7 Experimentelle Fuzzyanwendungen

Die Zeit der Fuzzyregelung begann 1973, als Sedrak Assilian und Ebrahim Mamdani versuchten, eine kleine Dampfmaschine mit einem lernfähigen Regelungsprogramm zu kontrollieren. Heute reicht das Spektrum der Fuzzy-geregelten Systeme von Antiverwacklungssystemen in Videokameras über Waschmaschinen und Staubsauger bis zu Zementöfen und ganzen U-Bahn-Anlagen.

Stand in den letzten Kapiteln die Frage im Vordergrund, wie eine Fuzzyregelung überhaupt funktioniert, so widmen wir uns jetzt der Frage, mit welchen Methoden ein Fuzzyprojekt erstellt, ausgetestet und verbessert werden kann: Wie gelangt man an geeignete Mess- und Stellgrößen, wie werden sie fuzzifiziert? Wie hat man die zugehörigen Grundmengen festzulegen? Sollten zuerst die Regeln aufgestellt und dann die Fuzzifizierung entsprechend vorgenommen werden oder umgekehrt? Welche Möglichkeit zur Verbesserung gibt es? An Hand der folgenden Beispiele[1] wollen wir dies eingehend untersuchen (In Klammern stehen die Dateinamen der mitgelieferten Fuzzyprojekte):

- eine Luftballonaufblasmaschine (ballon.fuz)
- das invertierte Pendel (pendel.fuz)
- die Laufkatze (laufkat2.fuz)
- ein Rangierproblem (truck1.fuz und truck2.fuz)

7.1 Die Luftballonaufblasmaschine

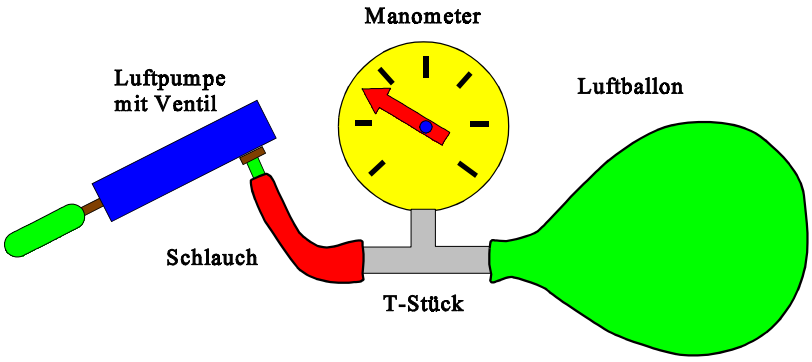
Bei der Gesellschaft für Mathematik und Datenverarbeitung (GMD) in Birlinghoven bei Bonn steht eine Fuzzy-geregelte Maschine, die in der Lage ist, Luftballons aufzublasen. Das kann doch wohl kein großes Problem sein, eine solche Maschine zu konstruieren, werden Sie jetzt vielleicht denken. Schließlich kann doch jedes Kind einen Ballon aufblasen. Tatsächlich kann ein solches Projekt aber einen Ingenieur schon gehörig zum Schwitzen bringen. Warum eigentlich?

Überlegen wir genauer, wie ein Ingenieur eine solche Maschine bauen könnte. Als Luftvorrat benutzt er eine Druckluftflasche; diese versieht er mit einem steuerbaren Ventil. Ferner stattet er die Maschine mit einem Messgerät aus, welche die ausgeströmte Luftmenge messen kann. Nachdem nun eine bestimmte Menge Luft in den Ballon hineingeströmt ist, ist er prall; das Ventil wird dann automatisch geschlossen. Nun geschieht etwas Seltsames: Einige Ballons kann die Maschine erfolgreich aufblasen, aber immer wieder gibt es einen lauten Knall und der Ballon ist geplatzt, noch ehe das Ventil schließt. Offensichtlich ist in diesen Fällen die Luftmenge zu groß. Andere Ballons werden dagegen erst gar nicht prall, wenn das Ventil schließt. Der Grund ist einfach: Luftballons werden in unterschiedlicher Größe und Beschaffenheit angefertigt. Dementsprechend variiert auch die benötigte Luftmenge.

Wie soll die Maschine nun erkennen, wann der Ballon genügend Luft erhalten hat? Jedes Kind könnte hier Rat geben: Wenn der Ballon prall wird, fällt das Pusten immer schwerer; dann muss man aufhören. Also wird unser Ingenieur seine Maschine zusätzlich mit einem Druckmessgerät ausrüsten. Ab einem bestimmten Druck soll die Maschine nun das Ventil schließen.

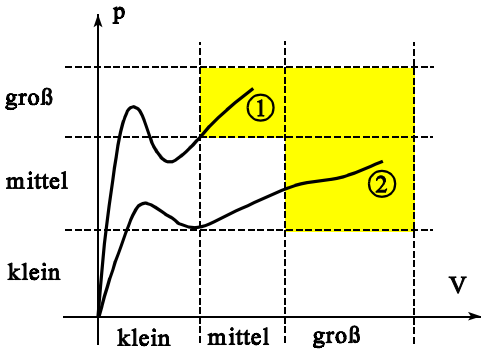
Aber auch nach dieser Verbesserung arbeitet unsere Maschine noch nicht korrekt. Im Gegenteil: Kaum ist ein wenig Luft in den Ballon gelangt, schließt auch schon das Ventil. Warum bläst die Maschine nicht weiter — der Ballon ist doch längst noch nicht prall? Nun, zum Anblasen des Ballons ist ein relativ großer Druck erforderlich. Auch das kann jedes Kind bestätigen. Meist ist es sogar einfacher, einen Ballon zum Platzen zu bringen als ihn anzublasen. Kein Wunder also, dass unsere Maschine beim Anblasen so früh ihre Arbeit beendet.

[1] Im Aufgabenteil finden Sie weitere Projekte zur Übung!



7.1 Mit dieser Versuchsanordnung können p-V-Kennlinien aufgenommen werden.

Was unser Ingenieur braucht, sind Tabellen oder Diagramme, welche ihm sagen, bei welchen Kombinationen von geströmter Luftmenge und Druck das Ventil geschlossen werden soll. Und dann muss er noch einen Weg finden, die Maschine mit diesen Kenntnissen auszustatten und damit arbeiten zu lassen.

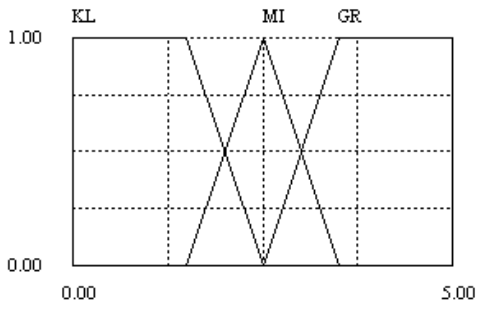


7.2 Typische *p*-*V*-Kennlinien von Luftballons

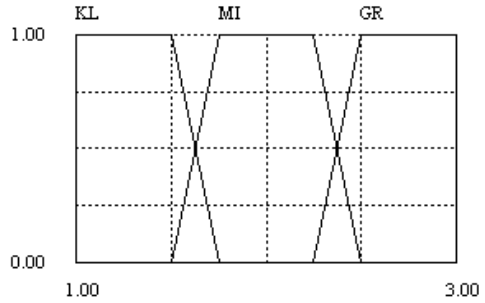
Durch eine Reihe von Versuchen mit der Apparatur aus Abb. 7.1 verschafft sich der Ingenieur einen Überblick über das Verhalten der Ballons beim Aufblasen. Die Werte des Druckes *p* und des Luftvolumens *V* (gemessen in Pumpenhüben) trägt er in ein *p*-*V*-Diagramm ein. Für jeden Ballon erhält er eine Kennlinie, welche im Ursprung des Koordinatensystems beginnt. Dabei platzen Ballons, die sich auf derselben oder einer benachbarten Kennlinie bewegen, nahezu an derselben Stelle im Diagramm. Aus der momentanen Lage des Messpunktes (*p*,*V*) im *p*-*V*-Diagramm lässt sich demnach ablesen, ob ein Platzen bevorsteht oder nicht.

Abb. 7.2 zeigt 2 typische Aufblasvorgänge. Bei dem kleinen Luftballon ① steigt der Druck rasch zu hohen Werten an, sinkt dann aber ab. Bei weiteren Pumpenhüben steigt der Druck schließlich wieder an und der Ballon platzt. Ganz anders verhält sich der große Ballon ②. Zu Beginn ist nur ein relativ kleiner Druck erforderlich; der Ballon platzt erst nach vielen Pumpenhüben, allerdings schon bei mittlerem Druck.

