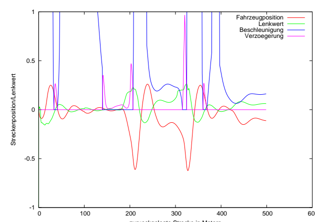
Durch die neuen Knoten *Sum* und *Old* ist es möglich neben Proportionalreglern auch Integral- sowie Differentialanteile [RN-] von Reglern zu implementieren. Gleichungen 4.2 zeigen die



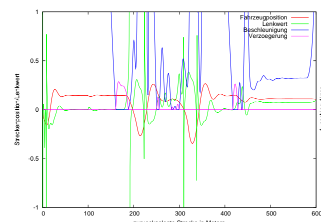
(a) Generation 0 auf Strecke 1



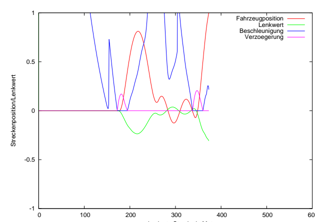
(b) Generation 49 auf Strecke 1



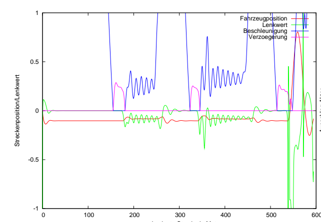
(c) Generation 0 auf Strecke 2



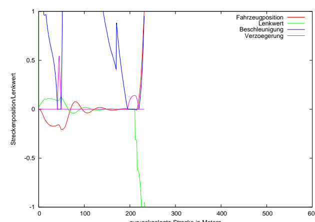
(d) Generation 49 auf Strecke 2



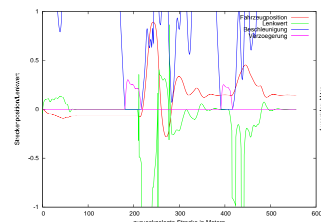
(e) Generation 0 auf Strecke 3



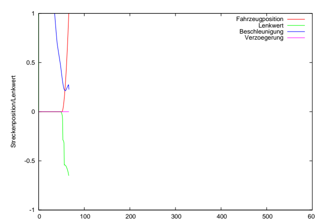
(f) Generation 49 auf Strecke 3



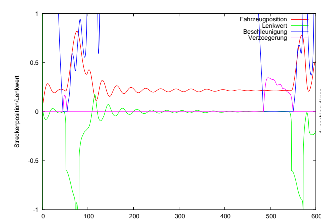
(g) Generation 0 auf Strecke 4



(h) Generation 49 auf Strecke 4

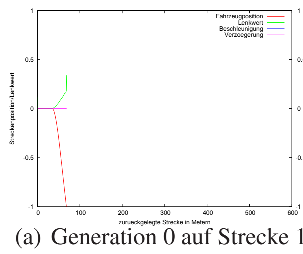


(i) Generation 0 auf Strecke 5

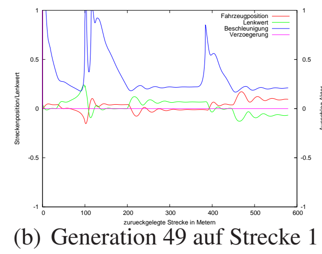


(j) Generation 49 auf Strecke 5

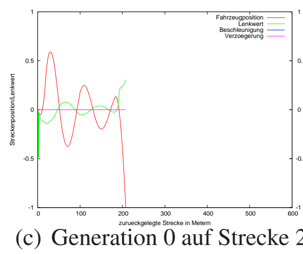
Abbildung 4.9: Vergleich der Fahrfunktionen für Generation 0 und Generation 49 mit Mengen elementarer Funktionen M_1 und M_2 und vorgegebenem Startindividuum aus Abbildung 4.7



