Berner Fachhochschule
Technik und Informatik

Bachelor Thesis 2013

Fachbereich Informatik

Intelligente Fahrzeugsteuerung mit TORCS

Pflichtenheft

Studierende: Friedli Michael

Professoren: Dr. Jürgen Eckerle

Experte: Dr. Federico Flueckiger

Datum: 11. April 2013

Dieses Dokument beschreibt die Ziele und die Ausgangslage der Bachelor Thesis.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Zweck des Dokuments

Dieses Dokument dient der Definition der Ziele für die Bachelor Thesis 'Intelligente Fahrzeugsteuerung mit TORCS'. Es soll ein Bot entwickelt werden, der im OpenSource Rennspiel 'TORCS The Open Racing Car Simulation'[1] ein Rennen bestreiten und gewinnen kann. Umgesetzt wird das ganze mit dem Framework des Wettbewerbs 'The Simulated Car Racing Championship' [2], welcher eine Java-Umgebung für die Implementierung von selbst programmierten Bots anbietet. Aufgrund der Erkenntnisse aus dem Modul 'Projekt 2' konzentrieren sich die Ziele auf folgende Punkte:

- Finden einer geeignete Parametrisierung des Fahrermodells mit wenigen Freiheitsgraden
- Der Bot soll in der Lage sein, eine Strecke zu erlernen mit Hilfe einer Warmup-Phase
- Notwendige Verhaltensmuster wie ein Folge-Modell oder ein Überhol-Modell sind zu realisieren
- Das jeweils aussichtsreichste Verhaltensmuster auswählen

Jeder dieser Punkte wird im folgenden Kapitel ausführlicher beschrieben.

[1] http://torcs.sourceforge.net/
[2] http://games.ws.dei.polimi.it/competitions/scr/

2 Rahmenbedingungen

Dieses Kapitel beschreibt den aktuellen Zustand und die Voraussetzungen des Projekts. Dazu gehören die benötigten Applikationen sowie die Resultate aus dem Vorgängermodul 'Projekt 2'.

2.1 Projekt 2

Aus der Arbeit vom Modul 'Projekt 2' bestehen zwei fahrfähige Bots. Ein DirectControllerBot, der Aufgrund der Sensordaten die er erhält, direkt eine Aktion bestimmt und ausführt, und ein FuzzyControllerBot, der die Sensordaten per Fuzzy-Logic in eine Fahraktion umwandelt. Dabei zeigte sich, dass der Fuzzy-Logic Ansatz für einzelne Fahraktionen wie Steuern oder Beschleunigen nicht geeignet ist. Jedoch ist die verwendete Fuzzy-Logic Library geeignet, um in dieser Arbeit für andere Entscheidungsfindungen benutzt zu werden. Der DirectControllerBot kann als WarmUp-Bot für diese Arbeit verwendet werden, da er sicher durch alle Strecken fährt und so die Sensordaten für eine spätere Auswertung speichern kann.

2.2 TORCS

The Open Racing Car Simulation (TORCS) ist eine Rennsimulation, die im Jahr 2000 unter der GPL veröffentlicht wurde. Es wurde in C++ geschrieben und ist für GNU/Linux, FreeBSD, Mac OS X und Microsoft Windows verfügbar. Dank seiner offenen Struktur, der Modularität und der Erweiterbarkeit ist TORCS eine beliebte Basis für viele Forschungsprojekte im Bereich Fahrzeugroboter. Obwohl die TORCS in C++ implementiert ist, existieren Erweiterungen, die eine Bot-Programmierung in Java erlauben.

2.3 Simulated Car Racing Championship Umgebung

Der mehrmals jährlich stattfindende Wettbewerb 'The Simulated Car Racing Championship' (im Folgenden Competition benannt) der Universität Mailand bietet eine detaillierte Java-API für die Programmierung von Bots. Die Wettbewerbe finden im Rahem von grossen IEEE-Konferenzen (Institute of Electrical and Electronics Engineers) statt. Das Turnier besteht aus drei Phasen. Eine Warmup-Phase in der die Bots alleine auf der Strecke unterwegs sind. Eine Qualifying-Phase in der die Bots einige Runden alleine gegen die Zeit fahren. Und zum Schluss die Race-Phase, in der die Bots gegeneinander antreten. Die Startaufstellung ist von den gefahrenen Qualifying-Zeiten abhängig.