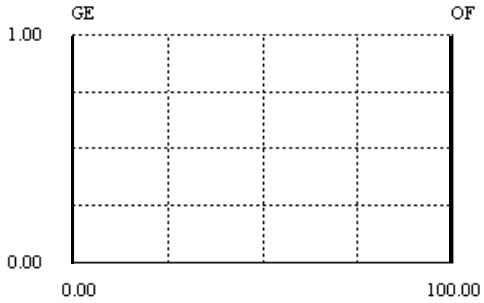
Wie kann man nun der Maschine diese Kenntnisse vermitteln? Natürlich ist es wieder die Fuzzyregelung, die diese Aufgabe erfolgreich meistert. Zur Fuzzifizierung teilt unser Ingenieur sowohl die *V*- als auch die *p*-Achse in drei Bereiche (klein, mittel, groß) ein und stellt sie durch überlappende Fuzzymengen dar (Abb. 7.2). Für das Volumen erhält man z.B. ein Diagramm wie in Abb. 7.3, für den Druck ein Diagramm wie in Abb. 7.4.



7.3 Das Luftvolumen



7.4 Der Druck



7.5 Die Ventilöffnung

Als Ausgangsvariable definiert er die Ventilöffnung wie in Abb. 7.5. Dabei bedeutet 100%, dass das Ventil vollständig geöffnet ist, 0% steht für ein geschlossenes Ventil. Die Rechteckkurven bei 0% und 100% sind dabei so schmal, dass sie in der Graphik nur als Linien erscheinen.

Der Abb. 7.2 entnimmt er die Regeln für das Schließen des Ventils:

```
WENN Luftmenge=mittel UND Druck=groß, DANN Ventil=geschlossen
WENN Luftmenge=groß UND Druck=mittel, DANN Ventil=geschlossen
WENN Luftmenge=groß UND Druck=groß, DANN Ventil=geschlossen
SONST Ventil=offen
```

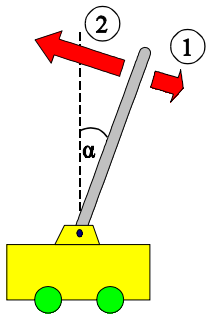
Solange die Luftmenge oder der Druck klein sind, kann gefahrlos weitergepumpt werden. Das Gleiche gilt für den Fall, dass ein mittlerer Druck und eine mittlere Luftmenge vorliegen. Erst wenn bei mittlerer Luftmenge ein großer Druck oder bei mittlerem Druck eine große Luftmenge vorliegen, wird das Ventil geschlossen. (Im ersten Fall handelt es sich um einen kleinen, festen Ballon, im zweiten um einen großen, weichen.)

Sie sollten nun das Projekt ballon.fuz nach den obigen Angaben erstellen oder gleich von der Festplatte laden. Mit Regeln | Testen können Sie dann erkunden, wie das Ventil auf die unterschiedlichen Kombinationen von Druck und Volumen reagiert.

7.2 Das invertierte Pendel

Einen Besenstiel zu balancieren ist keine große Kunst, zumindest für einen Menschen. Kommen Maschinen ins Spiel, wird das schon erheblich schwieriger. Unsere elektronische Schaltung aus Abb. 1.3 war jedenfalls nicht in der Lage, einen solchen Balanceakt zu regeln.

Für Fuzzy ist dieses invertierte Pendel – wie eine Vorrichtung nach Abb. 7.6 in der Fachliteratur genannt wird – zu einem Paradebeispiel geworden. In diesem Rahmen hat sie eine ähnliche Bedeutung gewonnen wie die Fruchtfliege Dro-



7.6 Das invertierte Pendel

sophila für die Genforscher oder die logistic map für die Chaostheoretiker. Immer wieder ist auf Kongressen oder Tagungen das invertierte Pendel als Anwendung der Fuzzyregelung zu finden. Mit Recht, denn kaum eine Fuzzyanwendung ist einerseits so spektakulär und andererseits so einfach in der Konzeption.

Überlegen wir zunächst die benötigten Ein- bzw. Ausgangsvariablen. Als Eingangsvariable benutzen wir natürlich den Winkel α zwischen Pendel und der Vertikalen (Abb. 7.6); als Ausgangsvariable bietet sich die Kraft auf den Wagen an. Diese beiden Variablen reichen allerdings für eine verlässliche Regelung nicht aus. Den Grund dafür können wir uns an Hand der Zeichnung rasch klar machen. Dargestellt sind zwei Situationen mit identischem Ablenkwinkel α. Im ersten Fall bewegt sich das Pendel gerade von links nach rechts. Da das Pendel schon nach rechts ausgelenkt ist und diese Auslenkung im Begriff ist, noch größer zu werden, muss als Korrekturmaßnahme eine relativ starke Kraft nach rechts wirken. Ganz anders verhält es sich beim zweiten Fall. Hier bewegt sich das Pendel gerade von rechts nach links. Ohne dass eine Kraft am Wagen angreifen müsste, nähert es sich von selbst der gewünschten Nulllage. Möglicherweise ist die Bewegung sogar so heftig, dass sogar eine geringe Kraft nach links erforderlich wird, damit das Pendel nicht über die Nulllage hinausschießt. [1]

Neben dem Winkel muss also auch der Bewegungszustand des Pendels bei der Fuzzyregelung berücksichtigt werden. Diesen Bewegungszustand beschreiben wir durch die *Winkelgeschwindigkeit*. Sie gibt an, um wieviel Grad der Winkel α sich in 1 s ändert. Bei einer konstanten Winkelgeschwindigkeit von 3° pro s verändert sich α demnach in 1 s um 3°, in 2 s um 6° usw. Positive Winkelgeschwindigkeiten sollen Bewegungen nach rechts und negative Bewegungen nach links anzeigen.

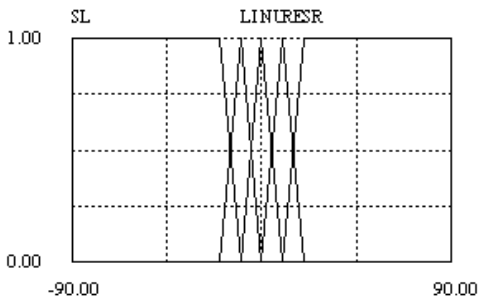
Wie sehen nun die Fuzzyvariablen für den Ablenkwinkel, die Winkelgeschwindigkeit und die Kraft im einzelnen aus? Unsere vorangehenden Überlegungen legen nahe, die Fuzzyvariable "Winkelgeschwindigkeit" durch die fünf Terme "stark negativ", "negativ", "null", "positiv" und "stark positiv" zu beschreiben. Zur Festlegung der Intervallgrenzen schätzen wir die maximale Winkelgeschwindigkeit ab. Lässt man das Pendel mit der Standardlänge $l = 2{,}5$ m aus der Startposition $\alpha = 10°$ einfach fallen, so braucht es ca 3 s, um die 90°-Marke zu überschreiten. (In diesem Moment hält die Simulation automatisch an.) Die mittlere Winkelgeschwindigkeit ist also $80°/3\,s \approx 27°/s$. Da die momentane Winkelgeschwindigkeit am Ende dieses Vorgangs um einiges größer ist, wählen wir als Intervallgrenzen −200°/s und +200°/s. Sollte ein Messwert einmal außerhalb der Grundmenge liegen, so ordnet das *Fuzzy*-Programm bei der Fuzzifizierung aus programmtechnischen Gründen lauter Nullen als Zugehörigkeitsgrade für die einzelnen Terme zu; es erfolgt keine Fehlermeldung. Dies kann bei der Simulation zu einem fehlerhaften Verhalten führen, wenn die Grenzen der Grundmenge überschritten werden. Die Grundmenge sollte also nie zu knapp bemessen sein. Vor allem bei kleinen Pendellängen muss man mit recht großen Winkelgeschwindigkeiten rechnen.

Die Intervallgrenzen der Fuzzyvariablen "Ablenkwinkel" sind selbstverständlich −90° und +90°. Auch diese Fuzzyvariable beschreiben wir durch fünf Terme, die wir genauso wie die Terme der Winkelgeschwindigkeit bezeichnen. Für die Fuzzyvariable "Kraft" sind größere Beträge als bei der Laufkatze erforderlich, da rasche Reaktionen nötig sind. Wir wählen [-20 N; +20 N] als Grundmenge. Auch hier könnte man eine ähnliche Aufteilung in fünf Terme vornehmen; die Erfahrung zeigt aber, dass die drei Terme "links", "null" und "rechts" hier vollkommen ausreichen. Alle weiteren Informationen zur Fuzzifizierung können Sie den Tabellen 7.1 bis 7.3 und den Abbildungen 7.7 bis 7.9 entnehmen.

