# Fuzzy-Logik

***Logik*** *die Lehre vom richtigen Denken des Menschen*

***Fuzzy-Logik*** *richtiges, menschliches Denken der Maschine*

Wilhelm Brezovits

***Vor 5 Jahren war ich 100% sicher, daß Computer nicht intelligent sind, Programme ausführen und Daten bearbeiten. Seit es ernstzunehmende Software (Fuzzy-Logik und Neuro-Module) gibt, die es ermöglicht Wissen zu erfassen, zu bearbeiten und zu optimieren (lernfähig) bin ich 100% sicher, daß meine Vorstellung vor 5 Jahren heute 100% falsch ist (oder in Fuzzy-Logik: der Wahrheitswert µ obiger Aussage = 0% = 0,0).***

Forscher und Wissenschaftler wissen, daß aus Computern mehr herauszuholen ist als das, was wir mit heutiger Soft- und Hardware erreichen. Das Konstruktionsprinzip herkömmlicher Programmiersprachen ist nicht darauf ausgelegt, besonders intelligentes Verhalten zu erzeugen. Heute treffen wir hauptsächlich „programmierbare Rechenmaschinen" und „nachrichtentechnisches Gerät" an.

Hier geht man von der Frage aus, wie man besonders schnell Rechenergebnisse ermitteln kann und diese Ergebnisse rasch zwischen den einzelnen Ein- bzw. Ausgabestationen überträgt.

Wissenschaftlern ist klar, daß COMPUTER MEHR SIND als nachrichtentechnische Geräte und besonders schnelle Rechenmaschinen. Mit dem Begriff ELEKTRONENGEHIRN wird dieses Mehr an Leistung respektvoll umschrieben. Erweitern wir den DATENBEGRIFF zu WISSEN so sprechen wir von einem ÜBERGANG von der DATENVERARBEITUNG zur WISSENSVERARBEITUNG.

***Dazu zählt Fuzzy Logik!***

Fuzzy Logik versucht eine Umsetzung „menschlichen Denkens" = „Expertenwissen" in Algorithmen. Expertenwissen ist das Wissen einer oder mehrere Personen die etwas wissen!

Dieses Wissen wird per Fuzzy Logik in Form normaler Sprache erfaßt und das Fuzzy Werkzeug generiert daraus C- Code. Dieser braucht dann nur noch compiliert werden.

*Damit wir alle „Experten" sind habe ich im nachfolgendem Beispiel das Lenken eines Autos gewählt.*

Expertenwissen dabei ist z. Bsp.: WENN vorne kein Platz ist UND links kein Platz ist UND rechts viel Platz ist DANN lenken wir nach rechts !

Es muß aber nicht immer Messen, Steuern und Regeln sein! Ein Beispiel der Anwendung von Fuzzy-Logik in der Medizin zur Diagnostik möchte ich anführen:

Wir fragen einen oder mehrere Ärzte:

> *Was hat der Patient wenn der Blutdruck den Wert x hat, das Labor den Blutbefund y vorweist und der Patient über z klagt?*

Die Ärzte antworten:

> *Der Patient hat zu 95 % die Krankheit a,*
>
> *der Patient hat zu 80 % die Krankheit b,*
>
> *der Patient hat zu 30 % die Krankheit c,*
>
> *die Krankheit d kann ich zu 100 % ausschließen,*
>
> *die Krankheit e kann ich zu 90 % ausschließen.*

Wir bekommen als Antwort Expertenwissen, welches per Fuzzy-Logik erfaßt wird.

Bei Fuzzy-Logik gibt es auch Neuronale Module (lernfähig)!

Man teilt dem System mit, daß die erhaltene Antwort richtig, falsch oder zu 70% richtig , u.s.w, ist. Das Neuro Modul im Fuzzy-Logik Werkzeug beurteilt daraufhin das Expertenwissen und wichtet es.

Folgendes kann dabei entstehen:

Die Aussage:

> *WENN der Patient den Blutdruckwert x hat UND das Labor den Blutbefund y vorweist UND der Patient über z klagt DANN hat der Patient zu 95 % die Krankheit a*

ist nur zu 73 % richtig!

Die Aussage:

> *WENN der Patient den Blutdruckwert x hat UND das Labor den Blutbefund y vorweist UND der Patient über z klagt DANN hat der Patient zu 80 % die Krankheit b*

ist zu 99 % richtig!

u.s.w. ...

