Diplomarbeit
als Voraussetzung zur Führung der Bezeichnung
Diplom-HTL-Ingenieur (Dipl.-HTL-Ing.)

Autonome Roboternavigation mittels Methoden der künstlichen Intelligenz

(Autonomous Robot Navigation Based on Artificial Intelligence)

von

Ing. Roland Stelzer, BSc
Ospelgasse 1-9/2/5
A-1200 Wien
<stelzerroland@yahoo.com>

Wien, März 2004 _____

Kurzfassung / Abstract

Deutsch

Dieses Projekt verbindet unterschiedliche Methoden und Technologien zu einem autonomen Roboter, der in der Lage ist, einer Linie zu folgen und auf am Weg befindliche Verkehrszeichen situationsgerecht zu reagieren. Neben Hardwaresteuerung und Bildverarbeitung bilden die Navigation entlang einer Linie mittels Fuzzy Logic und die Interpretation der Verkehrszeichen unter Verwendung einer Self-Organizing-Feature-Map (Kohonen-Netzwerk) die wesentlichen Bestandteile der Arbeit.

English

This project combines the areas fuzzy logic, neural nets, image processing and mobile robotics. The only sensor of the system is a camera. A fuzzy logic system navigates the robot along a line. Traffic signs along the road are detected, transformed and identified using a self organising feature map (Kohonen-network).

Danksagung

Ich bedanke mich recht herzlich bei Dr. Herbert Hörtlehner, der während meines Studium an der University of Derby durch seinen unkonventionellen Unterricht mein Interesse an Robotik und den Methoden der künstlichen Intelligenz geweckt hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Vision eines völlig selbständig navigierenden Autos ist wohl so alt wie das Auto selbst. Viele technische Hilfsmittel, angefangen vom Rückspiegel über automatische Gangschaltung bis zu elektronischen Einparkhilfen, erleichtern zwar das fahren, trotzdem ist es nach wie vor der Mensch, der das Fahrzeug steuert und kontrolliert. Erst in den letzten Jahren kommt man dem autonomen Vehikel durch Verwendung neuester Technologien der Künstlichen Intelligenz (KI) gepaart mit der Rechenleistung moderner Computer schrittweise näher.

Ziel dieser Arbeit ist es, ein völlig autonomes Vehikel zu entwickeln, das in der Lage ist, auf einem ihm unbekannten Parkour situationsgerecht zu navigieren. Unter situationsgerecht wird hier verstanden, dass zum einen ein durch eine farbige Linie gekennzeichneter Kurs verfolgt wird und während dessen Verkehrszeichen befolgt werden, die an der Linie positioniert sind. Bei den Verkehrszeichen handelt es sich hier um Geschwindigkeitsbeschränkungen, da auf diese einfach und wirklichkeitsnah reagiert werden kann. Der Roboter verändert also während der Fahrt entlang der Linie seine Geschwindigkeit in Abhängigkeit von den Verkehrszeichen.

Die Idee zu diesem Projekt entstand bereits während meines Studiums an der University of Derby. Dort hatte ich die Möglichkeit, mich eingehend mit Fuzzy Logic (FL) zu beschäftigen. Besonders spannend erschien mir immer die praktische Anwendung dieser Methoden. Also entwickelte ich im

Rahmen meiner Diplomarbeit and der University of Derby eine web-basierte Fuzzy-Shell, über die seit Anfang 2003 unter anderem die Heizung in einem Projektraum in der HTL Spengergasse in Wien gesteuert wird. Eine Fuzzy-Shell ist ein Tool, mit dem FL-Systeme sehr einfach, meist grafisch, designed werden können. Das Heizverhalten kann via Web konfiguriert und mitverfolgt werden. Während diese Heizung mit Innen- und Außentemperaturfühler zwar eine hübsche Anwendung ist, sich mit Fuzzy-Steuerungen auseinander zu setzen, sollte die nächste Herausforderung ein System sein, bei dem sich „richtig was rührt“, also ein etwas schnelleres Echtzeitsystem. So entstand die Idee des autonomen Fahrzeuges.