[1] Sollten Sie schon wie in Aufgabe 6 aus Kapitel 4 versucht haben, eine Fuzzyregelung für das invertierte Pendel mit dem Ablenkwinkel als einziger Eingangsvariable zu ersinnen, werden Sie wahrscheinlich zu keiner Lösung gelangt sein, die den Test mit unserer Simulation des Programms *Fuzzy* zufriedenstellend bestehen kann. Jetzt kennen Sie den Grund dafür!

	a	b	c	d	h
SL	-90	-90	-20	-10	1
LI	-20	-10	-10	0	1
NU	-10	0	0	10	1
RE	0	10	10	20	1
SR	10	20	90	90	1

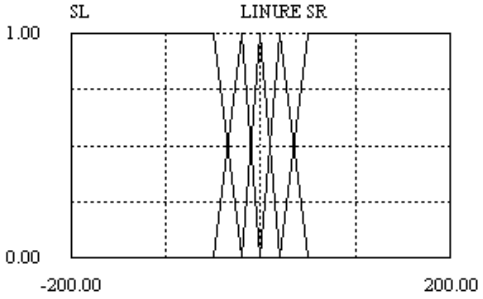
Tab. 7.1 Der Ablenkwinkel in °



7.7 Diagramm für den Ablenkwinkel

	a	b	c	d	h
SL	-200	-200	-50	-20	1
LI	-50	-20	-20	0	1
NU	-20	0	0	20	1
RE	0	20	20	50	1
SR	20	50	200	200	1

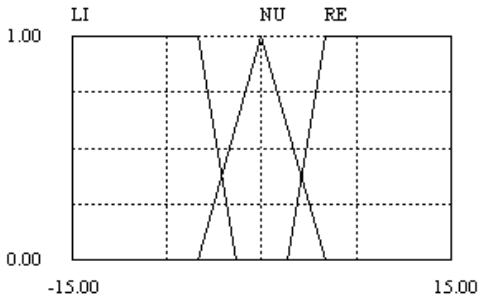
Tab. 7.2 Die Winkelgeschwindigkeit in °/s



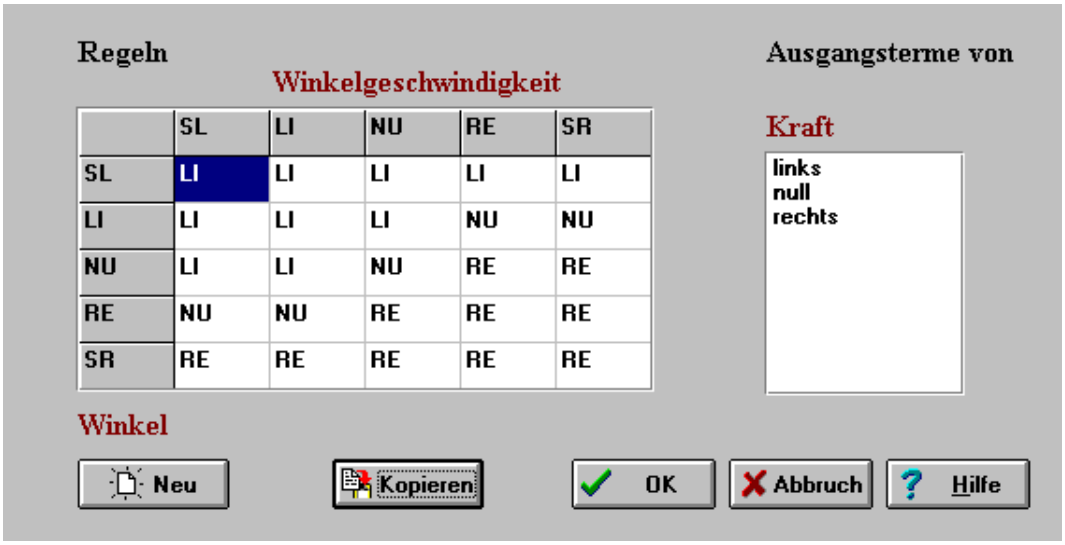
7.8 Diagramm für die Winkelgeschwindigkeit

	a	b	c	d	h
LI	-15	-15	-5	-2	1
NU	-5	0	0	5	1
RE	2	5	15	15	1

Tab. 7.3 Die Kraft in N



7.9 Diagramm für die Kraft



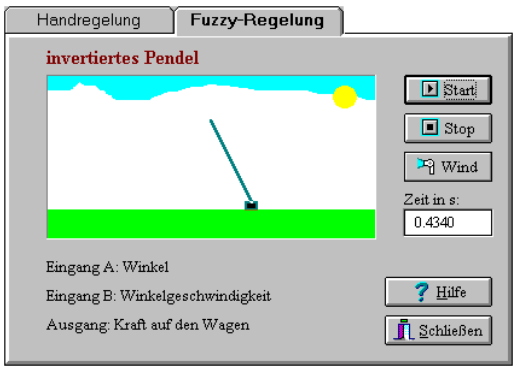
7.10 Die Regeln für das invertierte Pendel

Wenden wir uns nun den Regeln zu: Die Erläuterungen zu Abb. 7.6 führen unmittelbar zu den beiden Regeln:

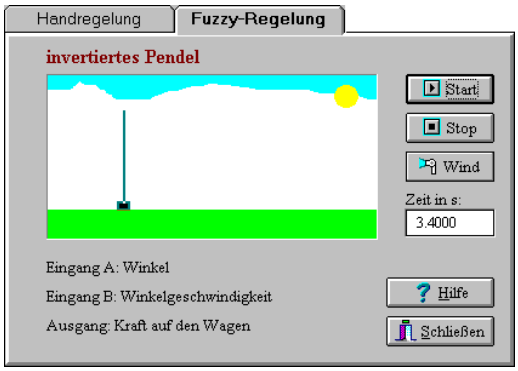
> WENN Winkel=rechts UND Winkelgeschwindigkeit=stark links, DANN Kraft=null
> WENN Winkel=rechts UND Winkelgeschwindigkeit=rechts, DANN Kraft=rechts

Mit weiteren, ähnlichen Überlegungen gelangen wir schließlich zu den Regeln aus Abb. 7.10.

Geben Sie nun diese Daten dem *Fuzzy*-Programm ein und testen Sie das Projekt mit Simulation| invertiertes Pendel| Fuzzyregelung. Erstaunlich schnell bringt die Fuzzyregelung unser Pendel aus der Startablenkung von 10° in die Vertikale. Fast hat man den Eindruck, dass das Simulationsprogramm seine Arbeit eingestellt hat, so ruhig bleibt das Pendel stehen. Mit dem Seitenwind-Knopf stellen Sie aber rasch fest, dass die Fuzzy-Regelung direkt auf äußere Einflüsse reagiert. In diesem Fall führt der Seitenwind von rechts dazu, dass der Pendelwagen nach links beschleunigt wird. Wenn



7.11 Die Startablenkung betrug hier -30°.



7.12 Schon nach wenigen Sekunden bleibt das invertierte Pendel vertikal stehen.

der Wind über eine lange Zeit wirkt, verschwindet unser Pendel daher aus dem Blickfeld. Dass das Pendel dabei trotzdem nicht umkippt, erkennen Sie an der weiterlaufenden Zeitangabe. Diese hört nämlich erst dann automatisch auf, wenn der Betrag des Ablenkwinkels 90° überschreitet. Wie Sie unsere Fuzzyregelung weiteren Test unterziehen können, erfahren Sie in der Aufgabe 1 am Ende des Kapitels.

7.13 Eine Szene aus dem Video ipend1.avi

7.3 Kombination von Fuzzyregelungen

Versucht man, mit der Fuzzyregelung des letzten Abschnitts ein reales invertiertes Pendel zu balancieren, gelingt dies nur bedingt: Nach einigen Sekunden setzt bei dem Wagen nämlich eine Driftbewegung ein. Eine solche Bewegung ist in dem Video “ipend1.avi” festgehalten. Dieses Video können Sie sich anschauen, indem Sie die Datei “ipend1.avi” aus dem Explorer anklicken. Es lässt sich aber auch direkt aus dem *Fuzzy*-Programm über Experiment | Videos betrachten.

