TECHNISCHE HOCHSCHULE MITTELHESSEN

Fachbereich
Mathematik, Naturwissenschaften und Informatik

Ausarbeitung Projektvortrag Fuzzy-Regelung

Im Rahmen der Veranstaltung:

Praktikum Künstliche Intelligenz(CS5330)

Autoren:

Joel Bartelheimer
joel.bartelheimer@mni.thm.de

Nico Müller
nico.mueller@mni.thm.de

Eingereicht bei:

Prof. Dr. Wolfgang Henrich

Abgabedatum:
24.04.2017

II Inhaltsverzeichnis

II Abbildungsverzeichnis

III Tabellenverzeichnis

IV Listingverzeichnis

Zusammenfassung

Die vorliegende Hausarbeit bearbeitet eine Problemstellung des autonomen Fahrens. Konkret soll ein durch einen Fuzzy-Regler gesteuertes Autos einem anderen Auto im optimalen Abstand folgen. Zunächst werden die physikalischen Bedingungen für die beiden Autos erläutert. Anschließen wird darauf eingegangen wie man mit Hilfe von numerischer Integration die tatsächliche Geschwindigkeit bzw. Bewegung eines Fahrzeugs simulieren kann. Der eigentlich Fuzzy-Regler wurde mithilfe von funktionaler Programmierung implementiert. Zuletzt wird die GUI sowie das Zusammenspiel zwischen den Modellen und der GUI vorgestellt.

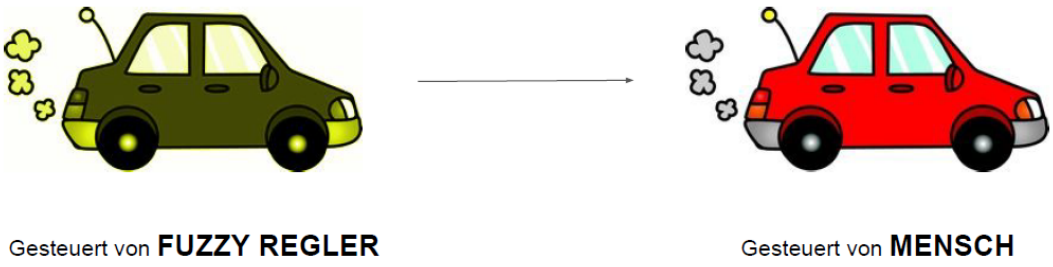

Abbildung 1: Problemstellung

1 Problemstellung

Die gestellte Aufgabe zur praktischen Ausarbeitung kommt aus dem Themengebiet des autonomen Fahrens. Es soll ein Fuzzy-Regler eingesetzt werden um ein Fahrzeug so zu steuern, dass es einem anderen, durch den Menschen gesteuerten, Fahrzeug folgt. Hierbei ist zu beachten, dass hier bei nur die Geschwindigkeit bzw. Bewegung in einer Achse kontrolliert wird. Das bedeutet, dass nur das Gas- sowie das Bremspedal jedoch nicht das Lenkrad geregelt werden muss. Außerdem gehen wir davon aus, dass es sich bei dem gesteuerten Fahrzeug um ein Fahrzeug mit Automatikgetriebe handelt und somit das Kuppeln sowie das Schalten der Gänge ebenfalls irrelevant ist.

Bei der Implementierung sollte darauf geachtet werden, dass beide Fahrzeuge, so weit wir möglich, den physikalischen Gesetzen unterliegen. Dies bedeutet, dass z.B. die Fahrwiderstände wie Luftwiderstand, Rollwiderstand usw. beachtet werden müssen. Ebenfalls sollten beide Fahrzeuge durch eine physikalische Spezifikation, wie z.B. das Gewicht, definiert werden. Beide Fahrzeuge sollten hierbei der gleichen physikalische Spezifikation unterliegen, um keines der beiden Fahrzeuge einen Vorteil zu bieten.

2 Implementierung

In diesem Kapitel wird eine Lösung für das gestellte Problem vorgestellt. Wir empfehlen dem Leser zuerst die Ausarbeitung zum theoretischen Teil der Fuzzy-Regelung zu lesen, da hier keine Grundlagen erläutert werden.

2.1 Funktionale Programmierung

Als funktionale Programmierung bezeichnet man ein Programmierparadigma in dem alle Programme lediglich aus Funktionen bestehen. Hierbei können neue Funktionen aus bereits bestehenden Funktionen zusammengesetzt werden. Ebenfalls werden Funktionen in der funktionalen Programmierung als Wert angesehen. Dies erlaubt es Funktionen als Parameter oder Rückgabe zu verwenden, dies nennt man „Funktion höhrer Ordnung“. In

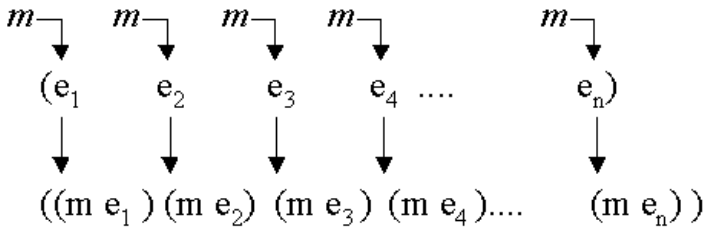

Abbildung 2: Veranschaulichung Map Funktion

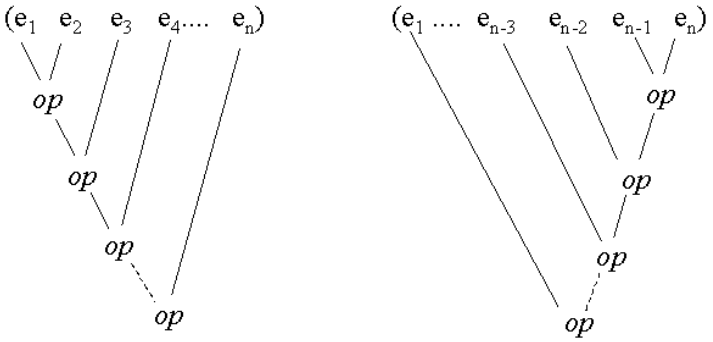

Abbildung 3: Veranschaulichung Reduce Funktion

sogenannten „reinen“ funktionalen Programmiersprachen gibt es keine Variablen womit Seiteneffekte vollkommen ausgeschlossen werden. Für unsere Implementierung wird die Sprache Scala[1] verwendet. Scala ist eine JVM-Sprache die funktionale Programmierung und Objektorientierte Programmierung verbindet. Wir haben diese Sprache gewählt, da wir einerseits unsere Kenntnisse in der Sprache durch das Projekt erweitern wollten aber anderseits auch der Meinung waren, dass sich eine Fuzzy-Regelung einfacher mithilfe von Funktionen höherer Ordnung ausdrücken lässt.