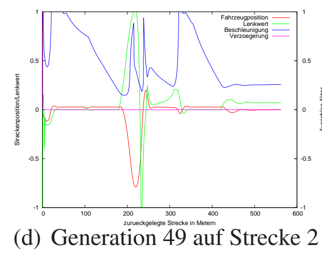
(a) Generation 0 auf Strecke 1



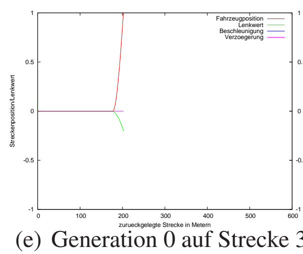
(b) Generation 49 auf Strecke 1



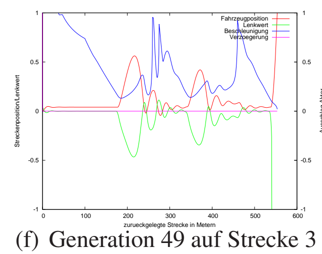
(c) Generation 0 auf Strecke 2



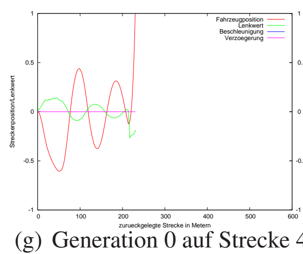
(d) Generation 49 auf Strecke 2



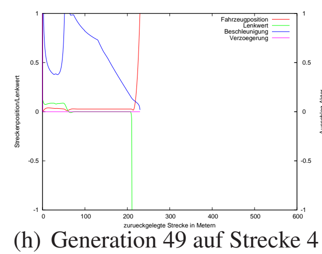
(e) Generation 0 auf Strecke 3



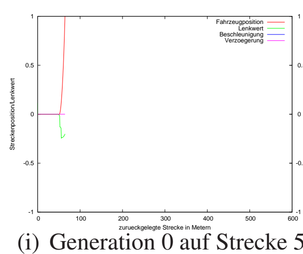
(f) Generation 49 auf Strecke 3



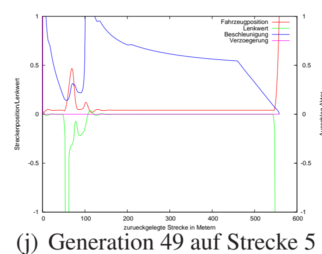
(g) Generation 0 auf Strecke 4



(h) Generation 49 auf Strecke 4



(i) Generation 0 auf Strecke 5



(j) Generation 49 auf Strecke 5

Abbildung 4.10: Vergleich der Fahrfunktionen für Generation 0 und Generation 49 mit Mengen elementarer Funktionen M_1 und M_2 ohne vorgegebenes Startindividuum

Softwaregleichungen eines Proportional-Integral-Reglers (PI-Regler), Gleichungen 4.3 einen Proportional-Differential-Regler (PD-Regler) und Gleichungen 4.4 zeigen Softwaregleichungen eines Proportional-Integral-Differential-Reglers (PID-Regler). Alle Gleichungen nach [RN-].

$$\begin{aligned} Sum(e) &= Sum(e) + e \\ y &= K_p * e + K_i * Sum(e) \end{aligned} \tag{4.2}$$

$$\begin{aligned} y &= K_p * e + K_d * (e – Old(e)) \\ Old(e) &= e \end{aligned} \tag{4.3}$$

$$\begin{aligned} Sum(e) &= Sum(e) + e \\ y &= K_p * e + K_i * Sum(e) + K_d * (e - Old(e)) \\ Old(e) &= e \end{aligned} \tag{4.4}$$

Diese Regler weisen bei gut eingestellten Parametern K_p, K_d und K_i ein geringeres Schwingungsverhalten als Proportionalregler auf. Desweiteren wird die dauerhafte Regelabweichung eines Proportionalreglers minimiert.