Architektur

Die von TORCS zur Verfügung gestellten Bots haben Zugriff auf alle Datenstrukturen des Spiels, da standardmässig keine Trennung zwischen Bot und Spiele-Engine besteht. Um Entwickler in einem fairen Wettbewerb gegeneinander antreten zu lassen, hat die Competition Umgebung klare Schnittstellen definiert, welche benutzt werden dürfen.

Die Anpassungen der Competition integrieren in TORCS neue wählbare Spieler. Diese kann man sich als Server in einer Client-Server-Architektur vorstellen. Die TORCS-Bots fungieren als Server und senden Sensordaten aus dem Spiel an den Java-Client. Der Client berechnet aus diesen Sensordaten Befehle und sendet diese zurück an den Server, welcher sie im Spiel ausführt. Der Server sendet die Daten in Echtzeit (ca. alle 20ms) an den Client, der als externes Programm läuft. Der Server wartet dann ca. 10ms auf eine Antwort des Bots. Falls kein Befehl in dieser Zeit zum Server gesendet wird, so wird die Simulation mit der als letztes erhaltenen Aktion fortgeführt.

Aufgrund der schlanken und übersichtlichen Architektur der Competition-Software und der implementierung in Java, wird der Bot dieser Arbeit mit diesem Framework entwickelt.

Die von der Competition gelieferten Klassen des Client weisen folgende Struktur auf.

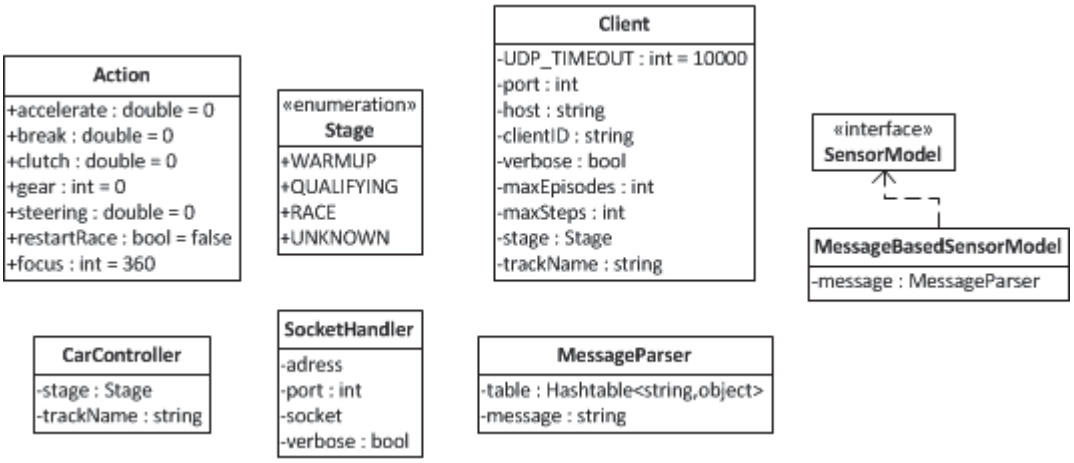

Abbildung 2.1: Architektur der wichtigsten Klassen

Die Klassen haben folgende Funktionen:

- Action: Enthält die Kontrollanweisungen, welche dem Bot gesendet werden.
- Stage: Enum zur Identifikation der verschiedenen Turnierphasen.
- Client: Baut die Verbindung zum Server auf und kommuniziert die Befehle.
- SensorModel: Interface für Sensordaten des Spiels.
- MessageBasedSensorModel: Implementation des SensorModels.
- CarController: Abstrakte Klasse, welche die grundlegenden Methoden des Bots enthält.
- SocketHandler: Öffnet und kontrolliert die Verbindung zum Server.
- MessageParser: Verarbeitet ein- und ausgehenden Daten.

Die wichtigste Klasse für die Bot-Programmierung ist der CarController. Die dort implementierte Methode *control(SensorModel sensors)* erstellt eine *Action*, welche alle Parameter der Fahrzeugsteuerung enthält.

Die Entwicklung des Bots tangiert hauptsächlich die Klassen CarController, da die Kommunikation zwischen Server und Client nicht verändert werden darf. Das Modell darf aber um eine unbegrenzte Anzahl an Klassen erweitert werden, welche den CarController unterstützen, eine Aktion zu berechnen.

3 Ziele

Die in der Einleitung erwähnten Ziele werden nachfolgend detailliert beschrieben und priorisiert.