2.1.1 Map, Reduce, Filter

Die drei Funktionen map, filter und fold sind essentiell um die funktionale Programmierung zu verstehen. `map` wendet eine Funktion auf jedes Element einer Liste an und kreiert somit eine neue Liste (siehe Abbildung 2). Die Funktion `filter` entfernt genau die Element einer List, die eine mit einer Funktion verbunden ein boolesches `false` liefern (siehe Abbildung 4). Die Funktion `fold` oder auch `reduce` wendet eine Funktion mit zwei Parametern solange auf Paare der Liste an, bis nur noch ein Element in der Liste enthalten ist (siehe Abbildung 3).

[1] https://www.scala-lang.org/

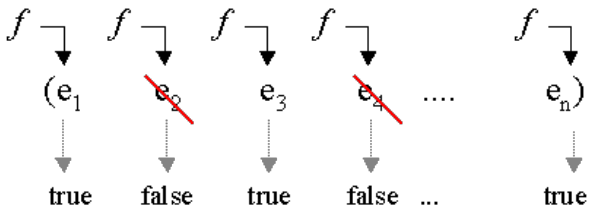

Abbildung 4: Veranschaulichung Filter Funktion

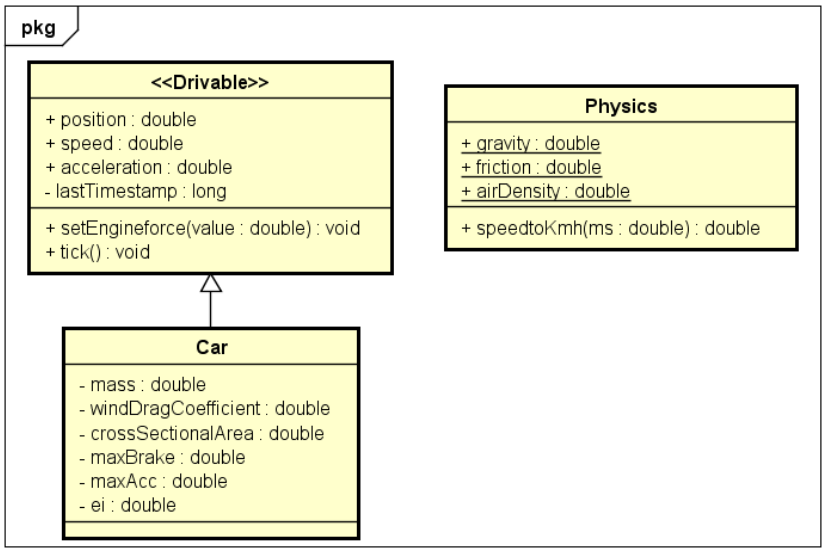

Abbildung 5: UML der Klasse Auto und deren Nachbarklassen.

2.2 Modell des Fahrzeugs

Die Modelle der beiden Fahrzeuge werden in der Klasse „Car“ realisiert (siehe Abbildung 5). Jedes erstellte Fahrzeug hat ein vorgegebenes Gewicht in kg, ein Strömungswiderstandskoeffizient (dimensionslos) sowie eine gegebene Fläche der projizierte Stirnfläche in m^2. Ebenfalls wird eine maximale Brems bzw. Antriebskraft festgelegt. Wir haben für unser Modell diese Werte von realen Fahrzeugen übernommen und uns auf die Werte aus Tabelle 1 festgelegt.

Zusätzlich wird für jedes Objekt des Typs „Car“ die aktuelle Position relativ zum Null Punkt in m, die aktuelle Geschwindigkeit in m/s sowie die aktuelle Beschleunigung in m/s^2 dokumentiert. Der Nutzer der Klasse hat die Möglichkeit die Brems bzw. Antriebskraft in N für das Auto über ein Interface zu setzen.

2.2.1 Bestimmung der Beschleunigung

Um die gesetzte Brems bzw. Antriebskraft, unter Berücksichtigung der physikalischen Gesetze, in eine Beschleunigung umzurechnen, betrachten wir zunächst die Fahrwiderstände

Tabelle 1: Physikalische Spezifikation der Fahrzeuge

Eigenschaft	Wert	Einheit	Bezeichner
Gewicht	1540	kG	m_{Fzg}
Zuladung	0	kG	m_{Zu}
Strömungswiderstandskoeffizient	0.67	-	cw
Projizierte Stirnfläche	1.86	m^2	A
Maximale Bremskraft	8000	N	$\overline{F}_{max}$
Maximale Antriebskraft	3000	N	F_{max}
Massefaktor	1	-	e_i

Tabelle 2: Physikalische Spezifikation der Umwelt

Eigenschaft	Wert	Einheit	Bezeichner
Erdanziehungskraft	9,81	m/s^2	g
Zuladung	1.2	kG/m^3	p_{Luft}
Strömungswiderstandskoeffizient	0.01	-	f_{roll}

die ein Fahrzeug überwinden muss. Als Fahrwiderstand F_{FW} wird die Summe der Widerstände bezeichnet, die ein Fahrzeug mit Hilfe einer Antriebskraft überwinden muss, um das Fahrzeug zu bewegen. Dazu gehört der Luftwiderstand F_{Luft}, der Rollwiderstand F_{Roll}, der Steigungswiderstand F_{Steig} sowie der Beschleunigungswiderstand F_B.