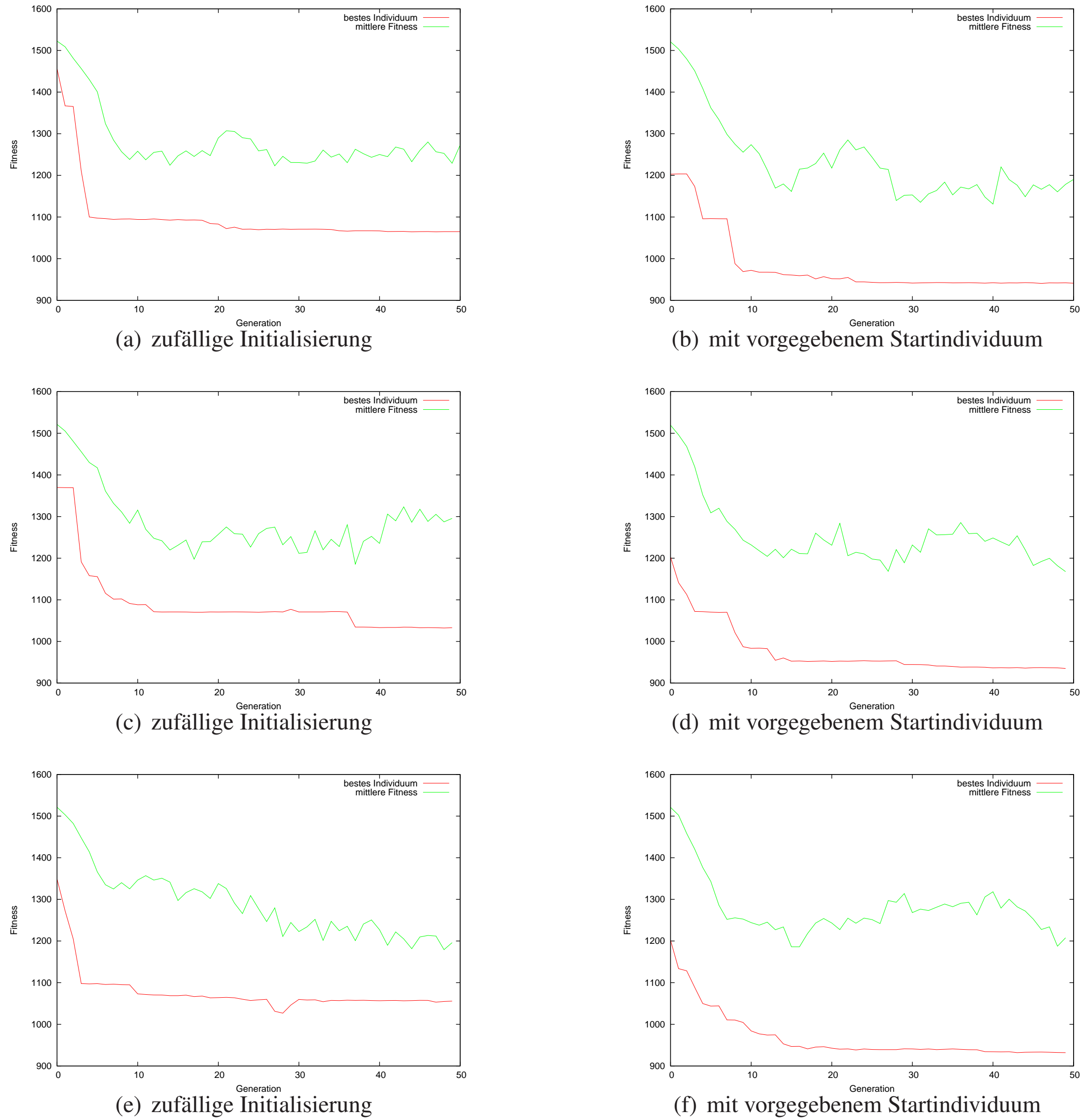
Name	Parameter	Beschreibung
Const1_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-1,1] zurück.
Const150_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-150,150] zurück.
K_p	0	Konstante des Proportionalteils des Reglers. Gibt den Wert -0.0234 zurück.
LR0	0	Streckensensor; Gibt den Wert folgender Rechnung zurück: $((track[15] + track[14]) - (track[3] + track[4]))/2$
Add	2	Gibt die Summe der beiden Parameter zurück.
Sub	2	Gibt den Wert der Differenz der beiden Parameter zurück.
Mul	2	Gibt das Produkt der beiden Parameter zurück.
Div	2	Gibt den Quotient der beiden Parameter zurück. (sichere Division durch Null)
Abs	1	Gibt den Absolutwert des Eingabeparameters zurück.
Sum	1	Summiert das Ergebnis seines Unterbaums über alle Zeitschritte auf und gibt die Summe zurück.
Last	1	Speichert den aktuellen Wert seines Unterbaumes und gibt den Wert aus dem letzten Zeitschritt zurück.