Erstes Ziel war es nur, ein Fahrzeug mittels FL-Steuerung eine Linie entlang zu schicken. Ich zog dafür verschiedene Möglichkeiten in Betracht. Eine war, eine dunkle Linie von hellem Untergrund über die Reflektion eines Lichtstrahls zu unterscheiden. Doch wesentlich spannender fand ich, in das Projekt auch Bilderkennungsmechanismen zu integrieren. Für diesen Zweck wurde an der Front des Roboters eine Kamera montiert. Aus der Überlegung heraus, welche Erweiterungsmöglichkeiten sich durch die Kamera ergeben, entstand die Idee zum zweiten großen Teil dieses Projekts, nämlich die Identifikation von Verkehrszeichen.

Abbildung 1.1 zeigt den verwendeten Versuchsaufbau. Die „Straße“ wird durch eine grüne Linie auf weißem Untergrund dargestellt. Die Verkehrszeichen (in diesem Fall ein Stopp-Schild) werden direkt auf die Linie positioniert.

Der Schwerpunkt dieses Projekts liegt in der Entwicklung der Steuerungssoftware unter Verwendung von KI-Methoden und nicht im Bereich von Mechanik oder Elektronik, obwohl mich dieses Projekt auch in diesen Gebieten neue Erfahrungen sammeln ließ. Um möglichst rasch zur Implementierung der KI-Methoden am Roboter zu kommen, entschied ich mich, das Chassis inklusive Getriebe und Motoren aus einem Bausatz des Magazins *Real Robots*®[20] zu verwenden.

Schwierigkeiten während der Implementierung machte die Bilderkennung unter wechselnden Lichtverhältnissen. Die Umwandlung des Bildes vom RGB- in das HSV-Farbmodell (Farbmodelle werden später detaillierter beschrieben) brachte schließlich die Lösung des Problems und machte die weitere

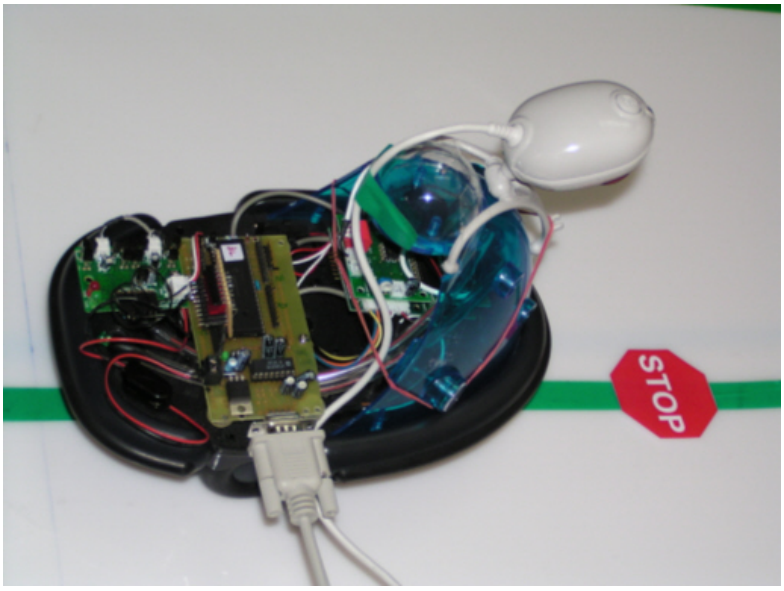


Abbildung 1.1: Roboter mit Kamera und Mikrocontroller

Bildverarbeitung weitgehend beleuchtungsunabhängig. Ein anderes Problem war die schlechte Qualität der Elektromotoren. Obwohl die Steuerung in der Simulation funktionierte, gab es in der Praxis anfänglich Schwierigkeiten, da die Motoren stark verzögert anliefen und nicht sofort stoppten, wenn das entsprechende Signal gesendet wurde. Dieses Problem wurde durch entsprechendes Tuning des Fuzzy-Systems behoben.

