Vergleich von klassischem und temporalem Fuzzy-Regler beschrieben in Fuzzy Control Language mit PID-Reglern

Thorsten W. SCHMIDT, Dominik HENRICH

Universität Bayreuth
Fakultät für Mathematik, Physik und Informatik
Lehrstuhl Angewandte Informatik III
D-95440 Bayreuth
E-Mail: {thorsten.w.schmidt, dominik.henrich}@uni-bayreuth.de
Http://ai3.inf.uni-bayreuth.de

1. Einleitung

1.1. Motivation

Seit vielen Jahren beschäftigt sich die Forschung mit Fuzzy-Logik und ihrer Anwendung in Fuzzy-Reglern. Seit einigen Jahren gibt es auch industriell eingesetzte Fuzzy-Regler. Diese werden zum Beispiel in Waschmaschinen oder anderen Geräten des häuslichen Gebrauchs verwendet. Die Vorteile der Fuzzy-Logik sind, dass vorhandenes Wissen über ein Regelungsprozess sehr leicht zur Modellierung eines Reglers verwendet werden kann und im weiteren Verlauf der Entwicklung und Verbesserung des Reglers dieses Wissen immer transparent bleibt und somit nicht verloren geht. Durch die klare Lesbarkeit der Fuzzy-Logik bleibt das Wissen an sich wartbar. Jedoch können diese Fuzzy-Regler nicht als Wartungssystem eingesetzt werden, da sie nicht in der Lage sind zeitliche Abhängigkeiten von Ereignissen untereinander oder überhaupt Zeit zu modellieren, wie dies beispielsweise beim Modell Checking möglich ist, welches Temporal-Logik verwendet [Karjoth87]. Aus diesem Grund ist eine Erweiterung der Fuzzy-Logik um zeitliche Aspekte nötig. In [Schmidt04] wird eine solche Erweiterung an den Prädikaten vorgenommen. Die Prädikate werden so erweitert, dass zum einen die so genannte temporale Fuzzy-Logik genauso mächtig ist wie die Temporal-Logik und zum anderen eine solide mathematische Basis geschaffen wird, so dass die Bedingungen, welche an Fuzzy-Prädikate und Zugehörigkeitsfunktionen gestellt werden auch erfüllt sind. Die Vereinigung der Temporal-Logik und Fuzzy-Logik zur *temporalen Fuzzy-Logik* mit einer Beschreibung der Zeit als Fuzzy-Zeit-Terme ist in [Schmidt05] gezeigt. Die Fuzzy-Zeit-Terme sind dabei nicht mit den Fuzzy-Zeit-Objekten von [Bovenkamp97] zu verwechseln.

Hier soll nun eine Untersuchung eines temporalen Fuzzy-Reglers im Vergleich mit PID-Reglern und klassischen Fuzzy-Reglern die Vorteile der temporalen Fuzzy-Logik zeigen. In [Giron02] wird eine Untersuchung der Stabilität von Fuzzy-Reglern auf hohem Abstraktionslevel durchgeführt. Dabei wird eine Literaturliste von annähernd 200 Veröffentlichungen angeführt, aber konkrete Ergebnisse lassen sich daraus nicht ableiten. Da wir aber unseren eigens entwickelten temporalen Fuzzy-Regler mit anderen Reglern vergleichen, führen wir diese Untersuchung selbst am Beispiel eines inversen Pendels durch.

1.2. Abgrenzung

Als Endziel, welches über die temporalen Fuzzy-Prädikate und Fuzzy-Zeit-Terme hinaus geht, möchten wir ein Wartungssystem, wie oben beschrieben entwickeln. Das Wartungssystem generiert Wartungsaufgaben für einen zu wartenden Prozess. Die Wartungsaufgaben werden dabei durch zeiterweiterte Fuzzy-Logik Regeln beschrieben. Natürlich kann das Wartungssystem auch als Überwachungssystem oder Regelungssystem verwendet werden. Das Hauptaugenmerk liegt jedoch auf dem Wartungssystem, welches Wartungsaufträge generiert. Die automatisch generierten Wartungsaufträge sind zeitlich so geplant, dass eine möglichst ökonomische Abarbeitung ohne größere Standzeiten des Prozesses möglich ist. Hierzu vergleichen

wir einen temporalen Fuzzy-Regler mit einem klassischen Fuzzy-Regler und verschiedenen PID-Reglern.

2. Temporaler Fuzzy-Regler

Der temporale Fuzzy-Regler nutzt zur Darstellung seiner Regeln und Daten eine Erweiterung der industriell eingesetzten Fuzzy-Regler-Sprache FCL (= Fuzzy Control Language) um temporale Aspekte sprachlich zu modellieren. Durch diese Erweiterung ist es möglich weitaus komplexere Regler zu schreiben, wodurch diese Regler ein breiteres Anwendungsgebiet haben als gewöhnliche Fuzzy-Regler.

2.1. Einordnung des temporalen Fuzzy-Reglers

Ein temporaler Fuzzy-Regler unterscheidet sich grundsätzlich von einem normalen Fuzzy-Regler. Zum einen in seinem Verhalten und dann auch in seinem Aufbau. Gehen wir davon aus, dass beide Regler zeitliche Aspekte behandeln sollen, dann benötigt der klassische Fuzzy-Regler die Zeit als Eingabevariable. Diese benötigt man aber bei einem temporalen Fuzzy-Regler nicht, denn dieser kennt die aktuelle Zeit und weiß somit, zu welchem Zeitpunkt eintreffende Daten aufgenommen wurden.

Die Schnittstellen des temporalen Fuzzy-Reglers kann man in zwei Klassen unterteilen. Zum einen die direkten Schnittstellen. Diese beinhalten den Input beziehungsweise Output, welche in das beziehungsweise aus dem System fließen. Diese Schnittstellen gibt es bei klassischen und temporalen Fuzzy-Reglern. Nur enthält der Input bei letzterem keine Zeit. Zum anderen gibt es die indirekten Schnittstellen. Dies sind die Schnittstellen zu anderen Komponenten, zu welchen ein der Prozess keine direkte Verbindung oder genauer gesagt keinen direkten Einfluss hat. Dies sind die Datenbank und das Orakel, welche in einem klassischen Fuzzy Regler nicht vorhanden sind (siehe Abbildung 1).

Das Informationspaar von Datenwert und Aufnahmezeitpunkt speichert der temporale Fuzzy-Regler in der Datenbank ab. Diese Datenbank bildet eine Historie auf welche immer zugegriffen werden kann. Im optimalen Fall treffen die Daten in äquidistanten Zeitabständen ein, so dass Zeitreihen mit gleichen Zeitabständen von benachbarten Daten vorliegen hat. Sind die Zeitabstände nicht äquidistant, so können die Daten immer noch interpoliert werden. Dies ist ein struktureller Unterschied zu einem klassischen Fuzzy Regler, der diese Datenbank nicht besitzt und auch nur auf aktuell in das System eingespeiste Daten zugreifen kann: „*Temperatur* **IS** *low*“. Der temporale Fuzzy-Regler dagegen kann zum Beispiel durch den Ausdruck „*Temperatur* **IS**$_{\textbf{TIME}}$ *gestern low*“ auf vergangene Werte zugreifen.

Ein weiterer struktureller unterschied ist das in Abbildung 1 dargestellte Orakel. Es soll verdeutlichen, dass es nicht einfach ist in die Zukunft zu sehen. Hier werden zur Vorhersage der zukünftigen Signalveräufe zum Beispiel lineare Vorhersagen oder andere einfache mathematische Extrapolationsmethoden eingesetzt. Sofern ein Modell des zu regelnden Prozesses bekannt ist, können auch die Modellannahmen beziehungsweise die Gleichungen die den Prozess beschreiben eingesetzt werden, um eine höhere Vorhersagegenauigkeit zu erreichen.

Ein Vergleich eines temporalen Fuzzy-Reglers mit einem klassischen Fuzzy-Regler gibt es in Kapitel 3. Außerdem werden die beiden Fuzzy-Regler mit optimierten PID-Regler verglichen.

2.2. Abkürzungen und Definitionen

Die Erweiterte Backus-Naur-Form (EBNF), ist eine Erweiterung der Backus-Naur-Form (BNF). Ursprünglich wurde sie von Niklaus Wirth zur Darstellung der Syntax der Programmiersprache Pascal eingeführt. Sie ist eine formale Metasprache, welche benutzt wird um kontextfreie Grammatiken darzustellen. Definiert ist sie durch die Norm ISO/IEC 14977 (final draft version SC22/N2249) in [IEC96].

Um eine einfachere und übersichtlichere Schreibweise zu erhalten definieren wir, dass Wörter, welche komplett in Großbuchstaben geschrieben sind immer Terminal-Symbole sind.