Tabelle 4.6: Menge elementarer Funktionen M_3 des Lenkbaumes

Auch hier lässt sich beobachten, dass die Evolution bei zufälliger Initialisierung (Abbildung 4.11 linke Spalte) keine Programme entwickelt, welche die Leistung der Programme aus den Versuchen mit gegebenem Startindividuum (Abbildung 4.11 rechte Spalte) erreichen. Desweiteren bringen die zusätzlichen Knoten der Mengen M_3 und M_4 keinen Vorteil gegenüber

Name	Parameter	Beschreibung
Const1_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-1,1] zurück.
Const150_ERC	0	Gibt eine Random Ephemeral Constant aus dem Bereich [-150,150] zurück.
C_1	0	Konstante des Anfangsindividuums. Gibt den Wert -0.022 zurück.
C_2	0	Konstante des Anfangsindividuums. Gibt den Wert 100 zurück.
LR1	0	Streckensensor; Gibt den Wert folgender Rechnung zurück: $track[8]-track[10]$
SensorF	0	Streckensensor; Gibt den Wert des Sensors track[9] zurück.
SpeedX	0	Gibt die aktuelle Geschwindigkeit des Fahrzeugs entlang der Fahrtrichtung zurück.
Add	2	Gibt die Summe der beiden Parameter zurück.
Sub	2	Gibt den Wert der Differenz der beiden Parameter zurück.
Mul	2	Gibt das Produkt der beiden Parameter zurück.
Div	2	Gibt den Quotient der beiden Parameter zurück. (sichere Division durch Null)
Abs	1	Gibt den Absolutwert des Eingabeparameters zurück.
Sum	1	Summiert das Ergebnis seines Unterbaums über alle Zeitschritte auf und gibt die Summe zurück.
Last	1	Speichert den aktuellen Wert seines Unterbaumes und gibt den Wert aus dem letzten Zeitschritt zurück.

Tabelle 4.7: Menge elementarer Funktionen M_4 des Gas/Bremse Baumes

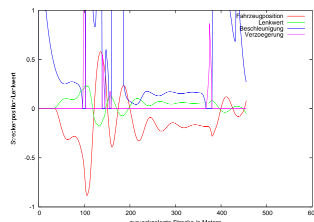


(a) zufällige Initialisierung
(b) mit vorgegebenem Startindividuum
(c) zufällige Initialisierung
(d) mit vorgegebenem Startindividuum
(e) zufällige Initialisierung
(f) mit vorgegebenem Startindividuum

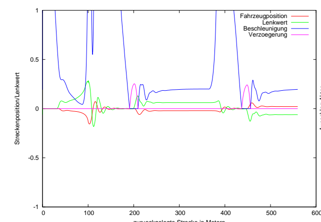
Abbildung 4.11: Fitness des jeweils besten Individuums und die mittlere Fitness der Generation unter Verwendung der Mengen elementarer Funktionen M_3 und M_4 und verschiedener Startseeds

den kleineren Mengen M_1 und M_2. Die Evolution kann die erweiterten Möglichkeiten hier nicht umsetzen. Um einen funktionierenden PD-, PI-, oder PID-Regler zu entwickeln bedarf es der Verwendung der Knoten *Sum* und *Old* in Verbindung mit den jeweiligen Regelabweichungen. Zeitgleich müssen die Parameter K_p, K_i und K_d korrekt gefunden werden um ein brauchbares Regelverhalten zu bekommen. Dies ist jedoch sehr unwahrscheinlich und mit der begrenzten Populationsgröße nicht zu erwarten. Damit lässt sich auch erklären, warum die Individuen in den Versuchen mit den neuen Knoten im allgemeinen schlechter abschneiden als in jenen ohne diese Knoten. Der zufällige Einbau der neuen Knoten führt eher zu einer Verschlechterung der Fahrleistung, da vor allem die Summenbildung von zufälligen Werten eher unkontrolliertes Verhalten hervorruft und nur in Verbindung mit der richtigen Stellgröße und dem korrekten Parameter zu einer Verbesserung führt.