Im folgenden Kapitel wird der theoretische Hintergrund der in diesem Projekt verwendeten Methoden beleuchtet. Im wesentlichen sind das Fuzzy-Logic und eine spezielle Form der künstlichen neuronalen Netze, nämlich die Self-Organizing-Feature-Map nach Tuevo Kohonen. Die Theorie zur Fuzzy Logic wurde in ähnlicher Form auch in meiner Diplomarbeit an der University of Derby mit dem Titel „*FuzzyWeb - A Web-Based Approach to Fuzzy System Design*“ [22] publiziert.

Das Kapitel „Implementierung“ geht auf die einzelnen Hardwarekomponenten des Systems und deren Kommunikation untereinander ein. Für die Kommunikation habe ich ein eigenes Protokoll entwickelt, um möglichst einfach und effizient Daten zwischen den beteiligten Komponenten auszutauschen.

Kapitel 4 geht im Detail auf die Algorithmen ein, die für die Identifikation einer Linie aus einem Bild und die Navigation entlang derselben verwendet wurden. Dieser Teil gliedert sich grob in die Bildanalyse, die als Ergebnis die Lagekoordinaten der Linie liefert, und ein Fuzzy Logic System, das die entsprechenden Schlüsse bezüglich der Navigation zieht.

In Kapitel 5 der genau beschrieben, wie einerseits erkannt wird, dass sich ein Verkehrszeichen im Blickfeld des Roboters befindet, und andererseits dieses Verkehrszeichen dann identifiziert und befolgt wird. Hier wird zuerst das Bild analysiert und dann so transformiert, dass es von einem neuronalen Netz weiterverarbeitet werden kann. Das hier verwendete neuronale Netzwerk ist - nachdem es ausreichend trainiert wurde - in der Lage festzustellen, um welches Verkehrszeichen es sich im Einzelfall handelt.

Den Abschluss bilden ein paar Überlegungen zu möglichen Einsatzgebieten und Erweiterungen dieses Projekts.

Kapitel 2

Methoden der künstlichen Intelligenz

Für die Steuerung des Roboters werden zwei wesentliche Methoden der künstlichen Intelligenz (KI) eingesetzt. Zum einen Fuzzy Logic (FL) und zum anderen eine spezielle Form der künstlichen neuronalen Netze, die Self-Organising-Feature-Map (SOFM). Die beiden Methoden werden in diesem Kapitel eingehend behandelt.

2.1 Fuzzy Logic

Lotfi Zadeh, Professor an der University of California in Berkeley, entwickelte Mitte der 1960er-Jahre das Konzept der Fuzzy Logic (FL). Grundsätzlich ist die FL eine Erweiterung der klassischen Logik, die auch Wahrheitswerte zwischen den Boolschen Werten *wahr* und *falsch* erlaubt. [13]

FL ermöglicht uns, Systeme mittels in natürlicher Sprache formulierter Regeln zu beschreiben. Das FL System konvertiert diese Regeln in deren mathematische Interpretation. Auf diese Weise können beispielsweise Steuerungsmechanismen auf Basis von Expertenwissen und Erfahrungen erstellt werden, ohne die genauen technischen Hintergründe und Zusammenhänge zu kennen. In vielen Fällen wird dadurch die Systementwicklung erheblich vereinfacht und beschleunigt.