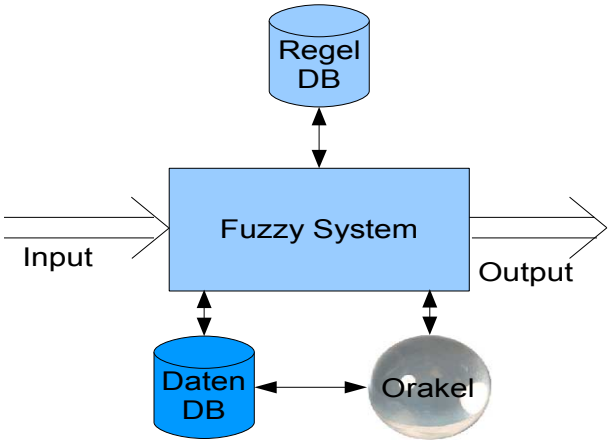


Abbildung 1: Temporaler Fuzzy Regler mit direkten (Input, Output) und indirekten Schnittstellen (Datenbank, Orakel).

Dagegen sind alle anderen Wörter Nicht-Terminal-Symbole. Für diese Wörter existiert eine Ersetzungsregel, durch welche eine neue Folge von Terminal und/oder Nicht-Terminal-Wörtern entsteht.

Wird ein Ausdruck, ein Symbol oder ähnliches grau dargestellt, so ist dieser Ausdruck oder dieses Symbol zwar in dem betreffenden Standard definiert wird, von uns aber nicht verwendet. Es handelt sich dabei um Funktionalität, welche für unsere Zwecke unnötig ist.

Werden Ausdrücke oder Symbole bei der Beschreibung eines Standards fett dargestellt, so handelt es sich um Neuerungen, welche wir einführen um diesen Standard zu erweitern. Bei den meisten Erweiterungen handelt es sich um zielgerichtete Erweiterungen, welche nötig sind um zeitliche Aspekte zu beschreiben. Manche Änderungen werden jedoch nur vorgenommen, um die Arbeit mit der Regelungssprache zu vereinfachen.

2.3. Fuzzy Control Language

Die Fuzzy Control Language definiert und standardisiert von der International Technical Electronical Commission (IEC) in [IEC97] beschreibt einen Fuzzy-Regler hinsichtlich seiner Eingabe und Ausgabe mittels Fuzzy-Termen und seinem Verhalten mittels Regeln. Dieses Kapitel beschreibt kurz die Fuzzy Control Language so, wie sie von der IEC standardisiert ist.

Die Sprache wird zur Beschreibung von programmierbaren Reglern verwendet. Diese Regler werden programmiert, indem sie die Fuzzy-Terme zur Fuzzifizierung und Defuzzifizierung mitgeteilt bekommen. Ebenso werden die Regeln im Speicher des Reglers abgelegt. Alle Schritte von der Fuzzyfizierung bis Defuzzifizierung können vom Regler selbst in Echtzeit mit einer Reaktionszeit von wenigen Millisekunden ausgeführt werden. Beschränkungen gibt es nur in der Anzahl der Regeln und Terme die maximal definiert werden können.

Die Fuzzy Control Language wird mit der Synstaxbeschreibungssprache Enhanced Backus-Naur Form (ENBF) definiert. Die Syntaxbeschreibungssprache EBNF ist in [IEC96] beschrieben. Die wichtigsten und im Folgenden verwendeten Definitionen zum EBNF Syntax sind kurz in Tabelle 1 beschrieben.

EBNF-Term	***Beschreibung***
{*A*}	Der Ausdruck *A* kann beliebig oft wiederholt oder ausgelassen werden.
{*A*}-	Der Ausdruck *A* kann beliebig oft wiederholt werden. Mindestens jedoch ein mal.
[*A*]	Optionale Auswahl von *A*. Es kann gewählt oder weggelassen werden.
A | *B*	Eine Auswahl von entweder *A* oder *B*. Es muss genau eine Möglichkeit ausgewählt werden.
GROSSSCHREIB-WEISE	Terminalsymbole, welche durch EBNF-Regeln nicht weiter ersetzt werden.
GemischteSchreib-weise	Nicht-Terminalsymbole, welche durch EBNF-Regeln solange weiter ersetzt werden, bis nur noch Terminalsymbole vorhanden sind.

Tabelle 1: Beschreibung der wichtigsten EBNF Regeln.

Im Folgenden dargestellt ist der Aufbau eines FCL Programmes wie im Standard von [IEC97] definiert. Dies kann als Programmgerüst angesehen werden, welches noch um Fuzzy-Terme, Variablen und Regeln gefüllt werden muss.

```
1     FUNCTION_BLOCK
2
3     {STRUCT StructName
4       {StructDefinition;}-
5     END_STRUCT}
6
7     {VAR_INPUT
8       {InputVariableName: DataType;}-
9     END_VAR}
10
11    {VAR_OUTPUT
12      {OutputVariableName: REAL;}-
13    END_VAR}
14
15    {FUZZIFY InputVariableName
16      {TERM FuzzyTermName := Points ;}-
17    END_FUZZIFY}
18
19    {DEFUZZIFY OutputVariableName
20      {TERM FuzzyTermName := Points;}-
21      [METHOD: DefuzzificationMethod;]
22      [DEFAULT := Real | NC;]
23      [RANGE := (Min .. Max)]
24    END_DEFUZZIFY}
25
26    {RULEBLOCK RuleBlockName
27      [AND:AndAlgorithm;]
28      [OR:OrAlgorithm;]
29      [ACCU:AccumulationMethod;]
30      [ACT:ActivationMethod;]
31      {RULE Integer: IF Condition THEN Conclu-
        sion [WITH WeightingFactor];}-
32    END_RULEBLOCK}
33
34    {OPTION
35      UserDefinedOptions
36    END_OPTION}
37
38    END_FUNCTION_BLOCK
```

Das Programmgerüst enthält noch Nicht-Terminalsymbole. Diese werden solange durch die im Folgenden dargestellten Regeln ersetzt, bis nur noch Terminalsymbole vorhanden sind. Dann liegt ein syntaktischgültiges und korrektes FCL Programm vor. Besondere Nicht-Terminalsymbole sind *String*, *Integer* und *Real*, welche für eine Zeichenkette, eine Ganzzahl und eine Gleitkommazahl stehen. Diese Sind nicht in Tabelle 2 angegeben.

Nicht-Terminalsymbol	***EBNF Ersetzungsregel***	***Beschreibung des Nicht-Terminalsymboles***
StructName	:== *String*	Gibt den Namen eines neuen Datentypes an. Dies wird nicht von uns benutzt.
StructDefinition	:== *String*	Definiert neue Datentypen. Da diese Möglichkeit von uns nicht Benutzt wird, beschreiben wir sie auch nicht genauer.
DataType	:== REAL | INT | *StructName*	Drei Verschiedene Datentypen sind möglich. REAL für Gleitkommazahlen, INT für Ganzzahlen und selbstdefinierte Structs.
InputVariableName	:== *String*	Inputvariablen für das Fuzzysystem deren Wert fuzzyfiziert wird.
OutputVariableName	:== *String*	Der Wert von Outputvariablen wird durch Regelaktivierungen gesetzt.
Min	:== *Real*	Untere Intervallgrenze von RANGE
Max	:== *Real*	Obere Intervallgrenze von RANGE
FuzzyTermName	:== *String*	Name eines Fuzzy Terms
Points	:== *Real* | {(*Real*, *Real*)}	Stützpunkte einer Zugehörigkeitsfunktionen. Jeder Fuzzy Term wird durch eine Zugehörigkeitsfunktion beschrieben, welche entweder ein Singleton (einzelner Punkt) oder ein Polygonzug (mindestens zwei Punkte) ist.
DefuzzificationMethod	:== COG | COGS | COA | LM | RM	Die Art, wie aus unscharfen Fuzzy-Werten scharfe Werte berechnet werden sollen. Am gebräuchlichsten ist die Center of Gravity Methode (COG=Schwerpunktsmethode) oder COGS (COG für Singletons). COA (Center of Area) ist nur ein Synonym für COG. LM beziehungsweise LR stehen für die left beziehungsweise right max Methode.
RuleBlockName	:== *String*	Eindeutiger Name eines Regelblockes. Es können beliebig viele Regelblöcke existieren. In einem Regelungsschritt können die aktuell zu aktivierenden Regelblöcke über ihren Namen angegeben werden.
AndAlgorithm	:== MIN | PROD | BDIF	Am gebräuchlichsten ist die Min-Methode für AND Verknüpfungen. Weitere Methoden sind das Produkt PROD oder die beschränkte Differenz BDIF. MIN: $\min(\mu_a(x), \mu_b(x))$ PROD: $\mu_a(x)\mu_b(x)$ BDIF: $\max(0, \mu_a(x) + \mu_b(x) - 1)$
OrAlgorithm	:== MAX | ASUM | BSUM	Am gebräuchlichsten ist die Max-Methode für OR Verknüpfungen. Weitere Methoden sind die algebraische Summe ASUM und die beschränkte Summe BSUM. MAX: $\max(\mu_a(x), \mu_b(x))$ ASUM: $\mu_a(x) + \mu_b(x) - \mu_a(x)\mu_b(x)$ BSUM: $\min(1, \mu_a(x) + \mu_b(x))$