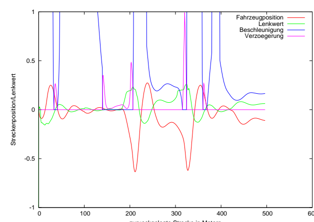
Dementsprechend zeigt sich in Abbildungen 4.12 und 4.13 zwar eine Verbesserung der Fahrleistungen von Generation Null zu Generation 49. Jedoch ist deutlich zu erkennen, dass die Aktorausschläge bei den Versuchen mit Mengen M_3 und M_4 in Generation 49 heftiger und unkontrollierter ausfallen als mit Mengen M_1 und M_2. Ohne Vorgabe eines Startindividuums gelingt es der Evolution kaum gute Fahrleistungen hervorzubringen (Abbildung 4.13 rechte Spalte). Die Funktionalität des Bremsens wird hier nicht entwickelt. Die Individuen beschleunigen konstant und verlassen so häufig die Strecke. Auch die Vorgabe des Startindividuums aus Abbildung 4.7 führt zu keiner Verbesserung gegenüber den Versuchen mit Mengen M_1 und M_2.



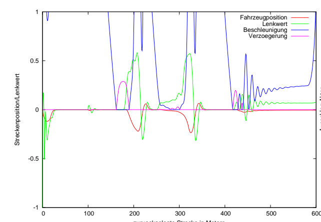
(a) Generation 0 auf Strecke 1



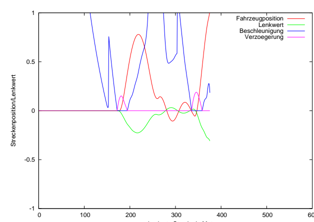
(b) Generation 49 auf Strecke 1



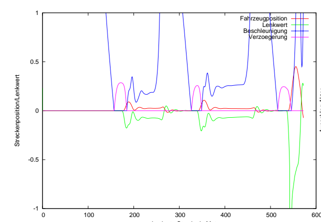
(c) Generation 0 auf Strecke 2



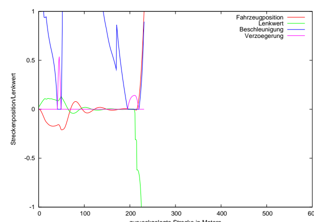
(d) Generation 49 auf Strecke 2



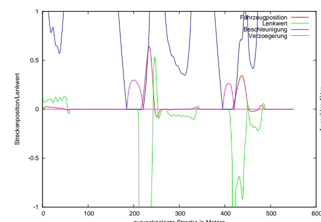
(e) Generation 0 auf Strecke 3



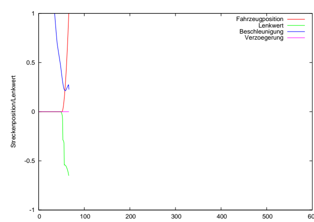
(f) Generation 49 auf Strecke 3



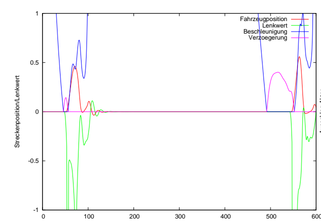
(g) Generation 0 auf Strecke 4



(h) Generation 49 auf Strecke 4

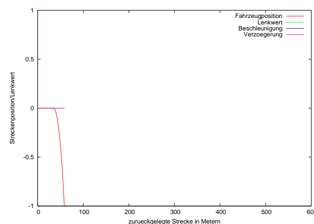


(i) Generation 0 auf Strecke 5

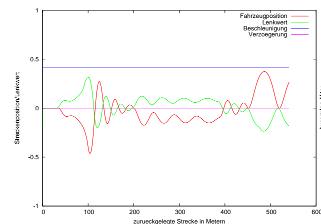


(j) Generation 49 auf Strecke 5

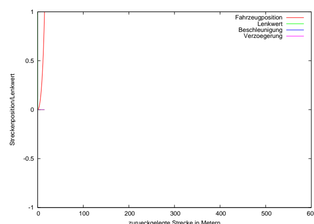
Abbildung 4.12: Vergleich der Fahrfunktionen für Generation 0 und Generation 49 mit Mengen elementarer Funktionen M_3 und M_4 und vorgegebenem Startindividuum aus Abbildung 4.7