Luftwiderstand Der Luftwiderstand ist maßgeblich für die Höchstgeschwindigkeit eines Fahrzeugs. Er hängt von der aerodynamischen Form des Fahrzeuges ab und steigt mit der Geschwindigkeit ins Quadrat. Außerdem spielt die momentane Luftdichte p_{Luft} in kg/m^3 eine Rolle für den Luftwiderstand. Diese ist abhängig von der Höhe in der Atmosphäre sowie der Temperatur der Atmosphäre. Wir haben für diese Umgebungsvariable einen Wert von $1,2$ gewählt, dies entspricht der Luftdichte bei einer Temperatur von 20°C auf Höhe des Meeresspiegels.

$$F_{Luft} = cw \cdot A \cdot \frac{p_{Luft} \cdot v_{Fzg}^2}{2} \tag{1}$$

Rollwiderstand Der Rollwiderstand beschreibt die nötige Arbeit um einen Reifen auf einer Kontaktfläche zu rollen. Durch die Fahrzeugmaße und das elastische Material des Reifens sowie dem Boden werden diese zusammengedrückt und bilden keine optimale Kreisform mehr. Diese Eigenschaft wird durch den Rollwiderstandskoeffizient f_{Roll} abstrahiert. Wir haben für diese Umgebungsvariable einen Wert von $0,01$ gewählt was einem

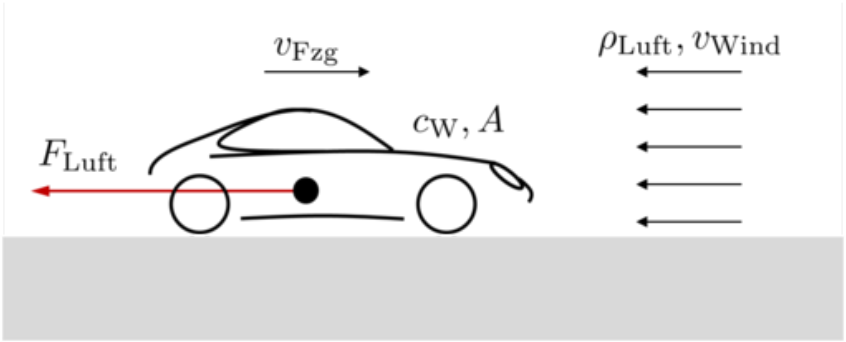

Abbildung 6: Prinzipskizze zum Luftwiderstand an Fahrzeugen (Simon Herzog (https://commons.wikimedia.org/wiki/File:Luftwiderstand.png)

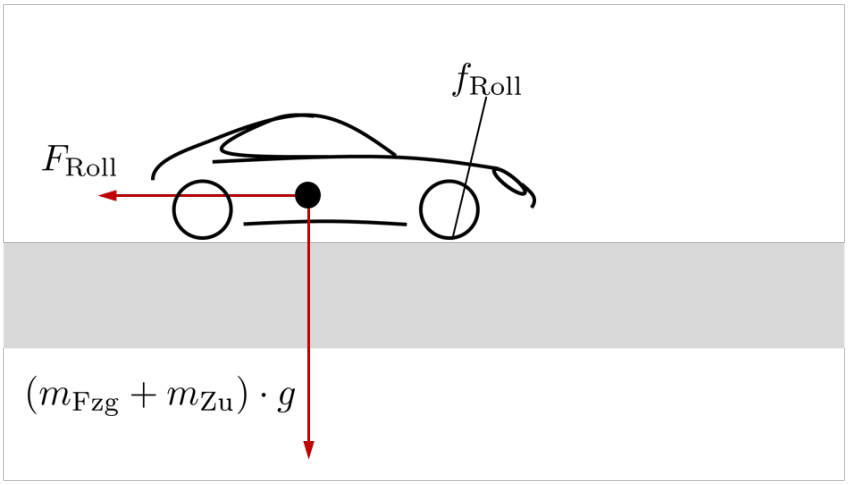

Abbildung 7: Prinzipskizze zum Rollwiderstand an Fahrzeugen (Simon Herzog (https://commons.wikimedia.org/wiki/File:Rollwiderstand.png)

Autoreifen auf einem Asphaltboden entspricht. In die Formel spielt außerdem die Erdbeschleunigung g sowie der Steigungswinkel a eine Rolle. Wenn der Steigungswinkel steigt oder sinkt, also $|a| \gg 0$, konvergiert der Rollwiderstand gegen 0.

$$F_{Roll} = f_{Roll} \cdot (m_{Fzg} + m_{Zu}) \cdot g \cdot cos(a) \tag{2}$$

Steigungswiderstand Der Steigungswiderstand beschreibt die nötige Kraft die benötigt wird um die Maße des Fahrzeugs über die Steigung zu bewegen. Falls die Steigung negativ ist, wird der Steigungswiderstand negativ und beeinflusst somit das Fahrzeug positiv. Der Steigungswiderstand spielt in unserer Aufgabenstellung z.Z. keine Rolle, da von einer ebenen Strecke ausgegangen werden soll. Dennoch wurde dieser im Sinne der Vollständigkeit implementiert und hier aufgeführt.

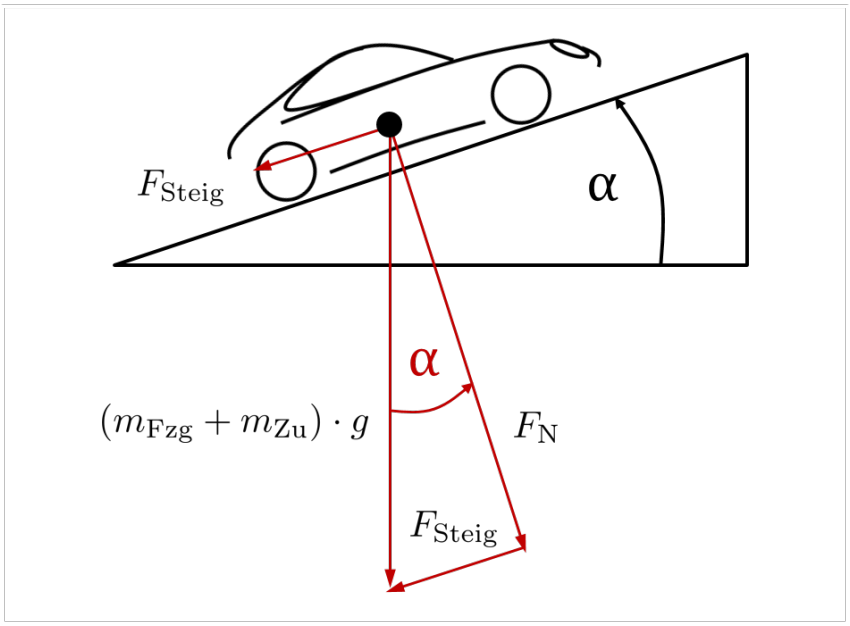

Abbildung 8: Prinzipskizze zum Steigungswiderstand an Fahrzeugen (Simon Herzog (https://commons.wikimedia.org/wiki/File:Steigungswiderstand.png)

$$F_{Steig} = (m_{Fzg} + m_{Zu}) \cdot g \cdot sin(a) \tag{3}$$

Beschleunigungswiderstand Der Beschleunigungswiderstand beschreibt die nötige Kraft die benötigt wird um die Beschleunigung eines Fahrzeugs zu ändern. Dieser Widerstand entsteht durch die Trägheit des Fahrzeugs. Der Massefaktor e_i (≥ 1) fasst den translatorischer Anteil, also Maße die sich in Fahrtrichtung bewegt sowie den rotatorischer Anteil, also alle rotierende Teile im Antriebsstrang (Motor, Kurbelwelle, Kupplung, Räder) zusammen. Der Massefaktor ist somit auch stark abhängig vom gewählten Gang des Fahrzeugs. In unserem Modell ist der Massefaktor auf 1 gesetzt.