AccumulationMethod	:== MAX | BSUM | NSUM	Am gebräuchlichsten ist die Max-Methode für die Akkumulierung. Weitere Methoden sind die beschränkte BSUM und die normalisierte Summe NSUM. MAX: max ($\mu_a(x)$, $\mu_b(x)$) BSUM: min (1, $\mu_a(x) + \mu_b(x)$) NSUM: $\frac{\mu_a(x)+\mu_b(x)}{(\max(1,\max_{x'}(\mu_a(x')+\mu_b(x'))))}$
ActivationMethod	:== MIN | PROD	Am gebräuchlichsten ist die Min-Methode für die Aktivierung. Eine weitere Methode ist das Produkt PROD. MIN: min (μ_{Rule}, $\mu_{if}(x)$) PROD: μ_{Rule} $\mu_{if}(x)$
Condition	:== <[NOT] *AtomIn*> | <[NOT] *SystemCheck*> | <[NOT] (*Condition* {<AND | OR> *Condition*}–)>	Eine beliebige Verschachtelungstiefe ist für die Bedingungen einer Regel möglich.
Conclusion	:== (*AtomOut*) | < *Conclusion*, *Conclusion* >	Eine beliebige Anzahl von Folgerungen sind pro Regel möglich.
AtomIn	:== (*InputVariableName* IS *FuzzyTermName*)	Atom in einer Bedingung ist eine Fuzzy-Bedingung.
AtomOut	:== (*OutputVariableName* IS *FuzzyTermName*)	Atom in einer Folgerung ist eine Fuzzy-Bedingung.
WeightingFactor	:== *Real*	Gewichtungsfaktor einer Regel. Wird kein Gewicht angegeben, so wird das Gewicht als 1.0 angenommen.
UserDefinedOptions	:== *String*	Hier können beliebige vom Benutzer zu definierende Optionen in freiem Format angegeben werden. Diese müssen auch von Benutzer selbst ausgewertet werden. Diese Möglichkeit wird nicht von uns genutzt.

Tabelle 2: Erläuterungen zu Nicht-Terminalsymbolen der EBNF-Beschreibung der Fuzzy Control Language.

2.4. Temporal Fuzzy Control Languange

Dieses Kapitel beschäftigt sich mit der zeitlichen Erweiterung der Fuzzy Control Language, der so genannten Temporal Fuzzy Control Language. Diese Erweiterung ist nötig, da sich, wie in Kapitel 2.2 beschrieben, die zugrunde liegende Fuzzy-Logik Sprache geändert hat. Die temporale Fuzzy-Logik hat mehr Prädikate und mehr Formulierungsmöglichkeiten (zum Beispiel die Angabe von Zeit in Bedingungen oder die Deklaration von Fuzzy-Zeit-Termen) wie die klassische Fuzzy-Logik. Diese Erweiterungen werden im Folgenden genauer vorgestellt.

Wie schon in Kapitel 2.3 eingeführt, kann man auch die Temporal Fuzzy Control Language in EBNF beschreiben. Schon im Programmgerüst gibt es Unterschiede zur Fuzzy Control Language. Diese sind, wie in folgender Legende erklärt, hervorgehoben:

- Grau: In FCL definiert, aber von TFCL nicht unterstützt. Dies betrifft nur die Definition von eigenen Datenstrukturen (Structs), Ganzzahlen als Datentypen und die Angabe von eigenen Optionen. Mit anderen Worten, in FTCL gibt es nur REAL als möglichen Datentyp.
- Normal: In FCL und TFCL gleichermaßen definiert.
- Fett: In FCL nicht vorgesehen, aber in TFCL definiert.

Gegeben ist nun das Programmgerüst, wie es für die Temporal Fuzzy Control Language verwendet wird.

```
1     FUNCTION_BLOCK
2
3     {STRUCT StructName
4       {InputVariableName: DataType}-
5     END_STRUCT}
6
7     {VAR_SYSTEM
8       {SystemVariableName [actuator]:
      DataType;}-
9     END_VAR}
10
11    {VAR_INPUT
12      {InputVariableName: DataType;}-
13    END_VAR}
14
15    {VAR_OUTPUT
16      {OutputVariableName [actuator]: REAL;}-
17    END_VAR}
18
19    {VAR_EVENT
20      {EventName;}-
21    END_VAR_EVENT}
22
23    {FUZZIFY InputVariableNameStar
24      {TERM FuzzyTermName := Points ;}-
25      [RANGE := (Min .. Max)]
```

```
26    END_FUZZIFY}
27
28    {DEFUZZIFY OutputVariableName
29      {TERM FuzzyTermName := Points;}-
30      [METHOD: DefuzzificationMethod;]
31      [DEFAULT := Real | NC;]
32      [RANGE := (Min .. Max)]
33    END_DEFUZZIFY}
34
35    FUZZY_TIME_TERM FuzzyTimeTermName
36      {FACT Integer := Poins}-
37      {TIME Integer := Poins}-
38    END_FUZZY_TIME_TERM
39
40    {EVENT EventName
41      {TASK TaskName := TaskNumber;}-
42    END_DEFUZZIFY}
43
44    {RULEBLOCK RuleBlockName
45      [AND:AndAlgorithm;]
46      [OR:OrAlgorithm;]
47      [ACCU:AccumulationMethod;]
48      [ACT:ActivationMethod;]
49      [PREDICTION:PredictionMethod;]
50      {RULE Integer: IF Condition THEN Conclu-
      sion [WITH WeightingFactor];}-
51    END_RULEBLOCK}
52
53    {OPTION
54      UserDefinedOptions
55    END_OPTION}
56
57    END_FUNCTION_BLOCK
```

Bei den Erläuterungen zu den Nicht-Terminalsymbolen in Tabelle 3 werden nur Änderungen und Neuerungen zu Tabelle 2 gezeigt. Außerdem verfügen nun auch Eingabevariablen über die Angabe eines Gültigkeitsberecihes RANGE für die eingehenden Daten. Diese Angabe wird benötigt, um den Bereich zur Anzeige von Daten festzulegen.

Datenfeld	***EBNF Notation***	***Beschreibung***
StructName	:== *String*	Nicht unterstützt
DataType	:== REAL | INT | *StructName*	Nur Realvariablen (Doublegenauigkeit) werden unterstützt.
EventName	**:==** ***String***	Name eines Ereignisses, welches beim Auftreten an einen Benutzer gegeben werden kann.
SystemVariableName	**:==** ***String***	Systemvariablen sind Realvariablen, welche nur innerhalb des Fuzzysystems verwendet werden. Sie werden nicht fuzzifiziert und auch nicht defuzzifiziert. Durch Angabe des Schlüsselwortes actuator wird die Variable zur Steuerung verwendet. Ansonsten ist es nur eine interne Variable.
InputVarName	:== *String* | ***InputVarNameStar***	Inputvariablen für das Fuzzysystem deren Wert fuzzyfiziert wird.
InputVariableNameStar	**:==** ***String*** **|** ***InputVarName***	Der Name der Variablen darf mit dem Zeichen '*' enden, wenn der Stern durch eine Zeichenkette ersetzt werden kann, so dass der resultierende Name einer unter VAR_INPUT definierten Variablen entspricht. Die Definition der TERME betrifft alle Variablen auf, welc*he InputVariableNameStar* expandiert werden kann.
OutputVariableName	:== *String*	Der Wert von Outputvariablen wird durch Regelaktivierungen gesetzt. **Durch Angabe des Schlüsselwortes actuator wird die Variable zur Steuerung verwendet. Ansonsten werden sie nicht nach Außen gegeben**
FuzzyTimeTermName	**:==** ***String***	Name eines Fuzzy-Zeit-Terms. Der Term gibt an, aus welchem unscharfen Zeitbereich ein Prädikat Daten verarbeitet.
Defuzzification-Method	:== COG | COGS | COA | **<MCOG :=** ***Real*****>** **| MM |** LM | RM	Die Art, wie aus unscharfen Fuzzy Werten scharfe Werte berechnet werden sollen. Am gebräuchlichsten ist die Center of Gravity Methode (COG=Schwerpunktsmethode) oder COGS (COG für Singletons)
PredictionMethod	**:== TAYLOR | LINEARITY | SPLINE | FIT**	Art der Vorhersage von Daten. Entweder durch annähern eines Taylorpolynomes, gewichtete Linearität, extrapolation durch Splineinterpolation oder durch Fit mit Downhill-Simplex.
Conclusion	:== (*AtomOut*) | **(*SystemModify*)** | < *Conclusion*, *Conclusion* | **GOTO** ***RuleBlockName*** >	Folgerung einer Regel. Neu ist hier *SystemModify* um Systemvariablen zu verändern und GOTO um den aktuellen Rebelblock zu wechseln.
AtomIn	:== (*InputVariableName* < IS | ***IsTime*** | ***IsExists*** | **GREATER | LESS** > *FuzzyTermName*)	Atom in einer Bedingung
AtomOut	:== (*OutputVariableName* < IS | ***TimeIs*** | ***TimeExists*** > *FuzzyTermName*) | **(*EventName* IS *TaskName*)**	Atom in einer Folgerung
SystemCheck	**:== (*SystemVariableName* < < | > | = > <REAL |** ***SystemVariableName*** **>)**	Vergleicht die Werte von Systemvariablen, ob die kleiner, größer oder gleich einem gegeben Wert oder dem Wert einer anderen Variablen ist.