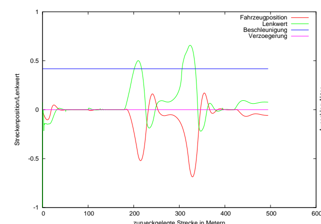
(a) Generation 0 auf Strecke 1



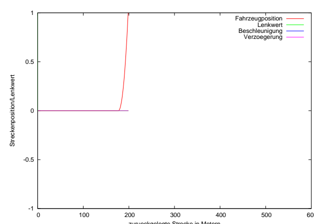
(b) Generation 49 auf Strecke 1



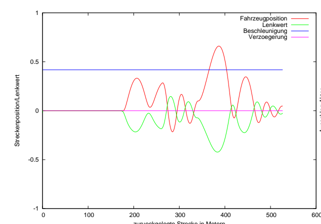
(c) Generation 0 auf Strecke 2



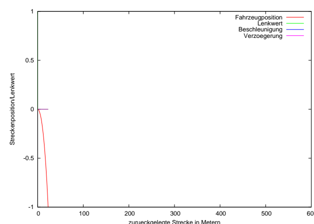
(d) Generation 49 auf Strecke 2



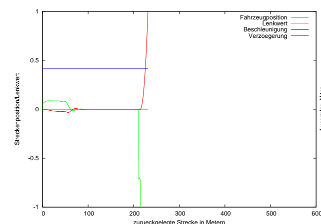
(e) Generation 0 auf Strecke 3



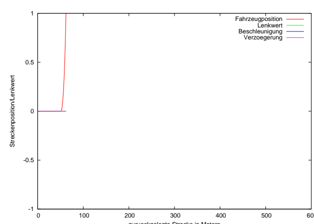
(f) Generation 49 auf Strecke 3



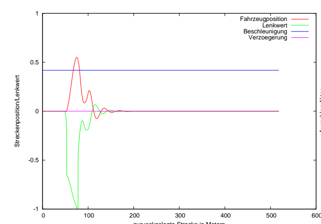
(g) Generation 0 auf Strecke 4



(h) Generation 49 auf Strecke 4



(i) Generation 0 auf Strecke 5



(j) Generation 49 auf Strecke 5

Abbildung 4.13: Vergleich der Fahrfunktionen für Generation 0 und Generation 49 mit Mengen elementarer Funktionen M_3 und M_4 ohne vorgegebenes Startindividuum