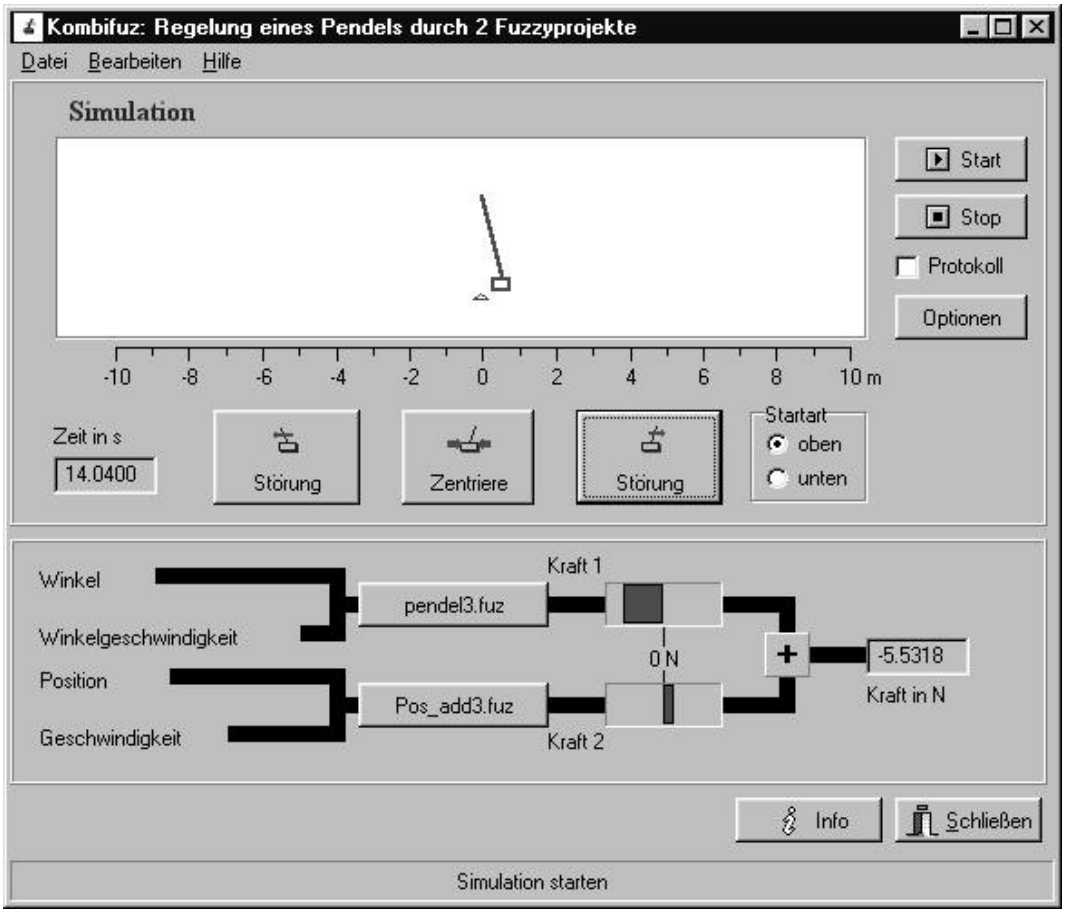
Woher stammt nun diese Driftbewegung? Zur Klärung dieser Frage nehmen wir einmal an, der Wagen hätte gerade durch eine Bewegung nach rechts das Pendel wieder in die vertikale Lage gebracht. Jetzt meldet die Fuzzyregelung "Kraft=Null". Der Wagen sollte nun gleichmäßig weiterfahren; in Wirklichkeit wird er aber durch die Reibung leicht gebremst, und das Pendel kippt dementsprechend wieder geringfügig nach rechts. Die Fuzzyregelung reagiert darauf, indem sie über den Motor eine Kraft nach rechts auf den Wagen ausüben lässt. Auf diese Weise läuft der Wagen immer ein wenig hinter der Pendelspitze her: Er driftet weg, in unserem Fall nach rechts.

Nun stellt sich natürlich die Frage, warum dieses Verhalten bei unseren Simulationsversuchen bislang nicht beobachtet wurde. Der Grund liegt in der Art und Weise, wie die Dynamik des Pendels im Programm *Fuzzy* implementiert wurde: Um die Simulation auch auf langsameren Rechnern ausführen zu können, waren die Bewegungsgleichungen des Pendels und auch deren Lösung stark vereinfacht worden.[1] Diese Vereinfachungen führen Ungenauigkeiten mit sich. Deswegen können wir auch nicht ausschließen, dass unser simuliertes Pendel sich ein wenig anders verhält als ein reales.

Selbstverständlich könnten unsere Probleme auch andere Ursachen haben, aber die obige Argumentation lässt sich untermauern. Dazu soll das Win95-Programm *Kombifuz* zum Einsatz kommen, welchem ein wesentlich genaueres Modell des Pendelsystems zugrunde liegt.[2] Sie können es aus dem Explorer oder auch direkt aus dem *Fuzzy*-Programm über Simulation | kombifuz starten. Dass dieses Programm das Pendel besser simuliert, zeigt das folgende kleine Experiment: Starten Sie das Programm, laden Sie aber noch kein Fuzzyprojekt; während der Simulation übt dann der Motor keine

[1] So wird z.B. die Masse des Pendels gegenüber der Masse des Wagens vernachlässigt (vgl. Anhang). Weiterhin wird die Lösung des Differenzialgleichungssystems nach dem Euler-Verfahren vorgenommen. Dieses Verfahren erfordert zwar nur wenig Rechenleistung, es ist als Verfahren 1. Ordnung aber recht ungenau. So lässt es ein frei schwingendes Pendel nach und nach aufschaukeln. Um dies zu verhindern, waren Dämpfungsglieder in die Bewegungsgleichungen eingefügt worden. Diese Dämpfungsglieder brachten offensichtlich auch einen didaktischen Vorteil mit sich: Sie verhinderten das Driften unseres invertierten Pendels!

[2] Im Programm *Kombifuz* wurden keinerlei Näherungen bei den Bewegungsgleichungen vorgenommen. Außerdem wurde zur Lösung ein modifiziertes Euler-Verfahren benutzt, welches von 2. Ordnung ist.



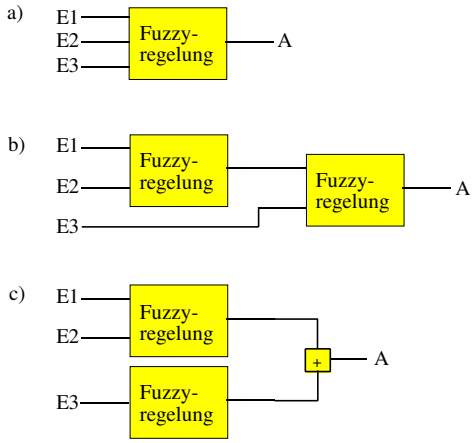
7.14 Das Programm *Kombifuz*

Kraft auf den Wagen aus. Wenn Sie nun den Start-Knopf drücken, beginnt die Simulation. Bringen Sie nun das System aus dem Gleichgewicht, indem Sie einen der beiden Störung-Knöpfe betätigen. Das Pendel kippt dann zur Seite; gleichzeitig bewegt sich der Wagen nach dem Rückstoßprinzip in die entgegengesetzte Richtung. Eine solche Bewegung des Wagens kann bei der vereinfachten Simulation nicht beobachtet werden: Da hier die Pendelmasse gegenüber der Wagenmasse vernachlässigt wurde, kann bei der vereinfachten Simulation das Pendel den Wagen genausowenig in Bewegung setzen wie ein Floh beim Sprung die Erde wegstoßen kann.

Tatsächlich weist das invertierte Pendel im *Kombifuz*-Programm dieselbe Driftbewegung wie im Realexperiment auf. Sie können sich davon überzeugen: Laden Sie das Projekt pendel5b.fuz, indem Sie den Knopf mit der Aufschrift "Projekt 1 öffnen" anklicken. Dieses Projekt ist ganz ähnlich konzipiert wie das Projekt pendel.fuz aus dem letzten Abschnitt. Sie können nun sehen, wie nach jeder Störung das invertierte Pendel ausbalanciert wird und anschließend zu einer Seite wegdriftet.

Diese Driftbewegung muss natürlich verhindert werden. Im Realexperiment hätte sie schließlich fatale Folgen: Der Wagen könnte vom Tisch fallen oder von den Versorgungskabeln abrupt gebremst werden. Wünschenswert wäre demnach, dass das Pendel immer wieder zum Ausgangspunkt zurückkehrt. Dies kann allerdings nur gelingen, wenn wir neben dem Ablenkwinkel und der Winkelgeschwindigkeit die Position des Wagens als weitere Eingangsgröße einführen.

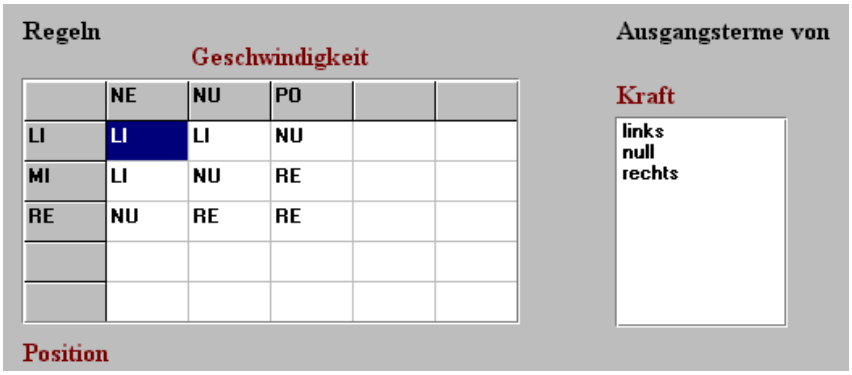
Damit benötigen wir jetzt eine Fuzzyregelung mit mehr als 2 Eingangsvariablen. Verschiedene Methoden bieten sich an, derartige Fuzzyregelungen zu konstruieren (Abb. 7.15): Die erste besteht aus einer einzigen Fuzzyregelung mit mehr als 2 Eingängen. Bei solchen Fuzzyregelungen können die Regeln nicht mehr in Form von (zweidimensionalen) Tabellen eingegeben werden. Andere Formen wie dreidimensionale Tabellen oder eine textmäßige Eingabe der Regeln wären erforderlich. Diesen Weg wollen wir hier nicht weiter verfolgen. Das ist aber auch gar nicht nötig, denn es gibt auch andere Möglichkeiten, mehr als zwei Eingangsvariablen fuzzymäßig zu bearbeiten. Die Idee besteht darin, mehrere Fuzzyregelungen mit jeweils zwei oder auch nur einer Eingangsvariablen zu kombinieren. Solche Fuzzyregelungen können z.B. hintereinandergeschaltet (kaskadiert) oder überlagert werden (Abb. 7.15b und 7.15c).