SystemModify	**:==** ***SystemVariableName*** **< :=** **REAL** | **:=** ***SystemVariableName*** | ++ | – – | >	Systemvariablen können einen festen Wert zugewiesen bekommen, genauso wie sie um die Regelaktivierung erhöht beziehungsweise erniedrigt werden können. Sie können aber auch den Wert einer anderen Systemvariablen erhalten
IsTime	**:== IS_TIME [*FuzzyTimeTermName*]**	IS_TIME betrachtet, ob alle Daten in ihrer Gesamtheit in dem gegebenen FuzzyTerm der Bedingung liegen. Ein Fuzzy Zeit Termes bestimmt den Zeitbereich, aus aus welchen Daten betrachten werden. Siehe dazu [Schmidt04] und [Schmidt05].
IsExists	**:== IS_EXISTS[*FuzzyTimeTermName*]**	IS_EXISTS betrachtet, ob es ein Datum gibt, welches in dem gegebenen FuzzyTerm der Bedingung liegt. Ein Fuzzy Zeit Termes bestimmt den Zeitbereich, aus aus welchen Daten betrachten werden. Siehe dazu [Schmidt04] und [Schmidt05].

Tabelle 3: Erläuterungen zu Terminalsymbolen der EBNF-Beschreibung der Temporal Fuzzy Control Language. Änderungen zu Tabelle 2 sind dabei Fett hervorgehoben.

2.5. Auswerten von TFCL-Beschreibungsdateien

Hier beschreiben wir wie eine TFCL-Datei ausgewertet wird. Eine solche Datei beinhaltet zuallererst Eingabe- und Ausgabevariablen. Für diese Variablen sind ein oder mehrere Fuzzy-Terme gegeben. Jeder Fuzzy-Term, auch die Fuzzy-Zeit-Terme, wird durch einen Polygonzug beschrieben, welcher die Zugehörigkeitsfunktion $\mu(x)$ angibt. Wenn diese Informationen eingelesen sind, dann sind die Schnittstellen zum Regler definiert. Anschließend müssen nur noch die Regelblöcke eingelesen werden.

Jeder Regelblock hat einen eindeutigen Namen, durch welchen er identifiziert wird. Dies ermöglicht es, unterschiedliche Regelblöcke zu unterschiedlichen Zeiten oder unterschiedlich oft auszuwerten. Soll ein Regelblock ausgewertet werden, so werden alle Regeln in diesem einzeln und nacheinander ausgewertet. Regeln die zuerst in einem Regelblock stehen werden auch zuerst ausgewertet. Dies ist aber nur von Belang, wenn Systemvariablen und GOTO-Anweisungen verwendet werden. Bei Systemvariablem werden die Änderungen sofort nach der Regelauswertung übernommen. Deshalb hat eine unterschiedliche Auswertungsreihenfolge auch unterschiedliche Auswirkungen. Auch GOTO-Anweisungen werden sofort ausgeführt, so dass eine Auswertung der weiteren Regeln im Regelblock nicht mehr stattfindet. Die Defuzzifizierung wird sehr wohl noch ausgeführt, aber alle folgenden Regeln haben darauf keinen Einfluss mehr.

Die Regeln der Regelblöcke werden beim Einlesen in zwei Teile aufgeteilt. Die Regel-Bedingung und die Regel-Folgerung. Dadurch, dass die Daten in einer eigenen und effizienten Datenstruktur vorliegen, benötigen wir keinen Interpreter, der immer wieder Zeile für Zeile interpretiert. Dies Beschleunigt die Auswertung der TFCL-Dateien.

Die Regel-Bedingung besteht aus einzelnen Atomen, welche durch AND beziehungsweise OR verknüpft sind. Intern wird eine Regel-Bedingung als Datenstruktur durch einen Baum dargestellt. Außerdem kann die Reihenfolge der Auswertung durch Klammerung der Ausdrücke festgelegt werden. Bei jeder Klammerung steigt man eine Ebene tiefer in den Baum. Ausdrücke die innerhalb einer Klammerungsebene stehen befinden sich auch auf der gleichen Ebene im Baum. Die Entscheidung einen Baum anstatt die herkömmlich verwendete Regel-Matrix zu verwenden ist darin begründet, dass eine Matrix sehr viel Speicherplatz belegt und man in dieser keine OR-Verknüpfung und noch weniger verschachtelte Ausdrücke darstellen kann.

Die Regel-Folgerung dagegen ist sehr viel einfacher aufgebaut. Sie besteht aus einer Liste von Folgerungen, welche gelten, sollte die Regel-Bedingung gültig sein. Zur Auswertung wird jedes Element dieser Liste nacheinander von Links nach Rechts ausgewertet. Elemente die zuerst in der Liste stehen werden auch zuerst ausgewertet. Einen Einfluss auf die Auswertung hat noch der Gewichtungsfaktor einer Regel, welcher die Aktivierung einer Regel verstärken oder abschwächen kann. Der Einfluss hiervon geht direkt in die Folgerungen ein, da diese immer unter Beachtung der gewichteten Regelaktivierung ausgewertet werden. Zum Beispiel die am meisten vorkommende Folgerung „*x* IS *hoc*h“. Die Ausgabevariable *x* ist *hoch* zu einem bestimmten Grad. Der Grad ist genau die gewichteten Regelaktivierung.

Nachdem die Daten eingelesen wurden. Können mit neuen Eingabewerten neue Ausgabewerte bestimmt werden. Diese Aufgabe unterteilt sich in vier Schritte.

Der erste Schritt der Auswertung ist die Aggregation (auch Fuzzifizierung genannt). Hierfür wird die Liste aller Regeln des aktiven Regelblockes sequentiell durchgegangen. Bei jeder Regel werden deren atomaren Bedingungen durch die vorliegenden Eingaben aktiviert. Die atomaren Fuzzy-Bedingungen werden durch traversieren des Baumes der Regel-Bedingungen gefunden.

Danach erfolgt die Aktivierung. Hierzu wird die Aktivierung aller atomaren Bedingungen, welche keine Kinder und jeweils den gleichen Vater im Baum haben, mit der Standard Fuzzy-Aktivierung berechnet. Das Ergebnis ist der Aktivierungsgrad des Vaters, dessen Kinder nun aus dem Baum gelöscht werden. Dieser Berechnungsschritt wird solange ausgeführt, bis der Baum nur noch aus einer Wurzel besteht. Die Aktivierung der Wurzel entspricht dann der Aktivierung der Regel. Bei der Berechnung hat eine AND-Verknüpfung eine höhere Priorität wie eine OR-Verknüpfung. Nach diesem Schritt ist für jede Regel bekannt, wie stark diese feuert.

Anschließend wird die Akkumulation durchgeführt. Hierzu setzt man die Aktivierung von jedem Fuzzy-Term in jeder Ausgabevariablen auf Null. Danach setzt man für jede Regelfolgerung die Aktivierung der Regel nach der MAX- oder PROD-Methode in den angegebenen Fuzzy-Term, der durch die Folgerung gegeben ist. Das Ergebnis von diesem Schritt sind Ausgabevariablen mit aktivierten Fuzzy-Termen. Außerdem gibt es noch Folgerungen welche Variablen (zum Beispiel Systembariablen) direkt ändern (zum Beispiel durch den Operator ++) oder den Programmablauf beeinflussen (zum Beispiel GOTO).

Der letzte Schritte ist die Defuzzifizierung. Mit den aktivierten Fuzzy-Termen in den Ausgabevariablen bestimmt man nun durch die gegebene Defuzzifizierungsmethode für jede Ausgabevariable deren scharfen Ausgabewert.

Damit ist die Auswertung der TFCL-Datei abgeschlossen. Gelangen neue Daten in das Fuzzy-System, so werden nur noch die letzten vier Schritte wiederholt. Das Einlesen der Beschreibungsdatei und das Aufbauen der Bäume zur Beschreibung der Regel-Bedingungen entfällt, da sich diese beim Programmablauf nicht ändern.

3. Vergleich vom Fuzzy- und PID-Reglern am Beispiel eines simulierten Stabwagens

In diesem Kapitel vergleichen wir qualitativ und quantitativ verschiedene PID- und Fuzzy-Regler miteinander. Dies geschieht am Beispiel eines simulierten Stabwagens mit einem inversen Pendel. Ein wichtiges Augenmerk liegt bei der Stabilitätsuntersuchung und der Zeit zum Entwickeln und Implementieren eines Reglers.

Zuerst zum Experiment selbst. Beim inversen Pendel handelt sich um ein Beispielproblem, das Balancieren eines Metallstabes im Schwerefeld durch Bewegung seines Fußpunkts (Abbildung 2). Das Beispiel ist an das inverse Pendel aus [Bothe95] angelehnt, bei welchem das Pendel zwar vorgestellt wird, aber keine Untersuchung des Verhaltens oder eine Auswertung der Regeln berechnet wird. Vergleiche dieser Art können in der Literatur bei [Butkiewicz00] gefunden werden. Hier wird ebenfalls ein Fuzzy-Logik-Regler mit einem PID-Regler verglichen, aber es wird nur die Zeit und das Schwingverhalten untersucht. Außerdem wurde kein temporaler Fuzzy-Regler bei den Experimenten verwendet. Dies und die wirkenden Kräfte untersuchen wir hier.