5 Zusammenfassung

Die Versuche dieser Arbeit zeigen eine prinzipiell stattfindende Evolution der Steuerung. Der Versuch zur Entwicklung einer Lenkung zeigte aufgrund schnell entwickelter guter Programme eine gute Machbarkeit der Aufgabenstellung. In den Versuchen zur Entwicklung aller Fahrfunktionen auf einer Strecke wurde der Evolution ein breites Spektrum an Funktionen bereitgestellt, um generelle Fahrfunktionen zu entwickeln. Es zeigte sich, dass die entstehenden Programme zwar in der Lage waren, die gegebenen einfachen Strecken zu umrunden, nutzten dabei aber nicht das volle Potential der Funktionen aus. Die Entwicklung von Programmen, die neben der Lenkung sowohl Beschleunigen als auch Verzögern beherrschen blieb aus. Stattdessen brachte die Evolution nur Steuerungen hervor, die das Fahrzeug auf eine Geschwindigkeit beschleunigten, welche für keine der auf der Teststrecke vorliegenden Kurven zu hoch war. Es konnte keine Reaktion auf nahende Kurven festgestellt werden. Dieser Ansatz wurde daraufhin verworfen und in einem zweiten Ansatz mit einer anderen Methode an das Problem herangegangen.
Anstatt der Evolution ein möglichst breites Suchfeld zu geben, wurden die Möglichkeiten limitiert um eine Evolution in einem lokalen Optimum zu forcieren. Dies zeigte gute Ergebnisse. Ausgehend von einem Startindividuum verfeinerte die Evolution die vorgegebenen Steuerungselemente und verbesserte die allgemeine Fahrtüchtigkeit der Programme.
Das vorgegebene Individuum reagierte bereits auf nahende Kurven, in dem es die momentane Geschwindigkeit mit einer errechneten Maximalgeschwindigkeit für den vor ihm liegenden Streckenabschnitt verglich. Die Lenkung des Fahrzeugs wurde durch einen einfachen P-Regler realisiert. Der evolutionäre Algorithmus fand sowohl für die Lenkung, als auch für die Gas/Bremse-Funktionalität kleine Verbesserungen im Code des Programms.
In wie weit diese Methode verallgemeinert werden kann muss Gegenstand weiterer Forschung sein. Es wurde gezeigt, dass prinzipiell eine Evolution eines Programms zur Steuerung eines Rennwagens erfolgt. Die Leistungen der entwickelten Steuerungen bleiben aber dennoch hinter den Erwartungen zurück. Um ein Fahrzeug im Renntempo um eine Rennstrecke bewegen zu können sind weitreichende Kenntnisse über dieses Strecke notwendig. Dem Programm muss explizit die Möglichkeit gegeben werden, zu errechnen wie schnell eine Kurve durchfahren werden kann. Darüberhinaus muss ein Verfahren entwickelt werden, welches es erlaubt anstatt der Mitte der Strecke die Ideallinie der Strecke als Referenz zu verwenden. Dies war in der Kürze der Bearbeitungszeit und der Komplexität der gestellten Aufgabe in dieser Arbeit nicht möglich.

Literaturverzeichnis

[ATL07] AGAPITOS, ALEXANDROS, JULIAN TOGELIUS und SIMON MARK LUCAS: *Evolving controllers for simulated car racing using object oriented genetic programming.* In: THIERENS, DIRK, HANS-GEORG BEYER, JOSH BONGARD, JURGEN BRANKE, JOHN ANDREW CLARK, DAVE CLIFF, CLARE BATES CONGDON, KALYANMOY DEB, BENJAMIN DOERR, TIM KOVACS, SANJEEV KUMAR, JULIAN F. MILLER, JASON MOORE, FRANK NEUMANN, MARTIN PELIKAN, RICCARDO POLI, KUMARA SASTRY, KENNETH OWEN STANLEY, THOMAS STUTZLE, RICHARD A WATSON und INGO WEGENER (Herausgeber): *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, Band 2, Seiten 1543–1550, London, 7-11 Juli 2007. ACM Press.

[BNKF98] BANZHAF, WOLFGANG, PETER NORDIN, ROBERT E. KELLER und FRANK D. FRANCONE: *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann, San Francisco, CA, USA, Januar 1998.

[Bri09] BRIEGLEB, VOLKER: *Games Convention setzt auf Online-Spiele.* Online, heise.de, Januar 2009.

[Dan08] DANIELLE: *Softwareanleitung des Wettbewerbspakets.* Online, April 2008.

[Ebn05] EBNER, MARC: *Evolutionäre Algorithmen.* Vorlesungsskript Uni-Würzburg, 2005.

[Koz92] KOZA, JOHN R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[Koz94] KOZA, JOHN R.: *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts, Mai 1994.

[Luk02] LUKE, SEAN: *ECJ-Hilfe.* Anleitung, Januar 2002.

[RN-] *Regelungstechnik - RN-Wissen.* http://www.rn-wissen.de/index.php/Regelungstechnik.

[SC96] SIEGEL, ERIC V. und ALEXANDER D. CHAFFEE: *Genetically Optimizing the Speed of Programs Evolved to Play Tetris.* In: ANGELINE, PETER J. und K. E. KINNEAR, JR. (Herausgeber): *Advances in Genetic Programming 2*, Kapitel 14, Seiten 279–298. MIT Press, Cambridge, MA, USA, 1996.

[uRS08] RALF SANDER, GERD BLANK UND: *Videospiele auf der Überholspur.* Online, Stern.de, Mai 2008.