7.15 Fuzzyregelungen mit mehr als 2 Eingängen

Für unser invertiertes Pendel benutzen wir die Überlagerungsmethode: Hier wird dem bereits bekannten Projekt pendel5b.fuz ein zweites, positionsabhängiges Fuzzyprojekt additiv überlagert. Genauer bedeutet dies: Das Fuzzyprojekt pendel5b.fuz berechnet zu jedem Messwertpaar aus Ablenkwinkel und Winkelgeschwindigkeit eine Kraft F_1, und das zweite Fuzzyprojekt ermittelt eine weitere Kraft F_2. Diese beiden Kräfte werden addiert; die resultierende Kraft wird schließlich dem Wagen als Stellwert zugeführt.

Wie funktioniert diese Überlagerungsregelung eigentlich? Abb. 7.16 zeigt die Regeln des Projektes pos_add5.fuz, welches für die zusätzliche Kraft F_2 zuständig ist. Zunächst fällt auf, dass dieses Projekt neben der Position eine weitere Eingangsgröße, nämlich die Geschwindigkeit des Wagens, besitzt. Sodann wird deutlich, dass die Positionsregelung nur indirekt wirkt: Wenn sich der Wagen weit von der Ausgangsposition entfernt, sagen wir nach rechts, zündet die Regel "Position=rechts UND Geschwindigkeit=rechts"; sie liefert eine Kraft F_2, die nach rechts (!) wirkt. Die resultierende Kraft $F_1 + F_2$ kann somit das Pendel jetzt nicht mehr in die vertikale Lage führen; vielmehr wird es leicht nach links geneigt. Nun tritt die Regelung pendel5b.fuz in Aktion: Um das Pendel vertikal zu stellen, muss sie eine Kraft nach links wirken lassen; diese ist es, welche den Wagen wieder zur Ausgangslage zurückführt. Ist hingegen die Regel "Position=rechts UND Geschwindigkeit=links" aktiv, wirkt keine zusätzliche Kraft F_2. Würde in diesem Fall weiterhin eine nach rechts gerichtete,



7.15 Die Regeln des Projekts pos_add5.fuz

zusätzliche Kraft F_2 wirken, würde sich das System unweigerlich aufschaukeln. Damit ist auch klar, dass wir auf die Geschwindigkeit als weitere Eingangsvariable hier nicht verzichten können.

Laden Sie nun mit dem Knopf Projekt2 öffnen auch das Projekt pos_add5.fuz. Starten Sie anschließend mit dem Start-Knopf die Simulation. Mit den beiden Störung-Knöpfen können Sie nun das Pendel wieder nach Belieben ablenken; bei jedem Anklicken wird das Pendel um einen vom Zufall bestimmten Winkel zwischen 0° und 20° im bzw. gegen den Uhrzeigersinn abgelenkt. Sofort wird die Fuzzyregelung tätig: Nach einem kurzen Balanciermanöver bewegt sich der Wagen immer wieder zur Ausgangslage zurück.

Die Kombination von zwei Fuzzyregelungen hat ausgereicht, unser Driftproblem zu lösen. In der Tat besticht diese Vorgehensweise durch ihrer Schlichtheit: Sie geht von einem bereits bestehenden Projekt aus; dieses Projekt erfasst das bisherige Expertenwissen, allerdings ohne das Driftproblem zu berücksichtigen. Dieser Mangel wird durch ein zusätzliches Projekt behoben. Diese Vorgehensweise ist einfacher, als ein einziges Fuzzyprojekt mit drei oder vier Eingangsvariablen zu erstellen. In der Tat könnte man hier gleich Hunderte von Regeln formulieren, und man müsste eine sorgfältige Auswahl vornehmen. So ist es nicht erstaunlich, dass auch Entwickler kommerzieller Fuzzyanwendungen auf Überlagerungen oder Verkettungen von Fuzzyregelungen zurückgreifen.

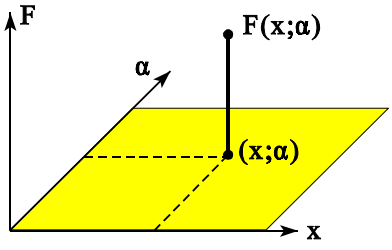
7.4 Noch einmal die Laufkatze

Bereits im letzten Kapitel wurden für die Laufkatze Fuzzyvariablen und ein zugehöriges Regelsystem ausführlich vorgestellt. Hier nun wollen wir der Frage nachgehen, ob sich dieses Fuzzyprojekt noch verbessern lässt. Mit anderen Worten: Können wir unsere Laufkatze noch schneller ans Ziel gelangen lassen?

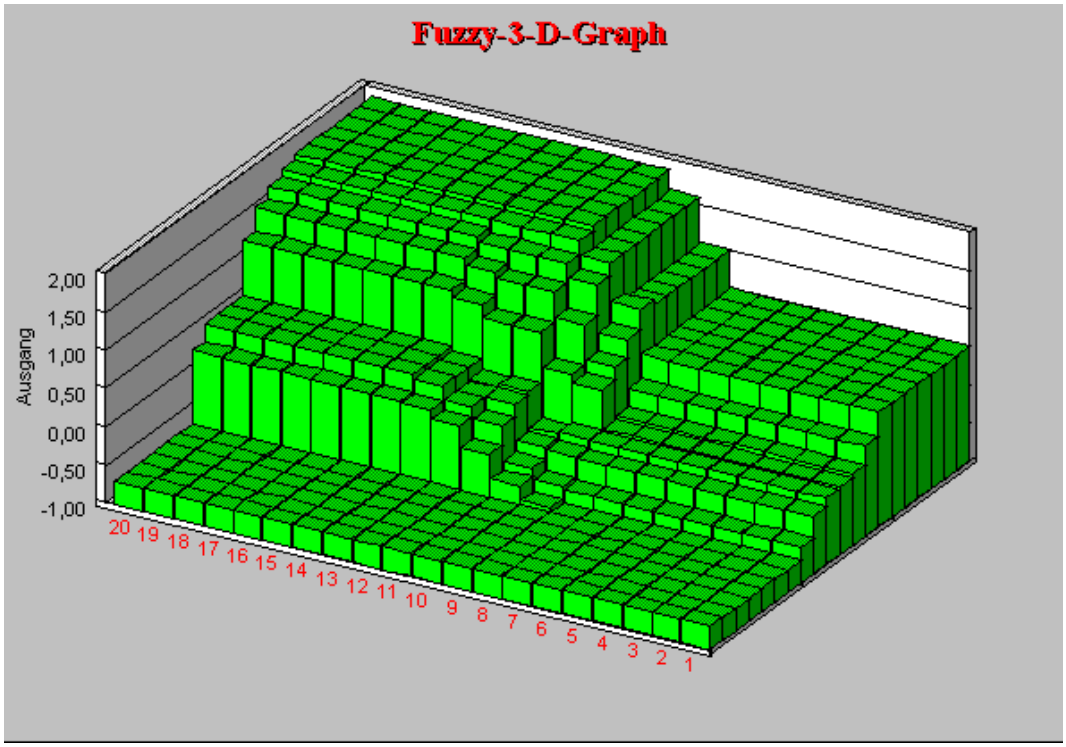
Verschiedene Möglichkeiten bieten sich hier an. Zunächst einmal könnte man die Winkelgeschwindigkeit als weitere Eingangsgröße einführen. Beim invertierten Pendel haben wir diese Größe ja schon mit Erfolg eingesetzt. In diesem Fall würde wieder eine Fuzzyregelung mit mehr als zwei Eingangsvariablen vorliegen. Wie man hier verfahren könnte, wurde im letzten Abschnitt bereits ausführlich dargestellt. Wir werden diese Möglichkeit deswegen hier nicht mehr weiter verfolgen.

Eine andere Möglichkeit zur Verbesserung einer Fuzzyregelung besteht in der Verfeinerung der benutzten Fuzzyvariablen. Darunter verstehen wir folgendes: Jede Fuzzyvariable besteht aus einer gewissen Anzahl von Termen. Bei einer Verfeinerung wird diese Anzahl der Terme erhöht. Natürlich müssen die Regeln entsprechend angepasst werden. Eine derartige Verfeinerung hatten wir schon im letzten Kapitel vorgenommen, als wir der Fuzzyvariablen "Position" den Term "nah" hinzufügten.