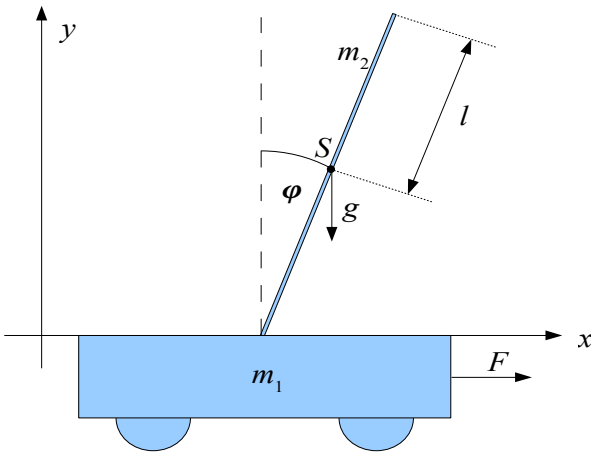


Abbildung 2: Inverses Pendel auf Stabwagen.

Die Untersuchung in [Chang97] geht noch ein Stück weiter. Hier werden PID- und Hybrid-Regler (= P-Fuzzy + ID-Regler) in Kombination mit einer manuellen Steuerung einer Verbrennungsanlage verglichen. Die Überwachten Kriterien sind Dampfdruck und -fluss und Sauerstoff- und Kohleverbrauch. Das Ergebnis der Untersuchungen ist, dass der Hybrid-Regler dem PID-Regler überlegen ist.

Bei unserem Experiment sei zur Vereinfachung im Folgenden angenommen, dass die Masse im Metallstab homogen verteilt ist. Die Massen m_1 und m_2 sind durch ein Drehgelenk reibungsfrei miteinander Verbunden. Hierbei erlaubt das Drehgelenk nur eine Bewegung in einer Ebene, der Stab hat also nur einen Freiheitsgrad. Der Winkel zwischen dem Stab und der y-Achse beträgt φ. Mit der Kraft F kann die untere Masse m_1 längs der x-Achse beschleunigt und dadurch der Stab verschoben beziehungsweise balanciert werden. Als Werte setzen wir folgende Konstanten ein: $m_1 = 1\text{kg}$, $l = 2\text{m}$, $m_2 = m_1$, $g = 9.81\text{m/s}^2$).

Betrachtet man die Richtungen, in welche die Kräfte F und g in Bezug des Massenschwerpunktes S wirken, dann erhält man für die Beschleunigungen der Auslenkung $\ddot{\varphi}$ und der Beschleunigung des Wagens $\ddot{x}$ nach [Schneider00] folgende Gleichungen:

$$\ddot{\varphi} = \frac{g \cdot \sin(\varphi) + \cos(\varphi)\dfrac{F + m_2 l \dot{\varphi}^2 \sin(\varphi)}{m_1 + m_2}}{l\left(\dfrac{4}{3} - \dfrac{m_2 \cos^2(\varphi)}{m_1 + m_2}\right)} \qquad \text{mit} \qquad \ddot{\varphi} = \frac{\partial \dot{\varphi}}{\partial t} = \frac{\partial^2 \varphi}{\partial^2 t}$$

$$\ddot{x} = \frac{F + m_2 l\left(\dot{\varphi}^2 \sin(\varphi) - \ddot{\varphi}\cos(\varphi)\right)}{m_1 + m_2} \qquad \ddot{x} = \frac{\partial \dot{x}}{\partial t} = \frac{\partial \partial x}{\partial \partial t}$$

Um vergleichbare und wiederholbare Ergebnisse zu erhalten, betrachten wir den Stabwagen nur in einer simulierten Umgebung mit obigen Formeln. In der Simulation bringen 5 verschiedene Regler (3 PID und 2 Fuzzy) den Stab in eine Ruhelage und halten ihn dort. Der Startwinkel ist immer $\varphi_0 = +1{,}0$ [rad] $\approx 57{,}3°$. Die Simulationszeit beträgt 2 Sekunden. Innerhalb dieser Zeitspanne ist es allen Reglern möglich die Auslenkung auf Null zu reduzieren und den Stab in dieser Lage zu halten.

Gewünscht ist eine möglichst kurze Zeit zum Stabilisieren des Stabes. Stabil bedeutet, dass sich der Stab nur noch ganz wenig bis gar nicht bewegt. Wir definieren die stabile Lage als eine Auslenkung von weniger als $\Delta\varphi = 0{,}005$ (<0,29°), was bei einer Stablänge l von 2 Metern einer Schwankung an der Spitze von weniger als plus minus einem Zentimeter entspricht. Für einen Zeitpunkt t_0, ab welchem der Stab stabil ist, gilt folgende Bedingung:

$$\min_{t_0}\left(\exists t_0 \forall t \geq t_0 : \varphi(t) \leq 0{,}005 < 0{,}29\,°\right)$$

Alle fünf Regelkreise sind im folgenden in PID- und Fuzzy-Regler unterteilt und im jeweiligen Kapitel 3.1 für PID-Regler und 3.2 für Fuzzy-Regler genauer beschrieben. In Kapitel 3.3 werden die Ergebnisse der einzelnen Regler im Experiment dargestellt und miteinander verglichen.

3.1. Der PID-Regler

Der PID-Regler unterteilt sich in drei unabhängige Regler. Den Proportional-, Integral- und Differential-Regler. Jeder Regler hat als Eingang eine Regeldifferenz $e(t)$. Die Regeldifferenz gibt an, um wie viel bei einem zu regelnden Prozess ein Istwert von einem Sollwert abweicht. Das Ziel des Reglers ist die Regeldifferenz klein und wenn möglich auf Null zu halten. Um dies zu leisten, kann der Regler durch einen Stellwert $u(t)$ direkt oder inderekt Einfluss auf den Prozess nehmen.

Der einfachste Regler ist der P-Regler. Er verstärkt durch den Proportionalitätsfaktor K_P die Regeldifferenz $e(t)$ und reagiert unmittelbar auf eine Veränderung in der Regeldifferenz. Problematisch ist es, wenn ein großes K_P verwendet wird, da in diesem Fall der Prozess dazu

neigt, sich aufzuschwingen und so die Regeldifferenz immer größer wird. Der P-Regler sieht wie folgt aus: $u(t) = K_P \cdot e(t)$

Der I-Regler reagiert langsamer auf Abweichungen in der Regeldifferenz als der P-Regler, da sich die Abweichungen erst über ein Integral über vergangene Werte aufsummieren müssen. Durch diese Trägheit ist der I-Regler unempfindlicher gegenüber kleinen Störungen. Es ist zu beachten, dass der Zeitraum über den Integriert wird hinreichend lang sein muss, denn ansonsten würde der I-Regler dem P-Regler entsprechen. Der Faktor K_I gibt an, wie stark der Integralanteil auf den Stellwert wirkt. Der I-Regler sieht wie folgt aus: $u(t) = K_I \cdot \int e(t)\,dt$

Der D-Regler reagiert nur auf Änderungen in der Regeldifferenz. Ist die Regeldifferenz konstant, so bleibt er inaktiv. Deshalb kann der D-Regler auch keine konstanten Regeldifferenzen ausgleichen. Dies macht ihn ohne einen anderen Regler unbrauchbar. Der Vorteil des D-Reglers ist seine schnelle Reaktion auf Störungen. Der Faktor K_D gibt an, wie stark der Differenzialanteil auf den Stellwert wirkt. Der D-Regler sieht wie folgt aus: $u(t) = K_D \cdot \frac{\partial e(t)}{\partial t}$

Verschiedene Kombinationen von P-, I- und D-Reglern sind möglich. Wir beschränken uns auf die gebräuchlicheren Kombinationen von P-, PI- und PID-Regler.

Alle Regler liefern ein Steuersignal $u(t)$, aber zum Regeln des Prozesses muss bekannt sein, wie das Steuersignal $u(t)$ Einfluss auf dem Prozess nimmt. Bei dem Beispiel des Stabwagens gibt das Steuersignal die gewünschte Drehgeschwindigkeit $\dot{\varphi}$ des Stabes an. Diese kann aber nicht direkt eingestellt werden. Nur die Kraft F, mit welcher der Stabwagen beschleunigt wird, kann geändert werden. Demnach benötigt man noch eine Umrechnung der benötigten Kraft F, um die gewünschte Geschwindigkeitsänderung $\Delta \frac{\partial \dot{\varphi}}{\partial t} = \ddot{\varphi}$ des Stabes zu erreichen. Stellt man die Gleichung für die Winkelbeschleunigung aus dem vorherigen Kapitel nach der Kraft F um, so erhält man folgende Gleichung:

$$F = \frac{m_1 + m_2}{\cos(\varphi)} \left(l \left(\frac{4}{3} - \frac{m_2 \cos^2(\varphi)}{m_1 + m_2} \right) \cdot \ddot{\varphi} - g \cdot \sin(\varphi) \right) - m_2 l \, \dot{\varphi}^2 \sin(\varphi)$$

Die Kraft F, welche nötig ist, um das System durch einwirken der Kraft F für Δt Sekunden so zu beschleunigen, dass die neue Drehgeschwindigkeit erreicht wird, wird mit einer vereinfachten Formel bestimmt. Wir gehen davon aus, dass die aktuelle Auslenkung des Stabes nahezu Null ist. Dadurch kann man den $\sin(\varphi)$ beziehungsweise $\cos(\varphi)$ mit 0 beziehungsweise 1 annähern. Diese Berechnung der Kraft F ist nur als Näherung für kleine Auslenkungen φ korrekt. Ist die Auslenkung φ des Stabes sehr klein, so gilt: $F \approx \frac{4 m_1 + m_2}{3} \cdot l \cdot \ddot{\varphi}$

Um eine möglichst optimale Regelung zu finden, müssen optimale Parameter K_i gefunden werden. Da das Pendel nur simuliert wird, kann man es sich erlauben einfach alle Parameter mit einer kleinen Diskretisierung (Schrittweite 0.05) in der Simulation zu testen. Getestet wird die Dauer zum Stabilisieren des Stabes. Liefern mehrere Parameter beziehungsweise Parameterpaare die kürzeste Stabilisierungszeit, so entscheiden wir uns für den Median der Parameter. Genauer gesagt zuerst für den Median von K_P, dann von K_I und dann von K_D. Die Parameter, bei welchen der Stab die geringste Zeit zum Stabilisieren benötigt sind in Tabelle 4 gegeben.