Weitere bereits realisierte Einsatzfälle von Fuzzy Logik:

Waschmaschine
Fuzzy Logik regelt die Drehzahl und achtet darauf, daß die Wäsche in der Trommel rollt und nicht fällt um den größten Wascheffekt (Reibung) zu erreichen!

Waschmaschine
Fuzzy-Logik prüft die Reinheit des Wassers vor und nach dem Waschvorgang und ermittelt so die notwendige Dauer des nächsten Waschvorganges und die Dosierung des Waschmittels.

Automatikgetriebesteuerung
Fuzzy-Logik erkennt ob der Fahrer einen sportlichen, aggressiven oder gleitenden Fahrstil hat, ob das Fahrzeug sich in einer Kurve befindet, bergaufwärts, bergabwärts oder in der Ebene unterwegs ist, und demnach entscheidet Fuzzy-Logik ob in einen anderen Gang geschaltet wird oder nicht.

Camcorder
Fuzzy-Logik hat einen größeren Bildausschnitt als der Benutzer des Camcorders zur Verfügung. Verwackelt der Benutzer so kann Fuzzy-Logik das ausgleichen, indem es beim Verwackeln den kleinen Bildausschnitt des Benutzers in den größeren Bildausschnitt von Fuzzy verschiebt.

## EINSATZGEBIETE VON Fuzzy-Logik

***Überall dort, wo***

⇒ kein exaktes (mathematisches) Modell der Regelungs-/Steuerungsaufgabe vorliegt

⇒ das System sich streng nichtlinear verhält

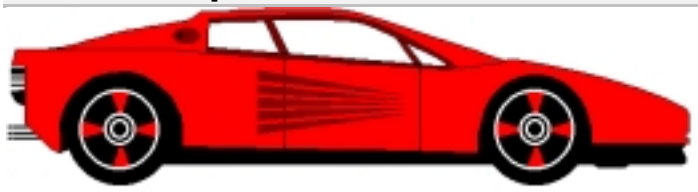
⇒ Expertenwissen (intuitives) vorliegt, aber nicht oder nur schwer in klassische Algorithmen umsetzbar ist.

***Und wo sollte Fuzzy-Logik nicht unbedingt verwendet werden?***

⇒ wenn das Problem einfach mit einem konventionellem PID-Algorithmus gelöst werden kann.

⇒ wenn ein einfaches, klar definiertes und schnell zu lösendes mathematisches Model existiert.

## Unser Beispiel



Mit Hilfe von Fuzzy-Logik wird das Lenkrad eines mit konstanter Geschwindigkeit fahrenden Autos gesteuert. Die Aufgabe des Fuzzy-Regelalgorithmus ist es, das Auto so zu steuern, daß es an Hindernissen (vorne, links und rechts) nicht anstößt!

### 1. Schritt

Zur Beschreibung des Abstandes vom Hindernis und zur Beschreibung der Ausweichrichtung verwenden wir „linguistische Variablen“ - das sind Variablen die als Werte Ausdrücke unserer normalen Umgangssprache aufnehmen (=„Terme“):

| Eingang/ Ausgang | linguistische Variable | Term_1 | Term_2 | Term_3 |
|---|---|---|---|---|
| Eingang | abstand_vorne | klein | groß | |
| Eingang | abstand_links | klein | groß | |
| Eingang | abstand_rechts | klein | groß | |
| Ausgang | lenkung | links | gerade | rechts |

**Anmerkung**: um ein besseres „Fahrverhalten“ unseres Fahrzeuges zu bekommen, ist es erforderlich mehr als hier angegebene Terme zu verwenden, dies würde aber den Rahmen unseres Beispiels sprengen! Die linguistische Variable abstand_vorne könnte also folgende Werte (Terme) bekommen: sehr_sehr_klein, sehr_klein, klein, mittel, groß, sehr_groß, sehr_sehr_groß, u.s.w..