Zu Beginn hielt sich die Anerkennung dieser Theorie sehr stark in Grenzen. Erst als 1974 Professor Mamdani mit seinem Studenten Assilian eine Dampfmaschine mittels FL steuerte, erweckte sie große Aufmerksamkeit. Es handelte sich um die erste praktische Anwendung der FL. Ein weiterer Schritt, der FL zum Durchbruch verhalf war die Entwicklung des ersten FL-Chips im Jahr 1975 durch die Bell Laboratories. Der Chip wurde in der Folge in einer großen Anzahl von Anwendungen, wie etwa Kameras oder Reis-Kochern verwendet. Heute ist FL einer der am schnellsten wachsenden Bereiche der künstlichen Intelligenz. [16]

2.1.1 Fuzzy-Sets und Membership-Funktionen

Das klassische logische Denken basiert auf einer zweiwertigen Aussagenlogik, auch *Boolsche Logik* genannt. Sie geht davon aus, dass jede Aussage entweder exakt wahr oder exakt falsch ist. Auf die Mengenlehre umgelegt heißt das, dass ein Element einer Menge entweder zugehörig ist, oder nicht; eine und genau eine der beiden Aussagen ist wahr. Mengen mit diesen strengen Grenzen im Sinne der klassischen Mengenlehre werden in der FL als *Crisp-Sets* bezeichnet.

Die FL basiert auf unscharfen Mengen, so genannten Fuzzy-Sets. Die Fuzzy-Set-Theorie ist über weite Strecken eine Generalisierung der klassischen Mengenlehre. [5] Eine klassische Menge A ist entweder durch ihre Elemente definiert, z.B. in der Form

$$A = \{a, b, c, ...\}$$

oder als Teil einer Obermenge X, für den eine bestimmte Bedingung zutrifft, z.B. in der Form

$$A = \{x \in X \mid x \text{ erfüllt die Bedingung } cond\}$$

Die Aussage $x \in X gehört\ zur\ Teilmenge A$ kann mathematisch äquivalent

über die sogenannte *charakteristische Funktion* μ

$$\mu_A : X \rightarrow \{0, 1\}$$

beschrieben werden.

In der Fuzzy Logic werden - anstatt jedem Element der Obermenge X nur entweder 0 oder 1 zuzuordnen - den Elementen aus X verlaufende Werte zwischen 0 und 1 zugeordnet. Der jeweilige Wert beschreibt den *Grad der Zugehörigkeit (Membership Value)* eines jeden Elements aus X zur jeweiligen Teilmenge A. So eine Funktion

$$\mu_A : X \rightarrow [0, 1]$$

wird als Zugehörigkeitesfunktion (Membership Function) bezeichnet.

Die Boolsche Aussagenlogik birgt große Einschränkungen im Hinblick auf die Darstellung von unscharfen Konzepten in sich. Man kann beispielsweise versuchen die Raumtemperatur im Sinne der Boolschen Logik mit den Begriffen *warm* und *kalt* zu definieren. Es wird keinen Zweifel geben, dass 35°C warm, und nicht kalt ist. Ebenso wird man sich darauf verständigen, dass 5°C kalt, und folglich nicht warm ist. Für eine Raumtemperatur von 18°C allerdings wird es schon wesentlich schwieriger, sie mit den Begriffen kalt und warm zu klassifizieren. Die Boolsche Logik bietet keine Möglichkeit, Werte zwischen wahr und falsch (in den Abbildungen auf der Ordinate durch die Werte 1 und 0 dargestellt) zu identifizieren.