Regler	k_P	k_I	k_D
P	-13,00	0	0
PI	-7,20	-0,90	0
PID	-7,5	-1,65	-0,35

Tabelle 4: Optimale Regler Parameter beim hier vorgestellten inversen Pendel.

Der PID-Regler lässt sich in Pseudocode wie im Folgenden angegeben beschreiben. Sind die Parameter K_I und K_D gleich Null beziehungsweise der Parameter K_D gleich Null so beschreibt der Pseudocode einen P- beziehungsweise einen PI-Regler.

```
1   // Ältesten Wert verwerfen
2   History.shift ();
3
4   // Aktuelle Auslenkung merken
5   History.add (φ);
6
7   // P-, I- und D-Anteile berechnen
8   φ̇' = Auslenkung * K_P;
9   φ̇' += History.integral (5 * Δt) * K_I;
10  φ̇' += History.derivate (Δt) * K_D;
11
12  // Kraft zum Ändern der Drehgeschwindigkeit
13  F = ( φ̇ - φ̇' ) / Δt * l * (4 * m1 + m2) / 3;
```

3.2. Der Fuzzy-Regler

Den Fuzzy-Regler stellen wir in zwei Varianten vor. Einmal den klassischen Fuzzy-Regler und dann den zeitlich erweiterten temporalen Fuzzy-Regler. Beide Fuzzy-Regler erhalten die gleichen Eingaben (Auslenkung φ und Drehgeschwindigkeit $\dot{\varphi}$) und regeln beide die Kraft F des Stabwagen. Auch sind die Zugehörigkeitsfunktionen der Fuzzy-Terme identisch, so dass der einzige Unterschied in den verwendeten Prädikaten liegt. Der klassische Fuzzy-Regler verwendet nur das Prädikat **IS**, der temporale Fuzzy-Regler nur das Prädikat **IS$_{\text{EXISTED}}$**.

Eine Optimierung der Regler findet ausschließlich über eine Veränderung der Zugehörigkeitsfunktionen statt. Die Regelbasis wird dabei nicht verändert. Es hat sich gezeigt, dass es genügt die Zugehörigkeitsfunktion des Fuzzy-Termes für die Kraft F anzupassen. Es wird dabei ausgegangen, das die Kraft F im Bereich -100N bis +100N liegt und es günstiger ist, wenn die meisten Zugehörigkeitsfunktionen einen kleinen Bereich von -15N bis +15N beschreiben. Diese Annahmen legen schon die meisten Stützpunkte fest. Die letztendlich als optimal gefunden Zugehörigkeitsfunktionen können der angegebenen TFCL-Datei entnommen werden (Zeilen 32-43). Gefunden werden diese, indem bei den Termen NG die dritte und NM die erste und zweite Stützstelle variiert werden. Die Terme PM und PG werden symmetrisch dazu behandelt. Die Stützstellen werden solange im Bereich zwischen 15 und 80 verschoben, bis eine schnelle Stabilisierung gefunden ist.

Eine vollständige Angabe der Regelbasis bei zwei AND-verknüpften Variablen mit 5 beziehungsweise 7 Fuzzy-Termen besteht aus 35 Regeln. Diese sind in Tabelle 5 als Matrix angegeben. Die Matrix gibt an, welche Regeln gebildet werden. Für den klassischen Fuzzy-Regler gilt zum Beispiel bei einer negativen mittleren (NM) Auslenkung φ und einer negativen kleinen (NK) Drehgeschwindigkeit $\dot{\varphi}$, dass die Kraft F negativ mittel (NM) sein soll. Für den temporalen Fuzzy-Regler ändern sich nur die Prädikate und es wird eine Zeitangabe hinzugefügt. Die Regeln sehen wie folgt aus:

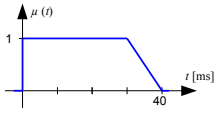
FCL: **IF** (φ **IS** NM) **AND** ($\dot{\varphi}$ **IS** NK) **THEN** (F **IS** NM)

TFCL: **IF** (φ **IS$_{\text{EXISTS}}$** *next* NM) **AND** ($\dot{\varphi}$ **IS$_{\text{EXISTS}}$** *next* NK) **THEN** (F **IS** NM)

Die beiden Fuzzy-Regler lassen sich durch folgende TFCL-Beschreibung angeben. Angegeben sind die Stützpunkte -79 und -58, -58 (siehe Zeile 33-34) für den temporalen Fuzzy-Regler. Der klassische Fuzzy-Regler hat die Stützpunkte -75, -61 und -61. Der Fuzzy-Zeit-Term in den Zeilen 45-49 und die Angabe der Vorhersagemethode in Zeile 56 werden nur für den temporalen Fuzzy-Regler benötigt. Alle anderen Zeilen sind für beide Fuzzy-Regler identisch.

```
1   FUNCTION_BLOCK
2
3   VAR_INPUT
4     auslenkung: REAL;
5     drehgeschwindigkeit: REAL;
6   END_VAR
7
8   VAR_OUTPUT
9     kraft: REAL;
10  END_VAR
11
12  FUZZIFY auslenkung
13    TERM NG := (-1.60,1)(-1.00, 1)(-0.14, 0);
14    TERM NM := (-1.00,0)(-0.14, 1)(-0.07, 0);
15    TERM NK := (-0.14,0)(-0.07, 1)( 0.00, 0);
16    TERM ZR := (-0.07,0)( 0.00, 1)( 0.07, 0);
17    TERM PK := ( 0.00,0)( 0.07, 1)( 0.14, 0);
18    TERM PM := ( 0.07,0)( 0.14, 1)( 1.00, 0);
19    TERM PG := ( 0.14,0)( 1.00, 1)( 1.60, 1);
20    RANGE := (-1.60 .. 1.6);
21  END_FUZZIFY
22
23  FUZZIFY drehgeschwindigkeit
24    TERM NM := (-6.30,0)(-0.40, 1)(-0.20, 0);
25    TERM NK := (-0.40,0)(-0.20, 1)( 0.00, 0);
26    TERM ZR := (-0.20,0)( 0.00, 1)( 0.20, 0);
27    TERM PK := ( 0.00,0)( 0.20, 1)( 0.40, 0);
28    TERM PM := ( 0.20,0)( 0.40, 1)( 6.30, 0);
29    RANGE := (-6.3 .. 6.3);
30  END_FUZZIFY
31
32  DEFUZZIFY kraft
```

```
33      TERM NG := (-100,1)(-79, 1)(-58, 0);
34      TERM NM := ( -58,0)(-15, 1)(-10, 0);
35      TERM NK := ( -15,0)(-10, 1)(  0, 0);
36      TERM ZR := ( -10,0)(  0, 1)( 10, 0);
37      TERM PK := (   0,0)( 10, 1)( 15, 0);
38      TERM PM := (  10,0)( 15, 1)( 60, 0);
39      TERM PG := (  60,0)( 80, 1)(100, 1);
40      METHOD: COG;
41      DEFAULT := 0;
42      RANGE := (-100 .. 100);
43    END_DEFUZZIFY
44
45    FUZZY_TIME_OBJECT next
46      FACT 1: (0,1) (1,1);
47      TIME 1: (0,0) (0,1)
                (30,1) (40,0);
48    END_FUZZY_TIME_OBJECT
49
50    RULEBLOCK control
51      AND:MIN;
52      OR:MAX;
53      ACCU:MAX;
54      ACT:MIN;
55      PREDICTION:LINEARITY;
56      RULE 0: (Regeln siehe Tabelle 5)
57    END_RULEBLOCK
58
59    END_FUNCTION_BLOCK
```



3.3. Ergebnisauswertung

Wir vergleichen nun die fünf vorgestellten Regler anhand von acht verschiedener Kriterien. Zuerst stellen wir die Kriterien vor und dann untersuchen wir jeden Regler anhand dieser Kriterien im Vergleich mit den anderen Reglern.