### 2. Schritt

Obige linguistische Variablen und Terme werden graphisch dargestellt:

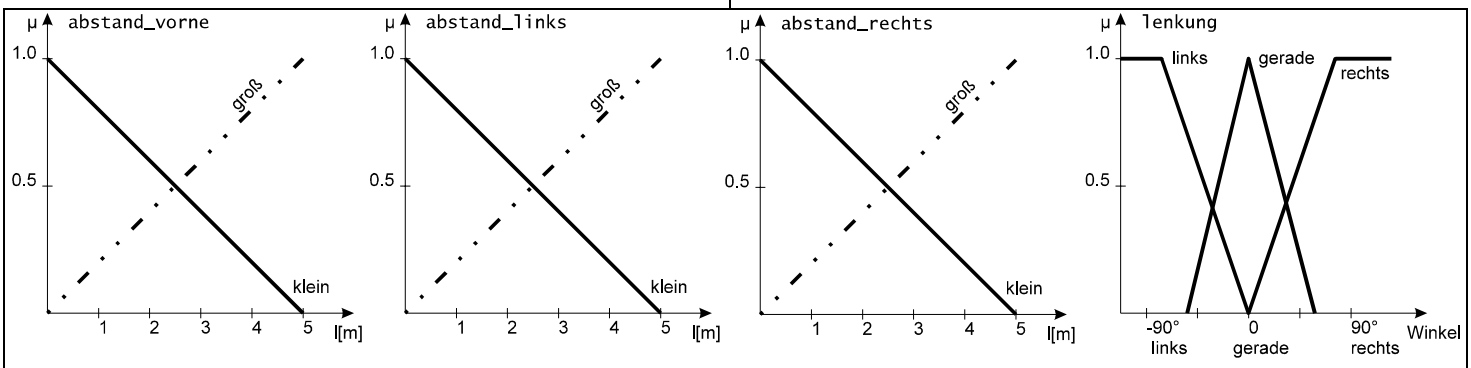



**Bild 1:** Eingangsgrößen: **abstand_vorne, abstand_links, abstand_rechts**; Ausgangsgröße: **lenkung**

### 3. Schritt

Eingabe des Expertenwissens (der Regeln, rules) in Form unserer normalen Sprache:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **WENN** | abstand_vorne = groß | **UND** | abstand_links=klein | **UND** | abstand_rechts=groß | **DANN** | lenkung=rechts |
| **WENN** | abstand_vorne = groß | **UND** | abstand_links=groß | **UND** | abstand_rechts=klein | **DANN** | lenkung=links |
| **WENN** | ... | **UND** | ... | **UND** | ... | **DANN** | ... |
| **...** | ... | **...** | ... | **...** | ... | **...** | ... |
| **...** | ... | **...** | ... | **..** | ... | **..** | ... |

Wir sehen hat das Fahrzeug *rechts* Platz, so lenken wir nach *rechts*;
hat das Fahrzeug *links* Platz, so lenken wir nach *links*;
u.s.w.;

**Anmerkung**: um ein besseres „Fahrverhalten“ unseres Fahrzeuges zu bekommen, ist es erforderlich mehr als hier angegebene Regeln zu verwenden, dies würde aber den Rahmen unseres Beispiels sprengen!

### 4. Schritt

Wir sind mit der Arbeit mit unserem Fuzzy-Logik Werkzeug fertig!

Jetzt ist nur noch ein Mausklick notwendig und das Fuzzy-Tool erzeugt aus unserer obigen Eingabe eine ANSI-C-Funktion!

Die ANSI C - Funktion sieht etwa so aus:

```
float fuzzy_lenkung (float abstand_vorne,
                     float abstand_links,
                     float abstand_rechts)
```

### 5. Schritt

Schreiben wir ein C-Programm, so können wir die erzeugte ANSI-C-Funktion folgendermaßen einbinden:

```
/* Prototypen */
extern float fuzzy_lenkung (float abstand_vorne,
                            float abstand_links,
                            float abstand_rechts);
/*=========================================================*/
void main (void)
{
  float abstand_vorne = 0,
        abstand_links = 0,
        abstand_rechts = 0,
        lenkradstellung = 0;
  while (1)
  {
  // Programmteil zum Einlesen der Abstandssensoren vom Auto
          abstand_vorne  = _____________________________;
          abstand_links   = _____________________________;
          abstand_rechts = _____________________________;
  // Aufruf des Fuzzy-Reglers !!!
          lenkradstellung = fuzzy_lenkung(abstand_vorne,
                                          abstand_links,
                                          abstand_rechts);
  // Mit Hilfe der Variable lenkradstellung
  // (deren Wert die Fuzzy-Funktion liefert)
  // wird in diesem Programmabschnitt
  // ein Schrittmotor angesteuert,
  // der das Lenkrad unseres Autos betätigt.
.....................................;
.....................................;
.....................................;
.....................................;
  } /* ende while (1) */
} /* ende main */
/*=========================================================*/
```

## Jetzt wollen wir unser Auto (und unser Programm) starten!

Nehmen wir an, unser Auto befindet sich in folgender Umgebung:

- Der Abstand zum Hindernis nach vorne ist 5 Meter (`abstand_vorne=5`).
- Der Abstand zum Hindernis nach links ist 1 Meter (`abstand_links=1`).
- Der Abstand zum Hindernis nach rechts ist 4 Meter (`abstand_rechts=4`).