Das FAT-Theorem (*Fuzzy-Approximation-Theorem*) garantiert, dass man auf diese Weise tatsächlich jedes Regelungsverhalten mit der gewünschten Genauigkeit erreichen kann. Dies müssen wir etwas genauer erläutern: Sicherlich haben Sie im Regel-testen-Fenster schon den 3-D-Graph-Knopf entdeckt. Seine Bedeutung ist schnell erklärt. Zu jedem Paar $(x;\alpha)$ von Messwerten berechnet das *Fuzzy*-Programm einen Stellwert F. Diese Zuordnung kann man graphisch darstellen. Über jedem Punkt der x-α-Ebene wird ein Stab errichtet, dessen Länge den Stellwert F repräsentiert (Abb. 7.17). Die Enden sämtlicher Stäbe bilden eine Fläche in einem dreidimensionalen Koordinatensystem, den Graphen der Zuordnung $(x;\alpha) \rightarrow F$. Selbstverständlich kann unser *Fuzzy*-Programm nicht für alle Paare (x,α) den zugehörigen Stellwert F berechnen. Es begnügt sich mit der Berechnung der Stellwerte für 20



7.17 So entsteht der Graph der Zuordnung $(x;\alpha) \rightarrow F$.



7.18 Der 3-D-Graph der Zuordnung (x;α)→F bei der Fuzzyregelung des Projekts laufkat2.fuz

mal 20 solcher Paare (*x*;*α*) und stellt die Ergebnisse durch Balken wie in Abb. 7.18 dar. Den Blickwinkel auf diesen Graphen können Sie übrigens mit dem -Knopf einstellen.

Da eine solche Fläche für jede Kombination von Messgrößen den zugehörigen Stellwert angibt, legt sie das Regelungsverhalten eindeutig fest. Das FAT-Theorem besagt nun, dass sich jede denkbare Fläche beliebig genau durch eine Fläche annähern lässt, die aus einer Fuzzyregelung hervorgeht.[1] Dabei hängt die erreichte Genauigkeit von der Anzahl der benutzten Terme ab.

Das FAT-Theorem gibt uns keine Auskunft darüber, wie die Fuzzyvariablen zu konstruieren sind, die zu der gewünschten Fläche führen, erst recht schweigt es sich darüber aus, wie man überhaupt an diese anzustrebende Fläche gelangt. Das ist ja gerade einer der großen Vorteile von Fuzzy, auf einfache und überschaubare Weise solche Zuordnungen zwischen Messwerten und Stellwerten zu erhalten. Eben diese Übersichtlichkeit nimmt aber gewaltig mit zunehmender Verfeinerung der Fuzzyvariablen ab.

Deswegen wollen wir hier für die Verbesserung unserer Laufkatzenregelung nun einen dritten Weg vorstellen.

Nach der Devise "Weniger ist oft mehr" versuchen wir, mit einer geringen Anzahl von Termen auszukommen. Dafür werden wir uns mit der Konstruktion der Zugehörigkeitsfunktionen etwas mehr Mühe geben als bislang. Bei einer kleineren Termanzahl fällt es natürlich leichter, das Resultat solcher Veränderungen zu kontrollieren und das System somit schrittweise zu verbessern.

[1] Es wird hier stillschweigend vorausgesetzt, dass die zugrundeliegende Funktion stetig ist.

Als Terme für die Ablenkung benutzen wir "links", "Mitte" und "rechts", als Terme für die Kraft "links", "Null", "rechts" und "stark rechts". Auf den vierten Kraftterm "stark rechts" wollen wir nicht verzichten, weil er für eine große Beschleunigung in der Phase nötig ist, in der die Laufkatze noch weit vom Ziel entfernt ist. Auch die Fuzzyvariable "Position" erhält vier Terme: "fern", "nah", "Ziel" und "zu weit". Versucht man ohne den "nah"-Term auszukommen, kann die Laufkatze nicht sanft genug abgebremst werden, die Last gerät in heftige Schwingungen, und man erreicht das Ziel erst nach langer Zeit.

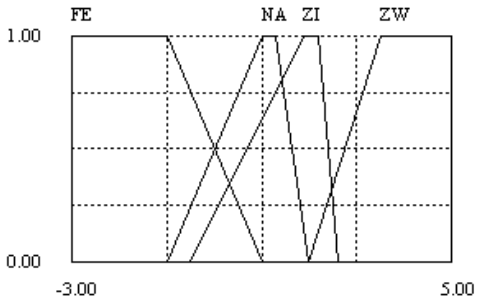
Wie kann man nun diese Abbremsphase geschickt gestalten? Offensichtlich ist hier von entscheidender Bedeutung, wie die Zugehörigkeitsfunktionen für die Terme "nah" und "Ziel" aussehen. Abb. 7.19 ist das Ergebnis von mehreren Versuchen. Auffällig ist die starke Asymmetrie der Graphen von "nah" und "Ziel". Dahinter steckt folgende Idee: Der Übergang von "fern" bis "Ziel" soll möglichst sanft erfolgen. Wenn die Laufkatze allerdings über das Ziel hinausgeschossen ist, soll die Regelung schneller reagieren. Weil die Laufkatze in diesem Stadium nicht mehr weit vom Ziel entfernt ist, werden die Kräfte auch nie sehr lange wirken; starke Pendelbewegungen sind somit ausgeschlossen.

Die anderen Fuzzyvariablen (Abb. 7.20 und 7.21) sind dagegen eher konventionell gehalten. Allenfalls bei der Kraft fällt die unterschiedliche Gewichtung der Terme "rechts" und "links" auf. Auch die Regeln (Abb. 7.22) lassen sich kaum anders fassen. Lediglich die Regel

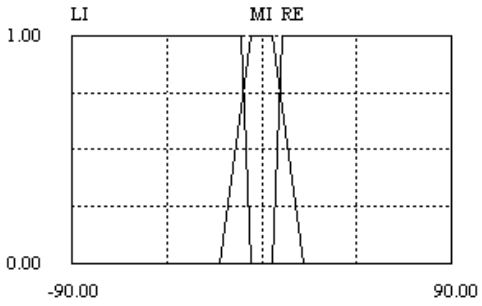
> WENN Position=nah UND Ablenkung=links,
> DANN Kraft=null

bedarf einer gesonderten Überlegung: Wenn das Pendel kurz vor dem Ziel nach links ausgelenkt ist, also die Last hinterherhinkt, dann soll keine Kraft mehr auf die Laufkatze wirken. Auf diese Weise besteht die Chance, dass die Last das Pendel wieder einholt und damit die Ablenkung kleiner wird.

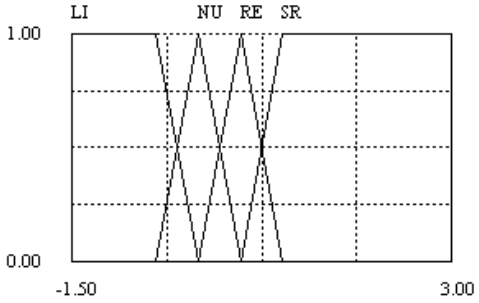
Geben Sie nun das Fuzzyprojekt wie angegeben ein oder laden Sie direkt die Datei laufkat2.fuz. Wenn Sie nun die Laufkatzensimulation mit Simulation | Laufkatze | Fuzzyregelung starten, werden Sie erkennen, dass im Vergleich zum



7.19 Die Position in m



7.20 Der Ablenkwinkel in °



7.21 Die Kraft in N

Regeln

Winkel

	LI	MI	RE		
FE	SR	SR	SR		
NA	NU	RE	RE		
ZI	LI	NU	RE		
ZW	LI	LI	LI		

Position

7.22 Die Regeln für die verbesserte Laufkatzenregelung

Projekt laufkat1.fuz der Abbremsvorgang viel günstiger erfolgt und die Ampel dementsprechend früher von rot auf grün springt.

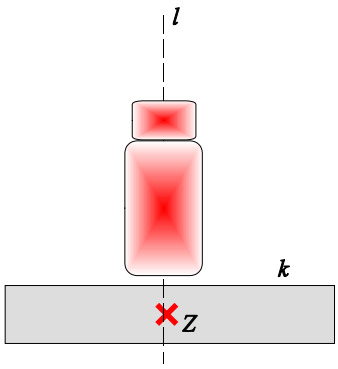
7.5 Optimieren von Fuzzyregelungen

An drei Beispielen haben Sie nun sehen können, wie man Fuzzyprojekte erstellen und verbessern kann: Zuerst werden die relevanten Mess- und Stellgrößen ermittelt. Diese werden durch Fuzzyvariablen dargestellt. Dabei ist es zweckmäßig, die Grundmengen jeder Fuzzyvariablen zunächst in nahezu gleich große, sich überlappende Fuzzymengen zu zerlegen. Gerade bei den Eingangsvariablen ist die Überlappung von entscheidender Bedeutung: Gehört ein Element der Grundmenge nämlich zu keiner Fuzzymenge, so kann kein zugehöriger Stellwert mit Hilfe der Regeln ermittelt werden. Für die Ausgangsvariable ist die Überlappung nicht unbedingt erforderlich. Hier ist es wichtig, im Auge zu behalten, dass die Größe der Fläche unter den einzelnen Graphen die Position des Schwerpunktes und damit den Stellwert bestimmt.