Das einfachste Kriterium ist die Stabilisierungszeit *s*, die ein Regler benötigt um den Stab in eine stabile Lage zu bringen. Stabil bedeutet, dass die Auslenkung φ ab dem Zeitpunkt t_0 immer kleiner als 0.005 (= 0.29°) bleibt. Ob die Auslenkung gegen Null geht oder innerhalb dieser Schwankungsbreite weiter schwingt spielt dabei keine Rolle.

Ein weiteres Kriterium ist das Integral über die Auslenkung φ oder bei diskreter Messung die Summe über die Auslenkungen. Ist diese Zahl klein, so wird der Stab sehr schnell in die Nullstellung gebracht und bleibt auch in dieser. Ein hoher Wert gibt an, dass der Stab entweder langsam ausbalanziert wird oder der Stab nicht ausbalanciert wird beziehungsweise immer noch kleine Schwingungen ausweist.

Das Integral über die Kraft *F* (die Summe über die diskrete Kraft) gibt an, wie viel Energie der Regler zum Stabilisieren und anschließenden Halten in dieser Stellung benötigt. Eine hohe Energie ist ineffizient aber nicht unbedingt langsam.

Die Summe über die Winkelgeschwindigkeit $\dot{\varphi}$ gibt den Weg an, den der Stab zurücklegt. Ein Wert von 100 bedeutet, dass der Stab den kürzest möglichen Weg zurückgelegt hat, also den direkten Weg zum Ziel ohne Überschwinger oder Schwingungen genommen hat.

Die Summe über die quadratische Änderung der Kraft gibt an, welcher Regler sehr oft die Kraft stark ändert. Ein hoher Wert entspricht einem hohen Verschleiß der Hardware.

Nach diesen fünf quantitativen Kriterien folgen drei qualitative Beurteilungen. Der Verlauf der Auslenkung beschreibt, wie sich der Stab verhält und wie seine Bewegung aussieht. Dagegen beschreibt der Verlauf der Kraft, wie der Stab ausbalanziert wird. Das letzte Kriterium beschreibt subjektiv den Aufwand, der zum Entwickeln des jeweiligen Regler nötig war.

In Tabelle 6 sind die wichtigsten Ergebnisse der Simulation zusammengefasst. Hell beziehungsweise dunkel hinterlegt sind die besten beziehungsweise schlechtesten Werte einer Spalte. Ähnliche Werte werden ebenfalls hell beziehungsweise dunkel hinterlegt. Als ähnlich schlecht zählen Werte, welche näher am schlechtesten Wert als am Mittelwert von Maximum und Minimum einer Spalte liegen. Für die besten Werte gilt dies analog. In Abbildung 3 sind die Verläufe der Kraft und der Auslenkung der dazugehörenden Regler darstellt. Das Verhalten der PID-Regler (P, PI und PID) ist im Großen und Ganzen recht ähnlich. Deshalb beschreiben wir sie auch nicht getrennt voneinander, sondern vergleichen diese mit den beiden

		Drehgeschwindigkeit $\dot{\varphi}$				
		NM	NK	ZR	PK	PM
Auslenkung φ	NG	NG	NG	NG	NG	NG
	NM	NG	NM	nm	nk	ZR
	NK	NM	NK	nk	zr	pk
	ZR	NM	nk	ZR	pk	PM
	PK	nk	zr	pk	PK	PM
	PM	ZR	pk	pm	PM	PG
	PG	PG	PG	PG	PG	PG

Tabelle 5: Matrix mit den Fuzzy-Regeln für die Kraft F zum Regeln des Stabwagens. Fuzzy-Terme in Großbuchstaben sind in [Bothe95] definiert, der Rest ist von uns hinzugefügt. Die Fuzzy-Terme reichen von NG (= negativ, groß) über ZR (= zero) bis PG (=positiv, groß)

Fuzzy-Reglern, die jedoch Unterschiede untereinander aufweisen.

Bei quantitativer Untersuchung der Ergebnisse in Tabelle 6 kann man erkennen, dass je nachdem, welches Kriterium wichtig ist entweder die PID- oder einer der Fuzzy-Regler besser sind. Ist es wichtig, dass der Regler schnell ist (Stabilisierungszeit t_0), dann sind die PID-Regler besser, ist es aber wichtig, dass wenig Energie (ΣF) zur Regelung benötigt wird oder die Hardwareabnutzung ($\sqrt{\Sigma}\, dF^2$) gering ist, dann sind die Fuzzy-Regler besser.

Die qualitative Untersuchung der Graphen der verschiedenen Regler in Abbildung 3 zeigt, dass die Auslenkung (dunkler Graph) sich immer recht ähnlich verhält, aber bei den Kräften (heller Graph) größere Unterschiede vorkommen. Dies spiegelt sich auch in den zuvor beschriebenen quantitativen Ergebnissen wieder. Auffällig ist, dass die Fuzzy-Regler insgesamt ein weicheres, aber dadurch auch langsameres Regelverhalten aufweisen. Das Verhalten äußert sich darin, dass die Kräfte bei den PID-Reglern kurz vor Stillstand mehrfach zwischen positiven und negativen Kräften hin und her springen. Dies bewirkt ein schnelles aber abruptes abbremsen. Die Fuzzy-Regler dagegen weisen einen recht glatten Kräfteverlauf mit nur sehr kleinen Sprüngen kurz vorm Stillstand auf. Dadurch wirken kleinere Beschleunigungen auf den Stab und die Hardware wird weniger beansprucht, was aber auch in einem langsameren Abbremsen resultiert.

Ein ebenfalls wichtiges Kriterium ist der Aufwand zum Erstellen eines Reglers. Dazu zählt sowohl die Modellierung als auch die anschließende Implementierung in der Programmiersprache C. Gemeinsam ist beiden Reglern der Programmieraufwand. Der PID-Regler ist in Kapitel 3.1 als Gleichungen gegeben und der Fuzzy-Regler als FCL-Programm in Kapitel 3.2. Der eigentliche Aufwand beim PID-Regler besteht darin, die richtige Stellgröße zu finden, da die Kraft F nicht als Stellgröße funktionieren kann. Die Umrechnung der Änderung der Drehgeschwindigkeit in eine Kraft stellte hier den nicht sehr großen Aufwand dar. Der Aufwand beim Erstellen des Fuzzy-Reglers beschränkt sich darauf, die FCL-Datei mit den Fuzzy-Regeln zu schreiben. Da die Fuzzy-Regeln die Regelung so beschreiben, wie sie ein Mensch auch beschreiben könnte, ist dieser Aufwand sehr gering. Aus diesem Grund geben wir der Entwicklung eines Fuzzy-Reglers einen geringeren Aufwand.

Wir bewerten nun die positiven (+1) und negativen (-1) Kriterien. Als Gesamtergebnis ergibt sich, dass die Regelung mit klassischer Fuzzy-Logik am schlechtesten und die Regelung mit temporaler Fuzzy-Logik am besten abschneidet. Die PID-Regler liegen im Gesamtergebnis zwischen den beiden Fuzzy-Reglern.

4. Schlussfolgerungen

Als Fazit stellt sich heraus, dass zum Entwickeln des PID-Reglers fundiertes Wissen über den zu regelnden Prozess nötig ist. Mit diesem Wissen können die möglicherweise nötigen Umrechnungen für eine geschlossene Regelkette berechnet werden. Der PID-Regler hat als Ausgabe die neue einzustellende Drehgeschwindigkeit. Diese muss zuerst in eine Winklebeschleunigung und anschließend in eine Kraft umgerechnet werden. Der Grund hierfür ist, dass wenn sich das Pendel schon in die richtige Richtung bewegt, dann muss es auch bei einer großen Auslenkung nicht weiter beschleunigt werden. Die Kraft muss also auch bei einer großen Auslenkung Null sein können, sie kann also nicht proportional zur Auslenkung gewählt werden. Würde die Kraft proportional zur Auslenkung gewählt werden, dann

Regler	***Σ Auslenkung φ***	***Σ Kraft F***	***Σ φ̇***	***Stabilisierungszeit t_0***	***√Σ dF²***	***Aufwand***	***Gesamt***
P	37,47	5814	100,8	0,56	189	mittel	+1
PI	37,48	5777	100,8	0,56	186	mittel	+1
PID	37,48	5842	100,8	0,56	210	mittel	+1
FCL	44,29	2353	102,0	0,76	92	wenig	0
TFCL	42,57	2452	101,4	0,74	93	wenig	+2

Tabelle 6: Ergebnisse aus den Experimenten mit den verschiedenen Reglern. Die rechte Spalte gibt das Gesamtergebnis bei hell +1 Punkt und dunkel -1 Punkt an.