Unser C-Programm hat dann folgende Werte:

```
/*=========================================================*/
void main (void)
{
  float abstand_vorne = 0,
        abstand_links = 0,
        abstand_rechts = 0,
        lenkradstellung = 0;
  while (1)
  {
   // Programmteil zum Einlesen der Abstandssensoren vom Auto
          abstand_vorne  =    5;
          abstand_links   =   1;
          abstand_rechts =    4;
// Aufruf des Fuzzy-Reglers !!!
          lenkradstellung = fuzzy_lenkung (abstand_vorne,  //5
                                           abstand_links,  //1
                                           abstand_rechts);//4
...
```

Der Fuzzy Regler wird also mit den aktuellen Werten 5, 1 und 4 aufgerufen, was jetzt passiert (und was die erzeugte Fuzzy-Software jetzt macht) sehen wir uns am besten in ***Bild 2*** an:

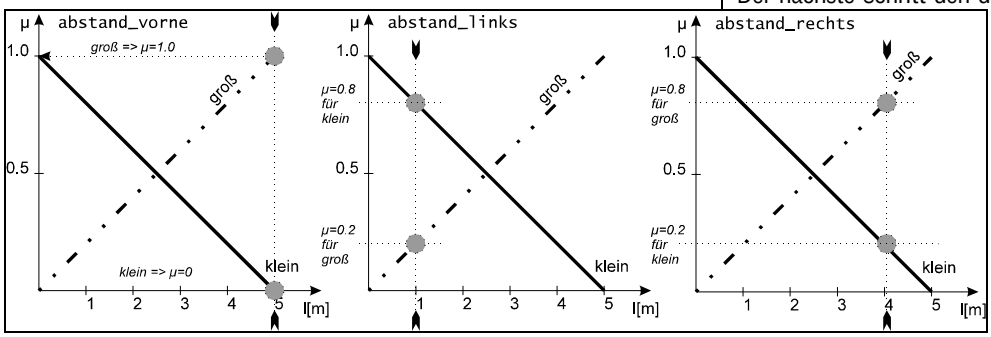



**Bild 2: Eingangsgrößen werden FUZZYFIZIERT**

Gehen wir mit den scharfen Werten `(5,1,4)` ins Schaubild so wird dort FUZZYFIZIERT, d.h. es wird unscharf gemacht.

Der `abstand_vorne` ist nicht mehr `5`, sondern:

Der Term `klein` von `abstand_vorne` ist zu 0 % richtig (... der Wahrheitswert μ = 0,0).

Der Term `groß` von `abstand_vorne` ist zu 100 % richtig (... der Wahrheitswert μ = 1,0).

Der `abstand_links` ist nicht mehr (scharf) 1, sondern unscharf für den Term klein μ = 0,8 und für den Term groß μ = 0,2.

Der `abstand _rechts` ist nicht mehr (scharf) 4, sondern unscharf für klein μ = 0,2 und für groß μ = 0,8.

Wir haben jetzt gesehen wie aus scharfen Werten unscharfe Werte werden (=FUZZYFIZIERUNG).

Der nächste Schritt den das erzeugte Programm beschreitet ist der, mit den unscharfen Werten in die WENN-DANN-REGELN = INFERENZ zu gehen.

INFERENZ = Abarbeitung der Regeln, der Wissensbasis = Verarbeitung von WISSEN !!! (keine Datenverarbeitung)

Die UND-Verknüpfung bedeutet hier die Bildung des Minimalwertes.

**Anmerkung:** es gibt noch viele andere Verknüpfungsmethoden, diese sind hauptsächlich Gegenstand von Diplom und Doktorarbeiten, aber auch von diversen theoretischen Betrachtungen diverser Fuzzy-Bücher (die man sehr schwer versteht!).