3.1 Parametrisierung

Verschiedene Fahreigenschaften, wie beispielsweise die angestrebte Maximalgeschwindigkeit oder das Risikoverhalten, sollen so implementiert werden, dass sie als Parameter verändert werden können. Das Risikoverhalten hat einen Einfluss auf die Wahl des geeigneten Verhaltensmusters. So wird ein hohes Risikoverhalten eher zu Überholen führen, als ein tiefes Risikoverhalten. Zu den zu parametrisierenden Eigenschaften zählen wie schon erwähnt sicher die gewünschte Maximalgeschwindigkeit und das Risikoverhalten. Die Maximalgeschwindigkeit hat einen grosse Einfluss auf die Fahrweise. Je nach höhe der Geschwindigkeit ist eine andere Fahrweise erwünscht, um Kurven und Überholmanöver zu absolvieren. Das Risikoverhalten ist im Gegensatz zur Maximalgeschwindigkeit eine eher abstrakte Variable. Vorstellbar ist, dass die Wahl von verschiedenen Verhaltensmuster (Überholen, Folgen, Spurwechsel) per Fuzzy-Logic gesteuert wird. Das Risikoverhalten hätte dann einen Einfluss auf die Auswertung des Fuzzy-Controllers. So kann je nach Risikoverhalten gesteuert werden, unter welchen Umständen der Bot zum Überholen ansetzt. Das Risikoverhalten steuert aber auch Aspekte der einzelnen Muster, so würde ein Bot mit mehr Risikobereitschaft im Folge-Modell dem vorausfahrenden Fahrzeug näher auffahren als ein Bot mit wenig Risikobereitschaft.

3.2 Strecke lernen

Der RennBot soll in der Lage sein, in einer Warm-Up-Runde die Strecke zu erlernen. Dazu wird ein WarumpuBot benötigt, der ein Fahrverhalten aufweist, dass mit jeder Strecke umgehen kann. Ziel dieses Bots ist lediglich das saubere Fahren auf der Strecke, ohne dabei auf die Performance zu achten. Der aus dem Modul 'Projekt 2' erstellte Bot kann mit einigen Anpassungen zur Streckenspeicherung als WarmUpBot verwendet werden. Aufgrund der Sensor-Auswertung berechnet der WarmUpBot seine Fahraktion um die Strecke sauber abzufahren. Dabei ist das Ziel nicht, möglichst schnell zu sein, sondern die Strecke fehlerfrei abzufahren. Die Speicherung der Daten ergibt ein Abbild der Strecke und soll so helfen, ein realistisches Rennen zu fahren. Denn auch ein menschlicher Fahrer fährt eine Rennstrecke nicht aufgrund des Ist-Zustandes (Sensordaten), sondern trainiert mehrere Male und kann sich so merken, wann er beschleunigen und bremsen muss. Die Streckeninformationen werden in einem ersten Schritt so analysiert, dass eine Ideal-Linie berechnet wird. Danach sollen aufgrund dieser Linie eine oder mehrere Zusatzlinien berechnet werden, welche als Spuren zu betrachten sind. Die Diese Spuren dienen der Vereinfachung des Vorgehens bei verschiedenen Verhaltensmuster. So wird ein Überholmanöver so ablaufen, dass der Bot einem Gegner hinterher fährt (Folge-Modell), aufgrund der Umstände erkennt, dass er Überholen kann (Decisionmaking), die Spur wechselt (Spur-Modell) und den Gegner dann Überholt (Überhol-Modell).

3.2.1 Spuren-Modell

Der Einfachheit halber wird die Strecke in Spuren aufgeteilt, wie auf einer mehrspurigen Autobahn. Der Bot fährt immer auf einer Spur und wechselt diese je nach Rennsituation. Zu Beginn wird die Strecke in zwei oder drei Abschnitte unterteilt. Dabei gibt es zwei Möglichkeiten, diese Spuren umzusetzen. Die Linien können einerseits einfach parallele Linien sein, die sich linear nebeneinander befinden. Damit der Bot eine Kurve mit diesem Modell ideal befahren kann, muss er mehrere Spurwechsel während der Kurve vornehmen. Die andere Möglichkeit ist, dass die Hauptspur der Ideal-Linie folgt und die Zusatzlinien sind leicht abgeänderte Ideal-Linien. Somit könnte der Bot eine Strecke auf der Ideal-Linie befahren, ohne einmal die Spur wechseln zu müssen. Problematisch ist jedoch, dass sich die Spuren gerade in Kurven kreuzen würden.