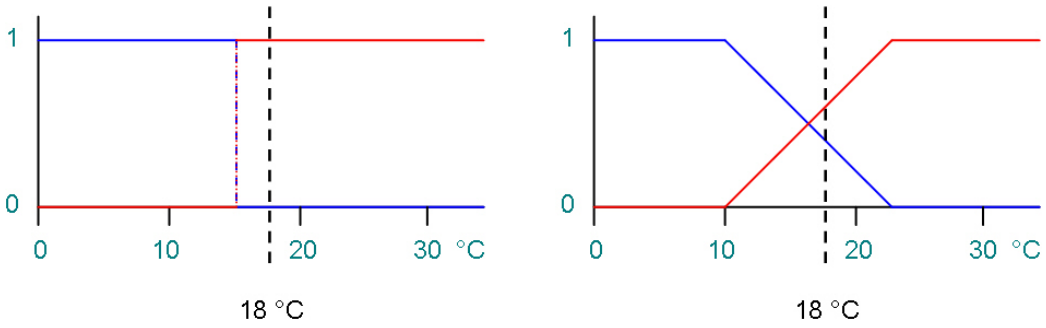


Abbildung 2.1: Boolean versus Fuzzy

Die Fuzzy Logic hingegen erlaubt jedem Element einer Grundmenge einen gewissen Grad an Zugehörigkeit zu einer anderen Menge. Dieser Grad der Zugehörigkeit, auch Membership-Value genannt, ist über eine Membership-Funktion definiert. Der Membership-Value liegt im Intervall $[0, 1]$.

Der blaue Graph in Abbildung 2.1 repräsentiert die Membership-Funktion für kalt, der rote steht für warm. Der Wert 18°C gehört in der Boolschen Logik (linkes Bild in Abbildung 2.1) gar nicht zur Menge kalt, und ganz zur Menge warm. In der FL (rechtes Bild) sieht man, dass 18°C Teil von beiden Mengen (Fuzzy-Sets) ist, und zwar jeweils zu einem gewissen Grad an Zugehörigkeit. Dieser Grad an Zugehörigkeit (Membership-Value) ist über die Membership-Funktion definiert. In der Skizze sehen wir, dass der Wert 18°C zu etwa 0,4 dem Fuzzy-Set kalt und zu etwa 0,6 dem Fuzzy-Set warm zugehörig ist.

2.1.2 Logische Operatoren auf Fuzzy-Sets

Aussagen können mittels der logischen Operatoren *UND* und *ODER* kombiniert werden. Für die klassische zweiwertige Logik gilt die in Tabelle 2.1 dargestellte Wahrheitstabelle.

A	**B**	**A und B**	**A oder B**	**nicht A**
Wahr	Wahr	Wahr	Wahr	Falsch
Wahr	Falsch	Falsch	Wahr	Falsch
Falsch	Wahr	Falsch	Wahr	Wahr
Falsch	Falsch	Falsch	Falsch	Wahr

Tabelle 2.1: Boolsche Wahrheitstabelle

Entsprechend dieser Boolschen Wahrheitstabelle wurde für das Fuzzy-UND das Konzept der *t-Normen (triangular Norms)* entwickelt. Das Fuzzy-ODER wird entsprechend durch *t-Conormen* (auch *s-Normen* genannt) abgebildet. Die am häufigsten verwendete t-Norm ist der *Minimum-Operator*, die dazugehörige s-Norm der *Maximum-Operator*. Tabelle 2.2 zeigt weitere gängige Kombinationen.

Die Anwendung der logischen Operatoren *UND* und *ODER* auf Fuzzy-Sets ist in Abbildung 2.2 grafisch dargestellt. Die für das *UND* verwendete

s-Norm	s-Norm
Minimum: $t_{min}(a,b) = \min(a,b)$	Maximum: $t_{max}(a,b) = \max(a,b)$
Algebraisches Produkt: $t_{ap}(a,b) = a \cdot b$	Algebraische Summe: $t_{as}(a,b) = a + b - a \cdot b$
Begrenzte Differenz: $t_{bd}(a,b) = \max(0, a+b-1)$	Begrenzte Summe: $t_{bs}(a,b) = \min(1, a+b)$

Tabelle 2.2: t- und s-Normen

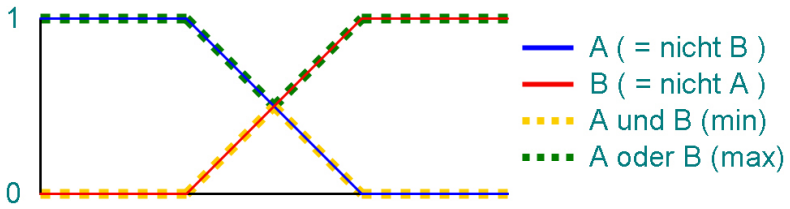


Abbildung 2.2: Anwendung logischer Operatoren auf Fuzzy-Sets

s-Norm ist der Minimum-Operator. Für das *ODER* wird die dazugehörige t-Norm, der Maximum-Operator, verwendet.