Nachdem nun die Ein- und Ausgangsgrößen fuzzifiziert worden sind, werden die Regeln aufgestellt. Anschließend wird die Fuzzyregelung getestet. Gegebenenfalls schaut man sich dazu auch die Stellwerte zu bestimmten Kombinationen von Eingangswerten mit Hilfe von Regeln|Testen an. Sollte die Fuzzyregelung noch nicht zufriedenstellend sein, verändert man einzelne Regeln. Manchmal kann es auch notwendig sein, die Fuzzymengen zu modifizieren. Meist ist dies allerdings nur für eine abschließende Feinabstimmung erforderlich. Macht man die Flanken im Fuzzydiagramm z.B. etwas steiler, wird das System härter geregelt werden, bei flacheren Flanken fällt die Regelung weicher aus.

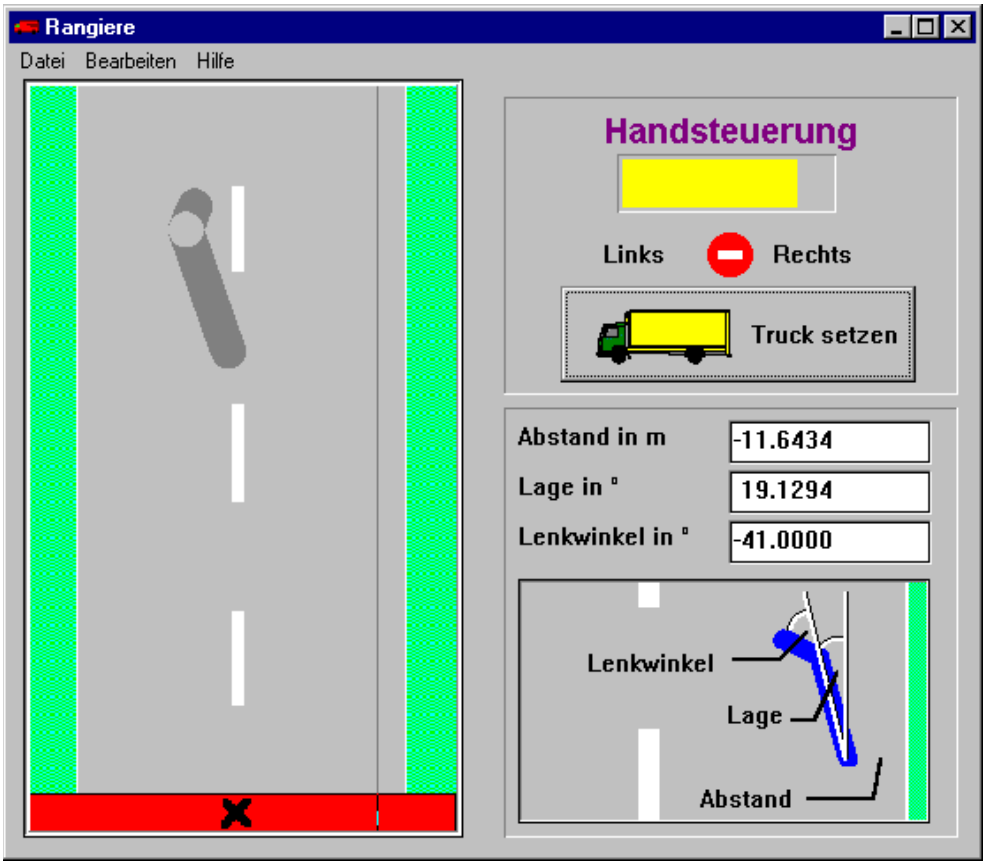
7.6 Rangierprobleme

Oft müssen LKWs rückwärts an eine Laderampe herangefahren werden. Sie sollen schließlich so zum Stehen kommen, dass ihre Längsache *l* nicht nur durch den Zielpunkt *Z*, sondern obendrein auch senkrecht zur Kante *k* der Laderampe verläuft (Abb. 7.23). In anderen Fällen geht es darum, einen LKW rückwärts an eine Bordsteinkante zu parken; der LKW soll also diesmal möglichst dicht an eine Kante und parallel dazu gesetzt werden. Beide Rangierprobleme sind nicht ganz einfach zu lösen. Ungeübte Fahrer schaffen dies auch nicht in einem Zug, d.h. sie müssen bei diesem Vorgang mehrfach die Bewegungsrichtung wechseln. Dabei ist das Einparken an der Bordsteinkante noch etwas schwieriger als das Heranfahren an die Laderampe; schließlich sollte der Bürgersteig sowie angrenzende Vorgärten nach Möglichkeit nicht mit dem LKW befahren werden.



7.23 LKW an der Laderampe

Da Ihnen – ebenso wie mir – vermutlich kein LKW für Rangierexperimente zur Verfügung steht, habe ich ein Programm geschrieben, mit dem genau diese beiden Probleme simuliert werden können. Dieses *Rangiere*-Programm kann vom *Fuzzy*-Programm aus durch Simulation | Rangiere aufgerufen werden. Der LKW ist dabei durch einen blauen Balken dargestellt; ein kurzer Balken an seinem Ende deutet die Stellung der Räder an der Vorderachse an (Abb. 7.24). Sie kann im Handbetrieb mit der Maus gesteuert werden. Wie dies geschieht, soll jetzt genauer dargelegt werden.



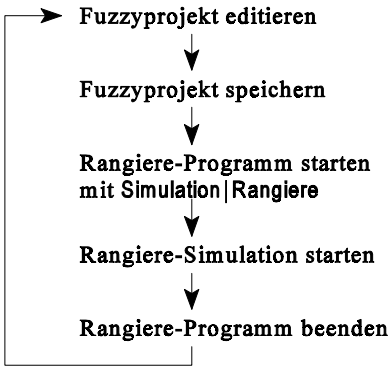
7.24 Beim Rangiere-Programm ist die Laderampe durch einen roten Balken am unteren Bildrand dargestellt. Der Zielpunkt ist durch ein Kreuz markiert.

Zunächst wird mit der Truck-Setzen-Taste der LKW in seine Ausgangsposition gebracht. Standard mäßig erfolgt dies zufallsgesteuert. Über Bearbeiten | Optionen können Sie aber auch andere Startwerte eingeben. Achten Sie bei Ihren Eingaben auf das Vorzeichen; insbesondere bedeutet ein Minuszeichen bei der Geschwindigkeit, dass der LKW rückwärts fährt. Sobald der Wagen nun auf der Straße steht, können Sie Ihn mit der Maus über den Steuerbalken oberhalb der Truck-Setzen-Taste steuern:

- Klicken mit der linken Maustaste setzt den Wagen in Bewegung bzw. hält ihn an.
- Klicken mit der rechten Maustaste führt zu einer Umkehrung der Bewegungsrichtung.
- Durch Bewegen der Maus nach links oder rechts wird die Lenkung des LKWs betätigt.

Im Gegensatz zur Handregelung ist für die Fuzzyregelung keine Richtungsänderung vorgesehen. In der Tat sind Fuzzyregelungen in der Lage, unsere Rangierprobleme in einem Zug zu lösen. Wenn Sie sich dies einmal anschauen wollen, müssen Sie das *Rangiere*-Programm zunächst mit Bearbeiten|Betriebsart umschalten für den Fuzzybetrieb vorbereiten. Sie können jetzt eines der beiden mitgelieferten Projekte, truck1.fuz oder truck2.fuz, laden und bei seiner Arbeit bewundern. (truck1.fuz regelt das Rückwärts-Einparken an eine Bordsteinkante, und truck2.fuz regelt das Rückwärts-Rangieren an eine Rampe.) Es empfiehlt sich, die Geschwindigkeit auf Werte zwischen $-10\,\frac{m}{s}$ und $-5\,\frac{m}{s}$ einzustellen.

Sollten Sie selbst ein Projekt für das Rangierproblem schreiben wollen, müssen Sie als Eingang A den Abstand des Hecks vom LKW zum Straßenrand in m, als Eingang B den Lagewinkel in ° und als Ausgang den Lenk-win kel, ebenfalls in °, benutzen. Nur so kann das Simulationsprogramm das Fuzzyprojekt sinnvoll umsetzen. Achten Sie dabei wieder auf eine korrekte Wahl der Vorzeichen. Die Fuzzyprojekte müssen mit dem *Fuzzy*-Programm hergestellt werden; das *Rangiere*-Programm ist dazu nicht in der Lage. Wenn Sie das *Rangiere*-Programm allerdings vom *Fuzzy*-Programm aus starten, wird automatisch der Dateiname des zuletzt gesicherten Projekts an das *Rangiere*-Programm übergeben; dieses lädt das zugehörige Projekt und schaltet auch automatisch in die Betriebsart Fuzzyregelung um. Für die Projekterstellung wird damit die in Abb. 7.25 wiedergegebene Vorgehensweise empfohlen.