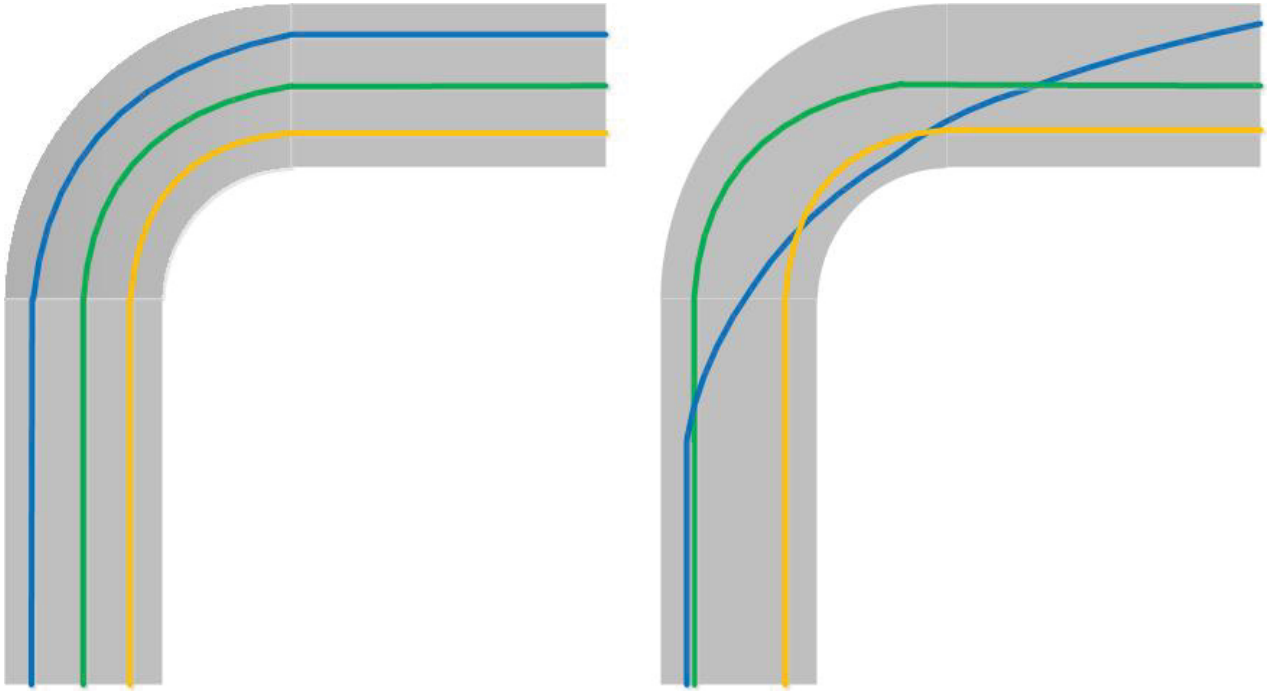

Abbildung 3.1: Links: Drei Spuren die parallel verlaufen; Rechts: Drei Spuren die sich an der Ideal-Linie (blau) orientieren

3.3 Verhaltensmuster

Um in einem Rennen zu bestehen, muss der Bot auf vordefinierte Verhaltensmuster zurückgreifen können. Vorläufig werden vier Muster in Betracht gezogen, ein Folge-Modell, ein Überhol-Modell, ein Spur-Modell und ein Spurwechsel-Modell. Das Folge-Modell steuert den Bot so, dass er einem vorausfahrenden Fahrzeug folgt und seine Aktionen diesem anpasst. Je nach Risikoverhalten ist der Abstand zwischen Bot und Leader grösser oder kleiner. Das Spur-Modell steuert den Bot in einer der vordefinierten Spuren. Der Bot folgt stur dieser Spur und wechselt diese nie. Das Spurwechsel-Modell gibt dem Bot den Befehl, die Spur zu wechseln. Das Überhol-Modell bildet den Überholvorgang ab. Es ist das komplexeste Modell, da es alle anderen Modelle benötigt. Der Bot folgt einem Leader (Folge-Modell) und analysiert stetig die Situation, um zu erkennen, ob ein Überholen möglich ist. Wenn er erkennt, dass eine Stelle den Überholvorgang zulässt, so wechselt er aus dem Folge-Modell in das Spur-Modell auf eine Spur links oder rechts neben dem Gegner und beschleunigt um diesen zu überholen. Während des ganzen Vorgangs muss der Bot auf Aktionen des Gegners reagieren können, falls dieser ebenfalls seine Position auf der Strecke ändert.