2.1.3 Linguistische Variable und Linguistische Terme

Bei Fuzzy-Steuerungen oder Fuzzy-Entscheidungssystemen hat man in der Regel eine bestimmte Anzahl von Eingangswerten, die über *linguistische Variablen*, wie etwa *Temperatur, Alter, Druck, Größe*, etc. angesprochen werden.

Am Beispiel eines Heizungssystems könnte eine der linguistischen Input-Variablen *temperature* heißen und einen exakten Wert (crisp Value) aus dem Bereich der reellen Zahlen beinhalten:

$$temperature = 22°C$$

Eine linguistische Variable hat einen Namen und und beinhaltet einen Wert aus dem Bereich der ihr zugeordneten *linguistischen Terme*. Die linguistischen Terme sind durch Fuzzy-Sets definiert.

$$temperature\ is\ cold$$

Alle linguistischen Terme einer linguistischen Variable bilden gemeinsam ein Term-Set. Im Falle der Temperatur könnte das Term-Set folgendermaßen aussehen:

$$T_{\text{temperature}} = \{cold, cool, ok, hot\}$$

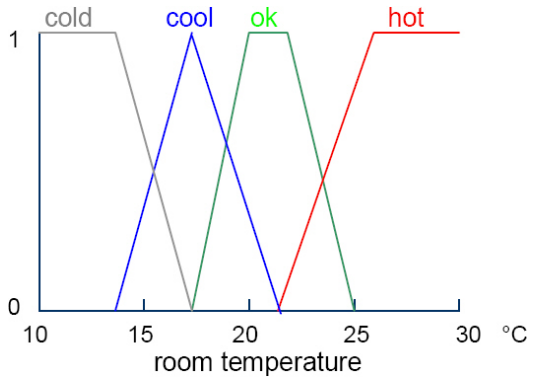


Abbildung 2.3: Linguistische Variable und Terme

Andere Beispiele für Linguistische Variable und dazugehörigem Term-Set sind:

Alter : jung, jugendlich, mittel, alt, sehr alt

Größe : sehr klein, klein, normal groß, riesig

Auf die gleiche Art und Weise wie die Eingänge, werden auch die Ausgänge einer Fuzzy-Steuerung über linguistische Variable und Terme beschrieben.

2.1.4 Fuzzy Regeln

Fuzzy Logic bietet dem Entwickler die Möglichkeit, komplexe Systeme auf Basis seines Wissens und seiner Erfahrung, mittels natürlichsprachlich formulierter Regeln zu beschreiben. Es sind dabei keine Systemmodellierungen oder komplexe mathematische Gleichungssysteme notwendig um den Zusammenhang zwischen Inputs und Outputs herzustellen. Fuzzy-Regeln sind auch für technisch weniger versierte Personen leicht zu lesen und zu verstehen.

Es sind meist nur einige wenige Regeln vonnöten um ein System zu definieren. Die gängigste Form der Fuzzy-basierten Wissensdarstellung ist die IF-THEN-Regel, auch *Fuzzy-Rule* genannt.

Die allgemeine Form einer Fuzzy-Rule ist

$$IF\ X\ IS\ A\ THEN\ Y\ IS\ B$$

wobei A und B linguistische Terme (durch Fuzzy-Sets definiert), und X und Y linguistische Variable sind. Der linke Teil der Regel ($X\ IS\ A$) wird *Bedingung* (engl. *condition*, *antecedent* oder *premise*), der rechte Teil ($Y\ IS\ B$) *Schlussfolgerung* (engl. *conclusion* oder *consequence*) genannt.