In der Praxis hat es sich allerdings gezeigt, daß es vollkommen egal ist, welche der tollen Verknüpfungsmethoden man verwendet, stimmen die linguistischen Variablen, die Terme, die Schaubilder und die Regeln so hat man hier genug Spielraum, sein System zu optimieren!

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **WENN** | `abstand_vorne = groß` μ=1,0 | **UND** | `abstand_links=klein` μ=0,8 | **UND** | `abstand_rechts=groß` μ=0,8 | **DANN** | `lenkung=rechts` μ=0,8 |
| **WENN** | `abstand_vorne=groß` μ=1,0 | **UND** | `abstand_links=groß` μ=0,2 | **UND** | `abstand_rechts=klein` μ=0,2 | **DANN** | `lenkung=links` μ=0,2 |
| **WENN** | μ=... | **UND** | μ=... | **UND** | μ=... | **DANN** | μ=... |
| **...** | ... | **...** | ... | **..** | ... | **..** | ... |

Mit den ermittelten Ausgangswerten (rechts:µ=0,8; links:µ=02; ...) für `lenkung` gehen wir jetzt ins Ausgangsdiagramm:

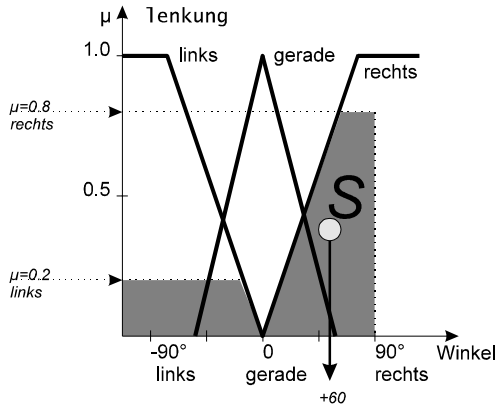



Bild 3: Ausgangsgröße

Dann bestimmen wir (bzw. das erzeugte Programm vom Fuzzy-Tool) den Flächenschwerpunkt im Ausgangsschaubild `lenkung`, projizieren diesen auf die x-Achse und schon haben wir einen SCHARFEN Wert den die ANSI-C funktion `fuzzy_lenkung` zurückliefert ( +60° rechts lenken)!!!

Diesen Schritt von „unscharf scharf machen“ nennt man auch DEFUZZYFIZIERUNG (Schwerpunkt bestimmen)!

**Anmerkung:**
es gibt noch viele andere DEFUZZYFIZIERUNGSMETHODEN, diese sind hauptsächlich Gegenstand von Diplom und Doktorarbeiten, aber auch von diversen theoretischen Betrachtungen diverser Fuzzy-Bücher (die man sehr schwer versteht!).

In der Praxis hat es sich allerdings gezeigt, daß es vollkommen egal ist, welche der tollen DEFUZZYFIZIERUNGSMETHODEN man verwendet, stimmen die linguistischen Variablen, die Terme, die Schaubilder und die Regeln so hat man hier genug Spielraum sein System zu optimieren!

Dieser Wert (+60°) wird jetzt in unserem C-Programm weiterverarbeitet:

```
// Aufruf des Fuzzy-Reglers !!!
       lenkradstellung = fuzzy_lenkung (abstand_vorne,
                                        abstand_links,
                                        abstand_rechts);
                ⇓
               60
// Mit Hilfe der Variable lenkradstellung (+60°)
// wird in diesem Programmabschnitt
// ein Schrittmotor angesteuert,
// der das Lenkrad unseres Autos betätigt.
.....................................;
```

Wir sehen: obwohl wir nur wenige linguistische Variablen verwenden, obwohl wir nur wenige Terme verwenden, obwohl wir nur lineare Funktionen in den Schaubildern verwenden, obwohl wir nur zwei Regeln angegeben haben lenkt das Fahrzeug in die RICHTIGE RICHTUNG (dort wo Platz ist!).