3.4 Decisionmaking

Damit der Bot in jeder Situation das geeignete Verhaltensmuster auswählt, benötigt er Kriterien um die einzelnen Muster zu beurteilen. Diese Kriterien setzen sich aus der Beschaffenheit der Strecke, den aktuellen Fahrparametern und den Gegnern auf der Strecke zusammen. Die Beschaffenheit der Strecke steuert primär die Fähigkeit, zu überholen. In einer Kurve zum Überholen anzusetzen ist in den meisten Fällen nicht erfolgsversprechend. Auch kann kein Überholen möglich sein, wenn der Bot bereits mit seiner maximalen Geschwindigkeit einem Gegner folgt. Die einzelnen Kriterien, welche eine Entscheidung beeinflussen, werden per Fuzzy-Logic evauliert und ausgewertet.

3.5 Vorgehen

Die verschiedenen Ziele spielen sehr komplex zusammen, sind einzeln jedoch nicht sehr effektiv. Das heisst, dass ein optimales Vorgehen alle Ziele im Auge behält und die jeweilige Umgebung angepasst werden muss. Daher wird

in einem ersten Schritt die einfachste Strecke behandelt, die Gerade. Danach wird die komplexität der Strecke und der Anforderungen sukzessiv gesteigert. Denkbar wäre eine kreisförmige oder ovale Strecke als Folgeschritt. Danach kann erst an eine Betrachtung von eigentlichen Rennstrecken gedacht werden.

3.6 Prioritäten

In der folgenden Tabelle werden die Prioritäten der einzelnen Aufgaben festgelegt:

Aufgabe	**Priorität**
Strecke lernen	hoch
Spurenmodell	hoch
Verhaltensmuster Spur	hoch
Verhaltensmuster Spurwechsel	hoch
Verhaltensmuster Folgen	mittel
Verhaltensmuster Überholen	mittel
Decisionmaking	niedrig
Parametrisierung	niedrig

Tabelle 3.1: Aufgaben nach Priorität

Die Überlegung hinter dieser Priorisierung liegt in dem übergeordneten Ziel, ein Rennen zu bestreiten. Dazu ist vorallem eine Kenntnis der Strecke erforderlich, um das Fahrverhalten den Streckenverhältnissen optimal anzupassen. Das Fahrverhalten soll dazu das Spurenmodell beachten, was die Verhaltensmuster Spur und Spurwechsel ebenfalls als höherwertig qualifiziert. Sobald diese Anforderungen erfüllt sind, kann der Bot zumindest ein einfaches Rennen gegen einen anderen Bot oder einen Menschen bestreiten, deshalb sind diese Anforderungen mit einer hohen Priorität versehen.

Dabei wird er die Strecke noch ohne Rücksicht auf den Gegner befahren. Den Gegner beachten und auf ihn reagieren ist jedoch ein elementarer Teil für einen Rennbot. Daher sind die Verhaltensmuster Folgen und Überholen von mittlerer Priorität.

Die Anforderungen Decisionmaking und Parametrisierung des Fahrverhaltens sind ebenfalls wichtig für einen ausgereiften Rennbot, jedoch bauen sie einerseits auf all den vorangegangenen Anforderungen auf und sind andererseits für einen guten Rennbot das Tüpfelchen auf dem i. Aus diesem Grund sind sie von niedriger Priorität.

3.6.1 Minimalziel

Als Minimalziel soll ein Bot entwickelt werden, der eine Strecke speichern und auswerten kann. Aufgrund dieser Auswertung soll der Bot mit Hilfe des Spurenmodells die Strecke optimal befahren und so eine gute Rennzeit erreichen. Dazu soll der Bot in der Lage sein, einen gegnerischen Bot im einfachsten Fall überholen können. Der einfachste Fall ist definiert, als eine Situation, in der der gegnerische Bot mit konstanter Geschwindigkeit und auf einer konstanten Spur fährt. Der Gegner vollführt also keine Rennfahrer-Manöver, um ein Überholen zu erschweren oder zu verunmöglichen.