$$F_B = (e_i \cdot m_{Fzg} + m_{Zu}) \cdot a \tag{4}$$

$$F_{FW} = F_{Luft} + F_{Roll} + F_{Steig} + F_B \tag{5}$$

Um nun die tatsächliche Kraft zu erhalten müssen die Fahrwiderstände von der Antriebskraft subtrahiert werden. Dieser Kraft, definiert als $kg * m/s^2$, kann nun in die aktuelle Beschleunigung umgerechnet werden.

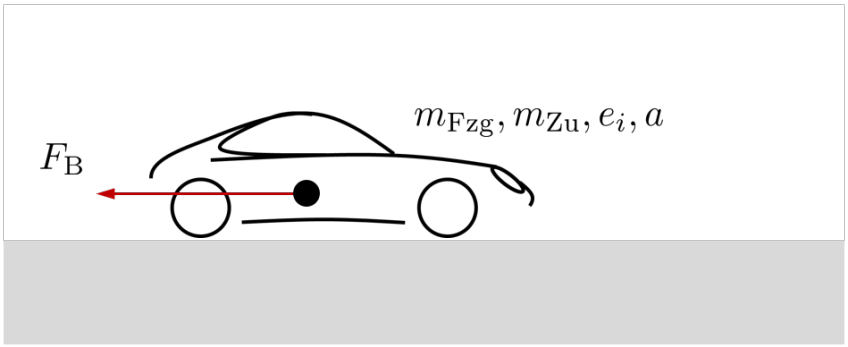

Abbildung 9: Prinzipskizze zum Beschleunigungswiderstand an Fahrzeugen (Simon Herzog (https://commons.wikimedia.org/wiki/File:Beschleunigungswiderstand.png)

$$a = \frac{F - F_{FW}}{m_{Fzg} + m_{Zu}} \tag{6}$$

2.2.2 Bestimmung der Position

Die Position eines jeden Fahrzeugs lässt sich über die Integration der Beschleunigung errechnen. Durch Ableitung der Position (siehe Gl. 1) ergibt sich die Geschwindigkeit (siehe Gl. 2). Durch ein weiteres Ableiten erhält man die Beschleunigung (siehe Gl. 3). Da unser System zunächst nur die Beschleunigung bereitsteht, lässt sich durch die Integration auf die Geschwindigkeit (siehe Gl. 4) sowie in einem weiteren Schritt auf die Position schließen (siehe Gl. 5).

$$s(t) = \text{Position zur Zeit t} \tag{7}$$

$$\dot{s}(t) = v(t) = \text{Geschwindigkeit zur Zeit t} \tag{8}$$

$$\ddot{s}(t) = \dot{v}(t) = a(t) = \text{Beschleinung zur Zeit t} \tag{9}$$

$$\begin{aligned} \dot{v}(t) &= a(t) \\ \int_0^t \dot{v}(t)dt &= \int_0^t a(t)dt \\ v(t)\Big|_0^t &= a(t) \cdot t\Big|_0^t \\ v(t) - v(0) &= a(t) \cdot t - a(0) \cdot 0 \\ v(t) &= a(t) \cdot t + v(0) \end{aligned} \tag{10}$$

$$
\begin{aligned}
\dot{s}(t) &= v(t) \\
\int_0^t \dot{s}(t)dt &= \int_0^t v(t)dt \\
\int_0^t \dot{s}(t)dt &= \int_0^t a(t) \cdot t + v(0)dt \\
s(t)\Big|_0^t &= 0,5 \cdot a(t) \cdot t^2 + v(t) \cdot t\Big|_0^t \\
s(t) - s(0) &= 0,5 \cdot a(t) \cdot t^2 + v(t) \cdot t \\
s(t) &= 0,5 \cdot a(t) \cdot t^2 + v(t) \cdot t + s(0)
\end{aligned}
\tag{11}
$$

Numerische Integration Die numerische Integration in unserem System wird durch eine kontinuierliche, schrittweise Integration umgesetzt. Hierfür muss in zyklischen Abständen die Beschleunigung gemessen werden und dann die draus resultierende Geschwindigkeit und Position berechnet und aktualisiert werden. Wichtig ist hierbei, dass die Zeit zwischen diesen zyklischen Aufrufen festgehalten wird.

Im Code Beispiel (siehe Listing 1) wird in der Funktion `tick()` die numerische Integration implementiert. Die Variable `past` enthält einen Zeitstempel der gespeichert wurde, als die Funktion das letzte mal aufgerufen wurde. Wichtig ist hierbei, dass der Sichtbarkeitsbereich der Variablen außerhalb der Funktion liegt. So kann bei jedem Aufruf die vergangene Zeit zwischen dem aktuellen und vorherigen Aufruf bestimmt werden. Nach Abschluss der Berechnung wird Zeitstempel überschrieben.

```
1   var acceleration: Double = 0
2   var speed: Double = 0
3   var position: Double = 0
4   var past: Long = System.currentTimeMillis()
5
6   def tick(): Unit = {
7     // Bestimmen der aktuellen Zeit
8     val now = System.currentTimeMillis()
9     // Berechnung der vergangenen Zeit in Sekunden
10    val t: Double = (now - past) / 1000.0
11
12    // Erste Schritt der Integration
13    acceleration = (engineForce - antiForce) / mass
14    // Zweiter Schritt der Integration
15    speed += (acceleration * t)
16    // Dritter Schritt der Integration
```

```
17      position += ((0.5 * acceleration * t * t) + speed * t)
18
19      // Speichern der Aufrufszeit
20      past = now;
21    }
```

Listing 1: Numerische Integration

2.3 Modell der Wissensbasis

Die Wissensbasis besteht aus der Regelbasis und der Datenbasis. Alle Wertebereiche für die Eingangsgrößen und Stellgrößen sowie die dazugehörigen linguistischen Termen und dazu assoziierten Fuzzy-Mengen gehören zu der Datenbasis. Die Regelbasis besteht aus den linguistischen Kontrollregeln. Die Wissensbasis soll durch den Benutzer zur Laufzeit des Programms konfigurierbar sein. Aus diesem Grund wurden alle nötigen Parameter in ein Konfigurationsobjekt `FuzzyConfig` gebettet das zur jeder Zeit vom Fuzzy-Regler geladen werden kann. Um das Verständnis zu verbessern werden nun die einzelnen Komponenten der Wissensbasis und deren dazugehörigen Modellierungen vorgestellt.