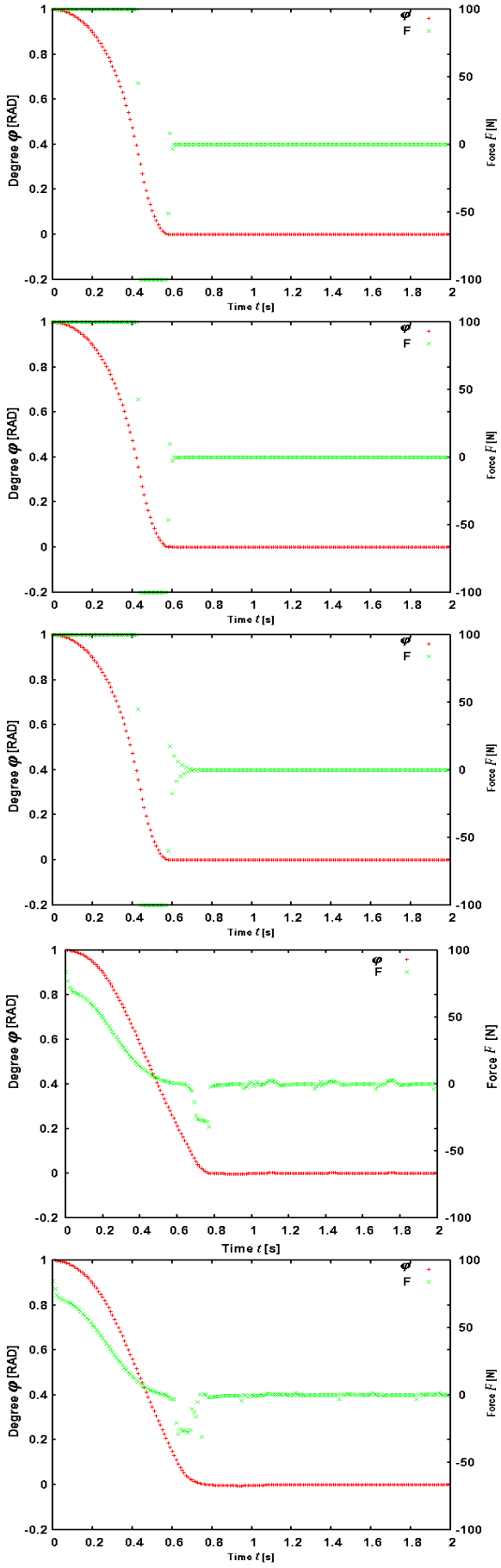


Abbildung 3: Auslenkungs- und Kraftverläufe des Stabwagens für die Regler: P, PI, PID, FCL und TFCL.

würde der P-Regler bei einer großen Auslenkung so lange beschleunigen, bis die Auslenkung Null ist. Erst dann würde auch die Kraft Null werden. Da der Stab aber beschleunigt wurde, besitzt er eine große Geschwindigkeit und schießt über das Ziel hinaus. Da zu Anfangs die Auslenkung klein ist, würde durch den Proportionalanteil nur eine kleine Kraft wirken und den Stab nur sehr langsam abbremsen. Das Ergebnis ist eine sehr große und ungedämpfte Schwingung.

Beim Fuzzy-Regler dagegen genügt es, das Problem als Mensch real oder in Gedanken lösen und auch beschreiben zu können. Eine Lösungsbeschreibung, welche in Fuzzy-Regeln beschrieben ist, kann direkt von einem Fuzzy-Regler verarbeitet werden. Lässt sich das Problem nicht direkt in Fuzzy-Regeln ausdrücken, so muss es zuerst übersetzt werden. Dies kann nur durch einen Experten geschehen, der sich mit Fuzzy-Logik und insbesondere hier mit temporaler Fuzzy-Logik auskennt.

Eine Optimierung ist bei beiden Reglerarten möglich. Beim PID-Regler gibt es drei Parameter, K_P, K_I und K_D die optimiert werden. gute Werte können in einer Simulation seht schnell gefunden werden, aber auch ohne Simulation ist die Parameterfindung nicht schwierig. Beim Fuzzy-Regler dagegen ergeben die Anzahl der Fuzzy-Terme und die Wahl der Stützstellen für die Zugehörigkeitsfunktionen einen so großen Suchraum, dass eine automatische Suche nach guten Zugehörigkeitsfunktionen nicht mehr möglich ist. Hier im Beispiel wären es 24 Parameter bei der gegebenen Anzahl von Fuzzy-Termen. Wird die Anzahl der Fuzzy-Terme noch variiert, dann steigt somit auch die Anzahl der Parameter weiter an. Es bietet sich an, die Fuzzy-Terme und Zugehörigkeitsfunktionen so zu wählen, wie sie sich aus der Beschreibung eines Menschen ergeben. Bessere Parameter kann man dann erhalten, indem man nur wenige Stützstellen verändert oder aber die Stützstellen nur ein wenig verändert. Erstere Methode haben wir mit besseren Erfolg verwendet.

Beide Arten von Reglern benötigen einen Experten, der sich mit dem verwendeten Regler und dem zu regelnden Prozess auskennt. Jedoch benötigt man weniger Zeit um eine Fuz-

zy-Regelbasis aufzustellen, als für die Umrechnung die bei einem PID-Regler nötig sein kann. Auch liest sich die Implementierung in Fuzzy-Logik einfacher (vergleiche PID-Pseudocode mit TFCL-Datei) und bleibt dadurch wartbarer. Der PID-Regler dagegen ist deutlich effizienter in seiner Leistung, was aber auch zu Lasten der Hardwareabnutzung geht. Wie zu sehen ist, haben beide Arten der Regelung ihre Vor- und Nachteile, die je nach Einsatzgebiet wichtiger oder zu vernachlässigen sind.

Als Gesamtergebnis stellt sich jedoch heraus, dass der temporale Fuzzy-Regler etwas besser als die PID-Regler und der klassischer Fuzzy-Regler etwas schlechter als die PID-Regler ist. Dieses Ergebnis hat aber keine Allgemeingültigkeit, sondern trifft nur auf dieses untersuchte Experiment zu. Eine allgemeingültige Aussage kann auch nie getroffen werden, da immer subjektive Entscheidungen bei einem solchen Vergleich nötig sein werden.

Wenn man sicher gehen muss, dass der Regler in jedem Fall stabil ist, dann sei noch auf die Ergebnisse von [Wang93] verwiesen. Dort werden Erweiterungen vorgestellt, mit welchen bei einem Fuzzy-Regler durch adaptives Verhalten sichergestellt wird, dass er immer stabil ist.

5. Literatur

[Bothe95] H.-H. Bothe, *Fuzzy-Logik – Einführung in Theorie und Anwendungen*, Springer-Lehrbuch, 2. Auflage, Berlin Heidelberg, 1995

[Butkiewicz00] B. S. Butkiewicz, *System with Hybrid Fuzzy-Conventional PID Controller,* International Conference on Systems, Man and Cybernetice, Nashville, 2000

[Bovenkamp98] E. G. P. Bovenkamp, J. C. A. Lubbe, *Temporal Reasoning with Fuzzy-Time-Objects*, 4th International Workshop on Temporal Representation and Reasoning, Daytona Beack, Florida, May 10-11, 1997

[Cárdenas02] M. A. Cárdenas Viedma, R. Martín Morales, *Syntax and Semantics for a Fuzzy Temporal Constraint Logic*, Annals of Mathematics and Artificial Intelligence, Volume 36, 2002

[Chang97] X. Chang, W. Li, *Application of hybrid fuzzy logic proportional plus conventional integrall-derivate controller to combustion control of stoker-fired boilers,* International Fuzzy Systems Association, Amsterdam, 2000

[Fick00] A. Fick, H. B. Keller, *Modellierung des Verhaltens Dynamischer Systeme mit erweiterten Fuzzyregeln*, Proceedings 10. Workshop Fuzzy Control des GMA-FA 5.22, Dortmund, Germany, 18. - 20. Oktober 2000

[Giron02] G. Ortega, J. M. Giron-Sierra, *A survey of fuzzy logic control with aerospace applications,* 15th IFAC (International Federation of Automatic Control) World Congress, Barcelona, 2002

[IEC97] IEC TC65/WG 7/TF8, *Fuzzy Control Programming*, International Technical Electronical Commission (IEC), 1997

[Karjoth87] G. Karjoth, *Prozeßalgebra und temporale Logik – angewandt zur Spezifikation und Analyse von komlexen Protokollen*, Diss. Mathematik/Informatik, Universität Stuttgart, 1987

[Schmidt04] T. W. Schmidt, D. Henrich, *Temporal erweiterte Prädikate der Fuzzy-Logik zur Überwachung und Wartung*, Proceedings 14. Workshop Fuzzy Control des GMA-FA 5.22, Dortmund, Germany, 10. - 12. November 2004

[Schmidt05] T. W. Schmidt, D. Henrich, *Inferenz mit Fuzzy-Zeit-Termen*, Proceedings 15. Workshop Fuzzy Computational Intelligence des GMA-FA 5.22, Dortmund, Germany, 16. - 18. November 2005

[Schneider00] U. Schneider, *Das invertierte Pendel*, http://www.informatik.htw-dresden.de/~iwe/Belege/SchneiderPendel/pendel.html, August 2000

[Takagi85] T. Takagi, M. Suego, *Fuzzy identification of systems and its application to modeling and control*, IEEE Transactions on Systems, Man and Cybernetics, Vol. 15, No. 1, pp. 116-132, 1985.

[Wang93] L.-X. Wang, *Stable Adaptive Fuzzy Control of Nonlinear Systems*, IEEE Transactions on Fuzzy Systems, Vol 1, No. 2, May 1993