7.25 Optimieren des Rangieren-Projekts

7.7 Aufgaben

1. An dem Projekt pendel.fuz sollen weitere Tests durchgeführt werden. Untersuchen Sie dazu, wie die Fuzzyregelung mit kürzeren Pendellängen und stärkeren Startablenkungen zurechtkommt. Die Vorgabewerte von 2,50 m für die Pendellänge und 10° für die Startablenkung können Sie mit Optionen | Parameter verändern.

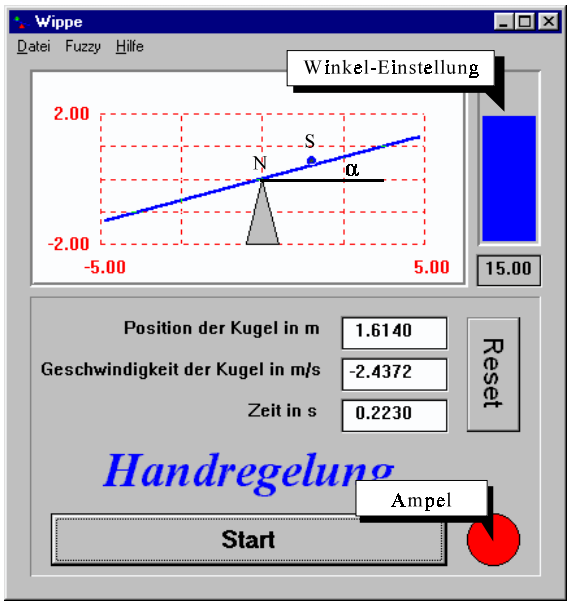
2. Vergleichen Sie die Leistungsfähigkeit der beiden Projekte laufkat1.fuz und laufkat2.fuz bei unterschiedlichen Startpositionen. Diese können Sie mit Optionen | Parameter verändern. Beachten Sie, dass die Grundmenge der Fuzzyvariable "Position" sowie die Zugehörigkeitsfunktion von "fern" entsprechend angepasst werden müssen.

3. Versuchen Sie die Leistungsfähigkeit des Projektes laufkat2.fuz durch Veränderungen an den Zugehörigkeitsfunktionen zu steigern. Benutzen Sie auch die Methode der Verfeinerung.

4. Je nach Luftfeuchtigkeit und Temperatur müssen Pflanzen unterschiedlich begossen werden. Entwickeln Sie ein Fuzzyprojekt mit den Eingangsvariablen Luftfeuchtigkeit und Temperatur sowie der Ausgangsvariable Wassermenge. Definieren Sie für diese Variablen zunächst geeignete Terme mit ihren Zugehörigkeitsfunktionen. Stellen Sie ein geeignetes Regelsystem auf, und testen Sie es mit Hilfe des *Fuzzy*-Programms.

5. In einer Rinne befindet sich frei beweglich eine Kugel; die Rinne ist dabei wie eine Wippe gelagert (Abb. 7.26). Durch Verstellen des Neigungswinkels kann die Kugel balanciert werden. Ziel ist es nun, die Kugel aus ihrer Startposition *S* möglichst rasch in die Nullposition *N* rollen zu lassen und dort zur Ruhe zu bringen.

   a) Starten Sie das Programm *Wippe* mit Simulation | Wippe und sammeln Sie erste Erfahrungen anhand einiger Simulationsläufe. Führen Sie dazu die Maus auf das Winkel-Einstellfeld: Indem Sie die Maus dort auf- und abbewegen, stellen Sie den Neigungswinkel der



7.25 Das *Wippe*-Programm

Wippe ein, und mit der linken Maustaste können Sie die Simulation starten bzw. anhalten. Startwerte für Position und Geschwindigkeit der Kugel können Sie über die entsprechenden Editierfelder vorgeben. Wie lange brauchen Sie nun nach einer gewissen Übungsphase, bis die Kugel zur Ruhe[1] kommt, d.h. bis die Ampel von Rot auf Grün umspringt?

b) Entwickeln Sie mit dem *Fuzzy* -Programm ein Fuzzyprojekt zur Regelung der Wippe. Als erste Eingangsvariable ist die Position der Kugel auf der Wippe (in m) und als zweite Eingangsvariable die Geschwindigkeit (in m/s) zu verwenden. Als Ausgangsvariable dient der Neigungswinkel (in °); die Winkelwerte sollten dabei zwischen -30° und +30° liegen.

c) Testen Sie Ihr Fuzzyprojekt aus. Dazu müssen Sie im *Wippe*-Programm die Betriebsart mit Fuzzy | Betriebsart umschalten auf Fuzzyregelung umstellen und Ihr Projekt laden. Nach wenigen Sekunden sollte Ihre Fuzzyregelung die Kugel an ihr Ziel gebracht haben!

6. Auch die Landung einer Raumkapsel auf dem Mond läßt sich mit *Fuzzy* regeln. Zur Vereinfachung betrachten wir hier allerdings nur vertikale Bewegungen.

a) Der Abbremsvorgang wird über das Raketentriebwerk gesteuert. Je mehr Brennstoff pro s vom Triebwerk als Verbrennungsgas ausgestoßen wird, desto größer ist die Bremswirkung. Beschreiben Sie qualitativ, wie die Bremswirkung (präziser: die Beschleunigung) von der Masse der Raumkapsel, der Abbrandgeschwindigkeit (das ist die Menge an Treibstoff in kg, die in 1 s abgebrannt wird) und der Geschwindigkeit der Verbrennungsgase abhängt.

[1] Ruhe bedeutet hier: Die aktuelle Position weicht höchstens um 3 cm von der Nullposition ab, und die Geschwindigkeit ist kleiner als 3 cm/s.

b) Starten Sie das Programm *Mond* [1] (aus dem Programmmanager oder mit Simulation | mond aus dem *Fuzzy*-Programm). Führen Sie einige manuelle Landemanöver bei verschiedenen Startbedingungen durch. Zur Regelung des Bremsvorgangs dient die Abbrandgeschwindigkeit; sie lässt sich mit der Maus einstellen. Nach welchen Größen richten Sie dabei Ihre Einstellungen?

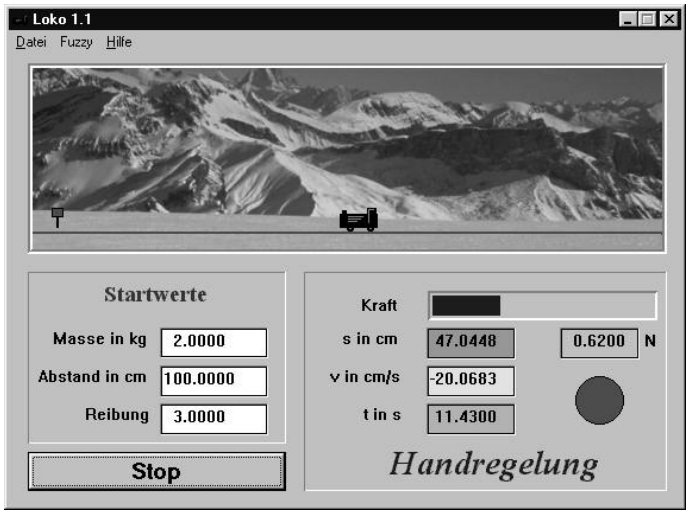
c) Erstellen Sie ein Fuzzyprojekt für die Mondlandung. Da das Mondlandeprogramm keinen eigenen Editor für Fuzzyprojekte besitzt, müssen Sie dazu das *Fuzzy*-Programm benutzen. Eingangsgrößen sind die Höhe der Kapsel über der Mondoberfläche und ihre Sinkgeschwindigkeit, Ausgangsgröße ist die Abbrandgeschwindigkeit. Beachten Sie, dass die Geschwindigkeit beim Sinken negativ ist. Testen Sie Ihr Fuzzyprojekt auch bei verschiedenen realistischen Startbedingungen.

d) Versuchen Sie, Ihr Fuzzyprojekt aus c) auch hinsichtlich des Brennstoffverbrauchs zu optimieren.

7. Bei dem Lokomotiven-Projekt ist das Ziel, eine (Spielzeug-) Lokomotive am Zielpunkt anhalten zu lassen. Da keine Bremse zur Verfügung steht, kommt es darauf an, rechtzeitig und wohldosiert die Antriebskraft der Lokomotive zu drosseln. Im Kapitel 9 wird das zugehörige Realexperiment ausführlich dargestellt.

a) Starten Sie das Programm *Loko* (aus dem Programmmanager oder mit Simulation | loko aus dem *Fuzzy*-Programm). Führen Sie einige manuelle Haltemanöver durch. Benutzen Sie dazu den “Kraftbalken” (Abb. 7.26).

b) Erstellen Sie ein Fuzzyprojekt für die Lokomotive. Beim Anhalten soll die Lokomotive möglichst gleichmäßig ihre Geschwindigkeit reduzieren. Dies können Sie mit dem s-v-Diagramm (Fuzzy | Plot) kontrollieren.



7.26 Das *Loko*-Programm

[1] Eine kurze Bedienungsanleitung zum Mondlandeprogramm finden Sie im Anhang.