Eingangs- Ausgangsgrößen Für jede messbare Eingangsgröße und jede Ausgangsgröße muss ein Wertebereich und ein Bezeichner festgelegt werden, sodass andere Programmteile diese wieder erkennen können. Diese Größen nennen wir `FuzzyValueConnector`.

Fuzzy-Menge und Linguistischer Term Da ein linguistischer Term immer nur einer Fuzzy-Menge zugeordnet sein sollte, wurden diese Elemente in unserem Model zusammen in der Klasse `FuzzyTerm` implementiert. Die eigentliche Zugehörigkeitsfunktion wird durch die Funktion höherer Ordnung `(Double => FuzzyBool)` ausgedrückt. Wobei `FuzzyBool` nur ein Fließkommazahl in den Grenzen $[0, 1]$ ist.

Fuzzy-Regeln Die Regelbasis in Form von linguistischen "Wenn - Dann - Regeln"wird durch die Klasse `FuzzyRule` modelliert. Jede Regel wird durch ihren Namen identifiziert und enthält eine List von `FuzzyTerm` die die Eingangsgrößen darstellen. Die Verknüpfung dieser Eingangsgrößen wird durch eine Liste von Operatoren modelliert. Außerdem wird eine Ausgangsgröße, ebenfalls in Form eines `FuzzyTerm` festgelegt.

Defuzzyfizierungs-Methoden Die Aufgabe der Defuzzyfizierungs-Methoden ist es aus der Ausgangsfuzzymenge einen Stellwert auszuwählen. In unserer Modellierung arbeitet

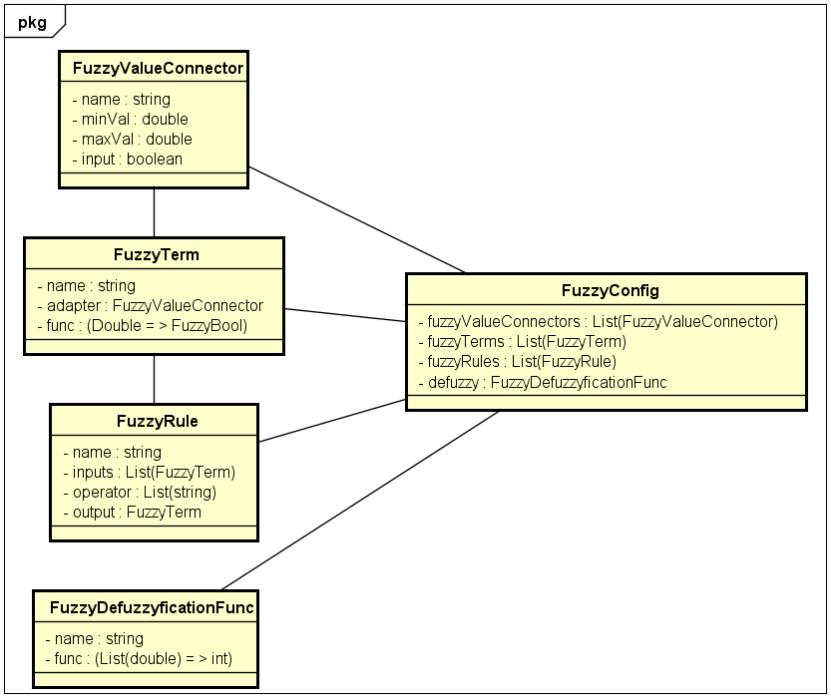

Abbildung 10: Modellierung der Wissensbasis

die Defuzzyfizierungs-Methoden mit einer Liste aller im Wertebereich enthaltenen Werte auf die die kombinierten Zugehörigkeitsfunktion aller Regeln bereits angewendet wurden. Sie liefert dann einen Stellwert in diesem Wertebereich zurück. Die Defuzzyfizierungs-Methoden wird durch die Funktion höherer Ordnung `List[Double] => Int)` beschrieben und durch einen Namen identifiziert und in der Klasse `FuzzyDefuzzyficationFunc` zusammengefasst.

2.4 Modell des Fuzzy-Reglers

Die eigentliche Fuzzy-Regelung findet in der Klasse `FuzzyCarController` statt. Die Klasse wird mit einer `FuzzyConfig` und zwei Objekten des Interfaces `Drivable` erzeugt. Eine Instanz stellt hierbei das Verfolger- und eins das Vorausfahrende Fahrzeug dar. Es wird eine Funktion `tick()` bereitgestellt. Diese muss vom Nutzer der Klasse zyklisch aufgerufen werden um einen Berechnungsschritt auszuführen. Jeder Berechnungsschritt durchläuft vier Phasen.

2.4.1 Phasen

Simulieren In der Phase „Simulieren“ werden die `tick()` Funktionen der beiden Fahrzeuge aufgerufen und somit die neuen Positionen der Fahrzeuge bestimmt.