***DAS IST Fuzzy-Logik!***

Abschließend möchte ich den Vorgang bei der Abarbeitung der Fuzzy-Funktion darstellen:

| FUZZYFIZIERUNG | INFERENZ | DEFUZZYFIZIERUNG |
|---|---|---|
| „scharfe“ Werte werden per Schaubilder (linguistische Variablen, Terme) in Wahrheitswerte umgewandelt ... | die WENN-DANN-Regeln (die Wissensbasis) werden abgearbeitet | aufgrund der Regeln werden im Ausgangsdiagramm Flächen wirksam, der Schwerpunkt wird bestimmt und schon können wir auf der Abszisse einen „scharfen“ Wert ablesen... |

Hinweise zum Test im Zielsystem

Bei den heute am Markt befindlichen Fuzzy-Werkzeugen besteht die Möglichkeit sogenannte ON-LINE-MODULE mitzubinden. Diese Module sorgen dann im Zielsystem dafür, daß über eine serielle Schnittstelle im Zielsystem DEBUG-Information an die Fuzzy-Oberfläche gesendet wird. Mit Hilfe dieser Einrichtung kann in Echtzeit das Verhalten des Systems in den obigen Schaubildern und Regeln beobachtet werden.

Hinweise zum Optimieren

Bei den heute am Markt befindlichen Fuzzy-Werkzeugen besteht die Möglichkeit sogenannte NEURO-MODULE zu verwenden, NEURO-Module beobachten den Ist und den Soll-Zustand der Ausgangsgröße und verändern den Wahrheitswert der Regeln indem sie den Regeln einen Wahrheitswert zwischen 0 und 100 % zuteilen.

Hinweise zu bisherigen Implementierungen von Fuzzy-Logik

**=> Softwareimplementierung:** wenn es portierbar sein soll!

Man verwendet ganz einfach ein Fuzzy-Tool das C-Code generiert. Die erzeugte C-Funktionen ruft man dann im eigenem Programm auf. Der Code braucht dann nur für das entsprechende Betriebssystem (wenn vorhanden) und für den entsprechenden Mikroprozessor oder Mikrocontroller compiliert werden.

Braucht man viel Performance vom System, empfiehlt sich der Einsatz eines schnellen Mikrocontrollers, wie sie zum Beispiel die 16 bit 80C166 Familie von Siemens bietet (100ns Abarbeitungsgeschwindigkeit für leistungsfähige, komfortable Befehle, Interruptantwortzeit ab 250 ns, Signale erkennen oder erzeugen im 50 ns Raster).

**=> Mikrocodeänderung:** wenn es langsam sein darf!

Diverse Halbleiterhersteller haben ihre 8 bit Mikrocontroller vergewaltigt indem der Mikrocode (nur bei CISC-Architekturen möglich) geändert wurde.

Die Befehle sind für Fuzzy-Aufgaben ausgerichtet, der Nachteil: der Baustein verliert die Universalität seines Befehlsvorrates und die Rechengeschwindigkeit entspricht nach wie vor einer langsamen 8 bit CISC basierenden Mikroprozessorarchitektur.

**=> Hardwareimplementierung:** wenn es schnell sein muß!

Wie Coprozessoren für floating-point Arithmetik gibt es auch Coprozessoren für Fuzzy-Logik.

Ein Beispiel dafür wäre der SAE 81C99 von Siemens.

Bei solchen Bausteinen kristallisiert sich die Leistungsfähigkeit bei einer großen Menge von Eingängen, Ausgängen und Regeln heraus.

In der folgenden Tabelle möchte ich die Leistungsfähigkeit von diversen Siemens Bausteinen welche Fuzzy-Algorithmen abarbeiten vergleichen:

BENCHMARK (Fuzzy-System mit 20 Regeln, 2 Eingängen und ein Ausgang)

| Baustein | Takt-frequenz | Zeit | Auflösung | ROM/ RAM |
|---|---|---|---|---|
| | Mhz | µs | bit | Byte |
| SAB 80C32 = C501 (8-bit-Mikrocontroller) | 12 | 1400 | 8 | 540/25 |
| SAB 80C537 (8-bit-Mikrocontroller mit MDU) | 16 | 900 | 8 | 450/25 |
| SAB C501-40 (8-bit-Mikrocontroller) | 40 | 420 | 8 | 540/25 |
| SAB 80C166 (16-bit-Mikrocontroller) | 20 | 60 | 16 | 560/30 |
| SAB C167CR-16F (16-bit-Mikrocontroller mit 128 KByte Flash, 4 KByte RAM und CAN on Chip) | ** | ** | ** | ** |
| SAE 81C99 (Fuzzy-Logik-Coprozessor) | 20 | 30* | 8 | -/- |

* ... Bei solchen Bausteinen erkennt man die Leistungsfähigkeit erst bei einer großen Anzahl von Eingängen, Ausgängen und Regeln!

**... demnächst verfügbar! □