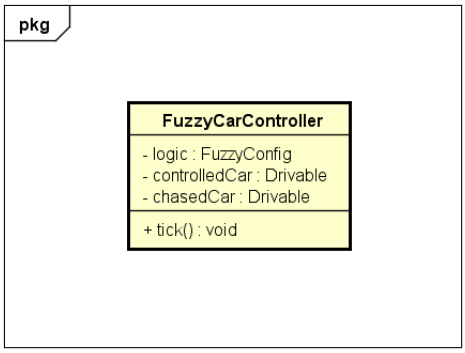

Abbildung 11: Interfaces des Fuzzy-Reglers

Messen In der Phase „Messen“ werden die aktuellen Werte der Fahrzeuge ausgelesen und eventuelle Berechnungen durchgeführt um weiter Eingangsvariablen für die Fuzzy-Regelung zu erzeugen. An dieser Stelle errechnen wir z.Z. nur den Abstand es wäre aber denkbar an dieser Stelle weitere Informationen auszulesen und diese für die Logik zu verwenden.

$$distance = chasedCar.position - controlledCar.position \tag{12}$$

Fuzzy-Regelung ausführen In dieser Phase werden die gemessenen Werte in die Regeln der Fuzzy-Regelung eingesetzt und die vier Schritte eines Mamdani Fuzzy-Reglers, Prämissenauswertung, Aktivierung, Aggregation und Defuzzyfizierung ausgeführt. Wie diese Teilschritte mithilfe von Funktionaler Programmierung umgesetzt wurden, wird im Abschnitt 2.4.1 beschrieben.

Stellwert setzen In dieser Phase wird der vom Defuzzyfizierungs-Interface bestimmte Wert an die jeweiligen `FuzzyValueConnector` übergeben. Unsere Implementierung benutzt jedoch nur eine Stellgröße und zwar die Antriebskraft für das Verfolgerfahrzeug.

2.4.2 Fuzzy-Regler nach Mamdani

In diesem Abschnitt werden die vier Schritte des Fuzzy-Reglers nach Mamdani wiederholt und deren Implementierung erläutert.

Prämissenauswertung Als erstes wird in der Berechnung die Prämissenauswertung für eine Regel durchgeführt. Hierfür ein Ausschnitt aus der theoretischen Ausarbeitung über die Fuzzy-Regelung:

> Für die Auswertung einer Regel wird zunächst der Akzeptanzgrad dieser Regel bestimmt. Dazu wird für jeden linguistischen Term der Prämisse der Zugehörigkeitsgrad für den kürzlich gemessene Messwert bestimmt. Also $\mu_v(x_v)$ mit $v = 1...n$. Da es in einer Regel mehrere Prämissen geben kann, die mit den Operatoren $[and, or]$ verknüpft werden können, müssen auch die Zugehörigkeitsgrade der n-verschiedenen Prämissenteilen durch eine geeignete Konjunktion, z.B. Minimum für *and*-Verknüpfungen, verknüpft werden.

Wir müssen also eine `FuzzyRule` betrachten und für jeden `FuzzyTerm` der Prämisse den richtigen Messwert auf die Zugehörigkeitsfunktion anwenden. In Zeile vier und fünf ist zu erkennen wie zwischen den verschiedenen `FuzzyValueConnector` unterschieden wird und je nach Fall die gemessene Distanz oder die aktuelle Geschwindigkeit in die Zugehörigkeitsfunktion eingesetzt wird. Nach der `map()` Funktion auf die `inputs` der `FuzzyRule` hat der Value `acceptance` den Typ `List[FuzzyBool]`, da die Zugehörigkeitsfunktion den Typ `(Double => FuzzyBool)` hat. In unserer Implementierung sind zum aktuellen Zeitpunkt nur *and*-Verknüpfungen möglich. Das bedeutet, dass die das Minimum der errechneten Akzeptanzgraden gewählt wird (siehe Zeile 9 in Listing 2). Durch die Funktion `foldLeft` in Zeile neun wird aus `List[FuzzyBool]` ein `Double` extrahiert welches das Minimum der Liste darstellt.

```
1 val acceptance = rule.inputs.map(
2   inputTerm => {
3     inputTerm.adapter.name match {
4       case "Distance" => inputTerm.func.apply(distance)
5       case "Speed" => inputTerm.func.apply(speed)
6     }
7   }
8 )
9 val minAcceptance = acceptance.foldLeft(Double.MaxValue)(_ min _.value)
```

Listing 2: Berechnung des Akzeptanzgrad

Aktivierung Der nächste Schritt in der Fuzzy-Regelung ist die Aktivierung. Die Aktivierung wird wie auch die Prämissenauswertung für jede Regel in der Regelbasis ausgeführt. Für die Inferenzmethode haben wir die MAX-MIN-Inferenz gewählt. Deren Verhalten wird jetzt aus der theoretischen Ausarbeitung zititert:

> Die Konklusion ergibt eine Fuzzy-Menge von Stellwerten. Um die Aktivierung der Konklusion zu bestimmen muss der linguistische Term mit dem Akzep-

tanzgrad der Prämisse verknüpft werden. Dazu wird die sogenannte Inferenzmethode gewählt. Bei der MAX-MIN-Inferenz wird die Ausgabe Fuzzy-Menge B an der Höhe des minimalen Akzeptanzgrad der Prämisse gekappt.

$$output^{R_r}_{x_1 \ldots x_n} : y \mapsto min(\mu_{1,r}(x_1), ..., \mu_{n,r}(x_n), \mu_r(y)) \tag{13}$$

Wir müssen also die Zugehörigkeitsfunktion des Ausgangs-`FuzzyTerm` um den, zuvor berechneten, Akzeptanzgrade der Prämisse kappen. Es wird also eine Funktion höherer Ordnung erstellt, die ein übergebenes`Double` auf die Zugehörigkeitsfunktion der Ausgangs-`FuzzyTerm` anwendet und das Resultat mit dem Akzeptanzgrad der Prämisse auf das Minimum vergleicht und dieses Minimum zurückgibt. Der Typ des Resultats der Aktivierung ist also `(Double => FuzzyBool)`

```
1 ((x: Double) => new
     FuzzyBool(Math.min(minAcceptance,rule.outputs.func.apply(x).value)))
```

Listing 3: Berechnung der Aktivierung

Aggregation Der letzte Schritt ist die Aggregation. Da die Prämissenauswertung und die Aktivierung für jede Regel in der Regelbasis ausgeführt wird, haben wir nach der erfolgreichen Aktivierung k verschiedene Fuzzy-Stellgrößen die nun durch die Aggregation zusammengeführt werden. Dafür wird die Maximumbildung verwendet.

Die Ausgabe aktivierten Fuzzy-Mengen $B_{1 \ldots k}$ werden mithilfe der Maximumbildung zusammengefügt. Diese eine Ausgangs Fuzzy-Menge $output_{x_1,\ldots,x_n}$ wird dann an das Defuzzifizierungs-Interface übergeben und mit der gewählten Defuzzifizierungs-Methoden in einen scharfen Wert umgewandelt.

$$output_{x_1,\ldots,x_n} : y \mapsto \max_{r \in (1,\ldots,k)} (output^{R_r}_{x_1 \ldots x_n}) \tag{14}$$

```
1 def combineOutputRules(outputFunctions: List[(Double) => FuzzyBool]):
     ((Double) => FuzzyBool) = {
2 (x) => new FuzzyBool(outputFunctions.foldLeft(Double.MinValue)((a, b) =>
     Math.max(a, b(x).value)))
3 }
```

Listing 4: Berechnung der Aggregation

Wenn man die k Regeln aus dem Schritt der Aktivierung nun zusammenfügt erhält man den Typ `List[(Double => FuzzyBool)]`. Daraus soll nun wieder eine Funktion des Typs `(Double => FuzzyBool)` gemacht werden. Die Kombination durch Maximumbildung wie in Listing 4 zu sehen gibt uns das gewünschte Resultat. Diese Funktion muss nun an das Defuzzyfizierungs-Interface gegeben werden.

Defuzzifizierung Die aus der Aggreagtion entstanden Funktion muss nun auf das Interval des `FuzzyValueConnectors` angewendet werden und danach mit einer Defuzzifizierungs-Methode in einen scharfen Wert umgewandelt werden.

```
1   var setpoint = (connector.minVal to
        connector.maxVal)(logic.defuzzy.func.apply(output))
```

Listing 5: Anwendung der Defuzzifizierungs-Methode

2.5 Grafische Benutzeroberfläche

Zur Realisierung der grafischen Benutzeroberfläche wurde JavaFX eingesetzt. Bei JavaFX handelt es sich um die aktuellste Technologie um Benutzeroberflächen für Java zu entwickeln. Da Scala ebenfalls eine JVM Sprache ist wie Java, können diese beiden Sprachen gut kombiniert werden. Für die JVM geschriebenen Scala Programme sind Byte-Code kompatible zu Java. Durch diese Kompatibilität können unter anderem bestehende Java Bibliotheken auch aus Scala verwendet werden, so auch das JavaFX Framework.

Ein Vorteil von JavaFX ist, das mithilfe eines Designer die Oberfläche erstellt werden kann. Der Designer erzeugt dabei keinen Quellcode, sondern eine HTML ähnliche Datei, eine FXML-Datei. Ein kleiner Ausschnitt ist in Listing 6 zu sehen.

```
114 <Label text="Geschweindigkeit" GridPane.rowIndex="2" />
115 <Label text="Position" GridPane.rowIndex="3" />
116 <TextField fx:id="tfNewton1" GridPane.columnIndex="1" />
117 <TextField fx:id="tfAcc1" GridPane.columnIndex="1" GridPane.rowIndex="1" />
```

Listing 6: Kleiner Auszug aus einer FXML-Datei

Neben der Definition von GUI-Elementen und Layoutinformationen ist in der FXML-Datei auch eine Controller-Klasse angegeben. Zum erzeugen der GUI wird mithilfe des FXMLLoader die FXML-Datei geladen. Der FXMLLoader erstellt dann zur Laufzeit eine Instanz der Controller-Klasse. Über die fx:id kann der FXMLLoader die Elemente der

FXML-Datei den entsprechenden Klassenvariablen in der Controller-Klasse zuordnen und mithilfe von *Dependency Injection* werden diese dann auch Objekten befüllt.

```
31 @FXML
32 var sliderNewton: Slider = _
33 @FXML
34 var tfNewton1: TextField = _
```

Listing 7: Annotierte Deklaration für den Slider und das Textfeld der Antriebskraft

Ein weiterer Vorteil von JavaFX ist die integrierte Binding-API. Mit Bindings können Variablen verknüpft, wenn sich dann der Wert der einen Variable ändert, aktualisiert sich automatisch der Wert der anderen Variable. Das Listing reflst:fxbinding zeigt wie der Wert des Textfeldes für die Antriebskraft mithilfe eines Konverters mit einer numerischen Variable newton1 verknüpft wird. Der Wert von newton1 wird dann zusätzlich noch mit dem Zahlenwert des Sliders verbunden. Der Inhalt des Textfeldes beinhaltet somit immer den aktuellen Wert des Sliders und das Fuzzy-Model könnte sich einfach auf eine Wertänderung der Variable newton1 registrieren.

```
114 Bindings.bindBidirectional(tfNewton1.textProperty(), newton1, new
        NumberStringConverter())
115 newton1.bindBidirectional(sliderNewton.valueProperty())
```

Listing 8: Binding, der Wert des Sliders wird mit dem des Textfeldes verknüpft

Die Binding-API von JavaFX ermöglicht es also, Interaktive und Reaktive GUI-Anwendungen zu entwickeln, was für die Entwicklung eines Simulators für Fuzzy-Regelung essentiell wichtig ist, da Während der Simulation viele Daten des Fuzzy-Models in Echtzeit auf der GUI angezeigt werden. Abbildung 12 zeigt den finalen Stand der Anwendung. Im unteren Teil der Anwendung werden die aktuellen Werte der Autos numerisch dargestellt, wobei rechts das vorfahrende Auto, links das Fuzzy-gesteuerte Auto und in der Mitte der Abstand angezeigt wird. Darüber auf der linken Seite wird eins von vier Diagrammen angezeigt, welche die Änderung der Wert über die Zeit darstellen. Auf der rechten Seite wird der aktuelle Fuzzy-Ausgang visualisiert. Zum interagieren mit der Simulation dient der Slider unten rechts. Durch ein hochschieben wird das vorfahrende Auto beschleunigt und zum Abbremsen wird der Slider nach unten verschoben. Alle GUI-Elemente, welche Daten der Fuzzy-Regelung darstellen werden in Echtzeit aktualisiert damit das Verhalten der Fuzzy-Regelung optimal beobachtet werden kann.

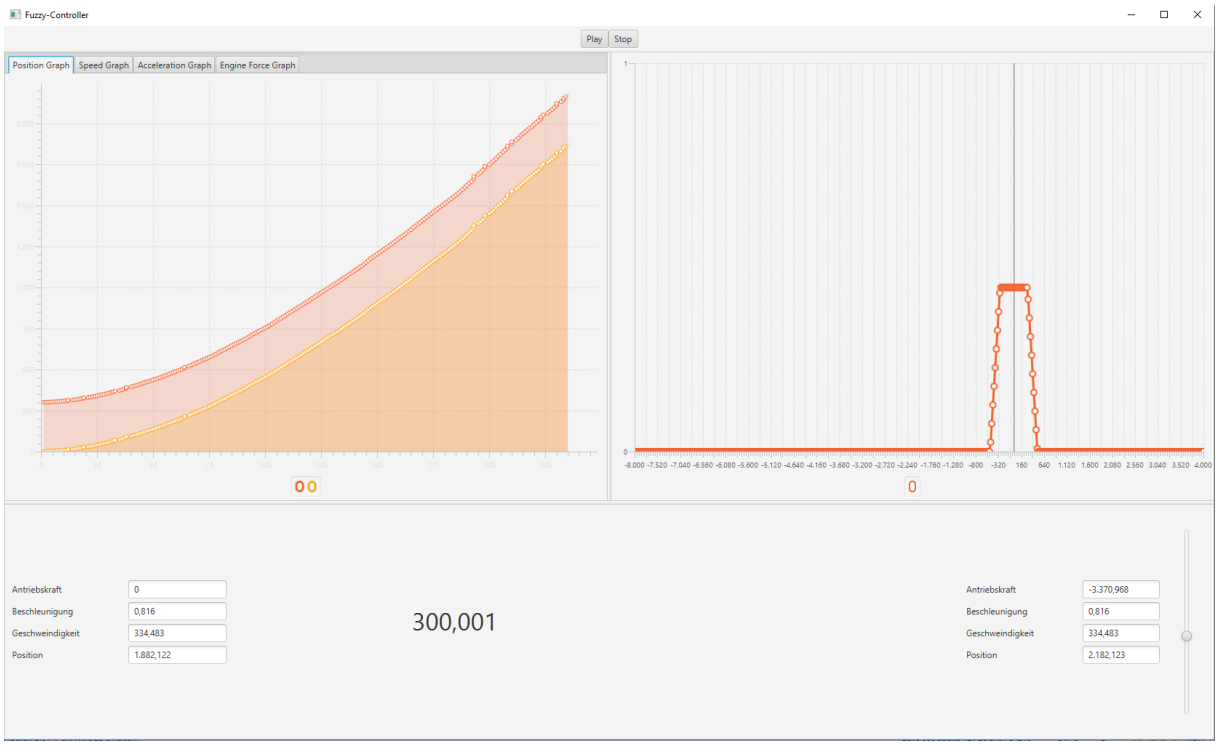

Abbildung 12: Bildschirmfoto der Anwendung

2.6 Konfiguration

Der Nutzer des Programms kann zur Laufzeit folgende Konfigurationen für die Fuzzy-Regelung vornehmen:

- Partitionierung der Eingangsvariablen
- Partitionierung der Ausgangsvariablen
- Erstellung der linguistischen Regeln
- Wahl der Defuzzifizierungsmethode

Es wurden folgende Voreinstellungen für die Regelung gemacht:

Wertebreiche Distanz: 0-500 Meter
Geschwindigkeit: 0-250 km/h
Beschleunigung: -8000-4000 N

Partitionierung Distanz
isVeryClose $= -\infty, 100, 200$

isClose $= 100, 200, 300$
isNormal $= 250, 300, 350$
isFar $= 300, 300, 500$
isVeryFar $= 400, 500, \infty$

Partitionierung Geschwindigkeit

isSlow $= -\infty, 15, 30$
isFast $= 15, 50, 100$
isExtreme $= 50, 150, 350$

Partitionierung Beschleunigung

fullBrake $= -\infty, -8000, -7000$
medBrake $= -8000, -5000, -2000$
brake $= -5000, -2000, 0$
roll $= -500, 0, 500$
speed $= 0, 1000, 2000$
medSpeed $= 1000, 2000, 3000$
fullSpeed $= 2000, 3000, \infty$

Regeln

```
1  IF Distance = isVeryFar AND Speed = isSlow THEN Force = fullSpeed
2  IF Distance = isVeryFar AND Speed = isFast THEN Force = fullSpeed
3  IF Distance = isVeryFar AND Speed = isExtreme THEN Force = fullSpeed
4  IF Distance = isFar AND Speed = isSlow THEN Force = fullSpeed
5  IF Distance = isFar AND Speed = isFast THEN Force = medSpeed
6  IF Distance = isFar AND Speed = isExtreme THEN Force = speed
7  IF Distance = isNormal AND Speed = isSlow THEN Force = roll
8  IF Distance = isNormal AND Speed = isFast THEN Force = roll
9  IF Distance = isNormal AND Speed = isExtreme THEN Force = roll
10 IF Distance = isClose AND Speed = isSlow THEN Force = brake
11 IF Distance = isClose AND Speed = isFast THEN Force = medBrake
12 IF Distance = isClose AND Speed = isExtreme THEN Force = fullBrake
13 IF Distance = isVeryClose AND Speed = isSlow THEN Force = fullBrake
14 IF Distance = isVeryClose AND Speed = isFast THEN Force = fullBrake
```

```
15 IF Distance = isVeryClose AND Speed = isExtreme THEN Force = fullBrake
```

Defuzzifizierungsmethode

Es stehen folgende Defuzzifizierungsmethoden zur Auswahl:

- Mean of Maxima
- Maxium criteria
- Center of Gravity

3 Fazit

Das Team ist mit der Implementierung sehr zufrieden. Es hat Spaß gemacht das theoretisch erlernte Wissen aus der ersten Ausarbeitung einzusetzen. Wir denken außerdem, dass es sehr nützlich für zukünftige Projekte sein könnte, die Mechaniken der Fuzzy-Logik bzw. der Fuzzy-Regelung kennen gelernt zu haben. Die Implementierung in Scala war unserer Meinung nach ein voller Erfolg. Die eigentliche Fuzzy-Regelung lässt sich mit Hilfe von Funktionen höherer Ordnung sehr direkt ausdrücken. Außerdem hat das Team durch das Projekt die Kenntnisse im Bereich funktionale Programmierung ausbauen können. Leider ist sehr spät aufgefallen, dass zusätzlich zu den verwendeten Eingangsgrößen Geschwindigkeit und Abstand die Ableitung des Abstands sehr nützlich hätte seinen können. Dafür hätte man noch eine numerische Ableitung der Messewerte implementieren müssen, doch würde die Regelung dadurch sehr viel effizienter funktionieren.