Die meisten Fuzzy-Steuerungen haben mehrere Eingangswerte (Input Values). Es ist daher auch möglich, dass die Bedingung einer Fuzzy-Rule eine Kombination mehrerer Eingänge beinhaltet, die durch logische Operatoren wie UND und ODER verbunden ist. Ein Beispiel dafür könnte folgendermaßen aussehen:

IF room temperature IS ok AND outside temperature IS cool

THEN turn heating on a little

2.1.5 Fuzzy Inferenz System

Fuzzy Inferenz Systeme (FIS) sind ein praktikabler Ansatz für die Lösung verschiedenartiger Probleme. Haupteinsatzgebiete sind Steuerungsautomatisation, Klassifizierung von Daten oder Entscheidungsfindung. Die Grundidee ist immer, Expertenwissen und Erfahrung einfließen zu lassen, wenn das erstellen eines exakten mathematischen Modells sehr aufwändig oder aufgrund der Komplexität des Systems praktisch unmöglich ist.

Es gibt verschiedene Arten von FIS. Die am häufigsten verwendete ist die *Mamdani-Inferenz*. Abbildung 2.4 zeigt das Grundschema eines Mamdani-FIS. In den folgenden Kapiteln werden die Einzelschritte im Detail erläutert. Nachdem alle Fuzzy-Sets und die Fuzzy-Rules definiert worden sind, arbeitet

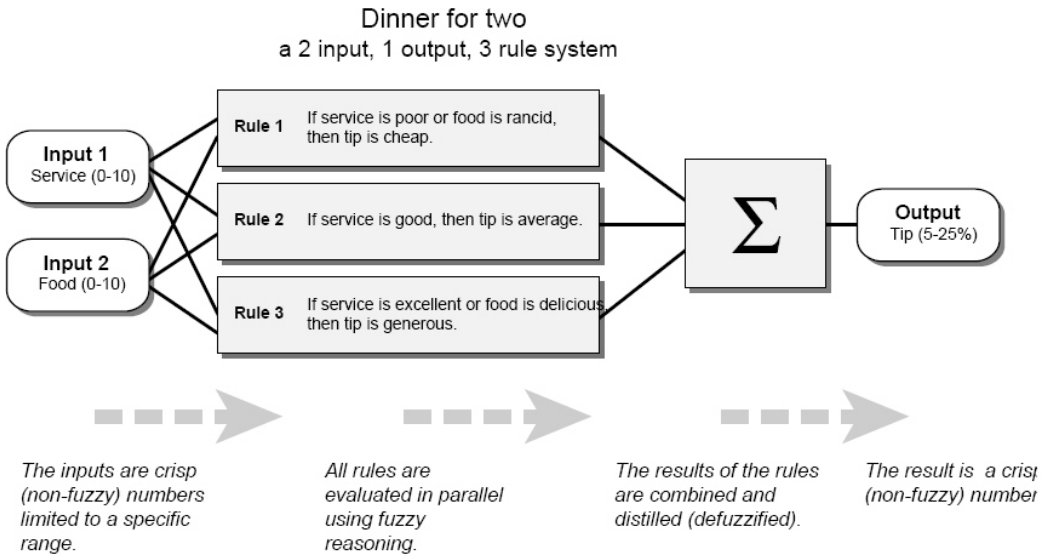


Abbildung 2.4: Fuzzy Inferenz System [24]

ein FIS nach einem in fünf Schritte gegliederten Algorithmus. Die einzelnen Schritte werden in den folgenden Abschnitten anhand von Beispielen im Detail erklärt.

Schritt 1: Fuzzifikation

Im ersten Schritt wird evaluiert welchen Grad an Zugehörigkeit (membership value) jeder Input-Value zu den entsprechenden Fuzzy-Sets hat. Abbildung 2.5 zeigt wie ein exakter Input-Value von $2,2\ ft$ zu 53 % *near* und 15 % *good* fuzzifiziert wird.

Schritt 2: Aggregation

Die Aggregation ermittelt den Wahrheitswert der gesamten Bedingung einer Fuzzy-Rule. Falls eine Bedingung aus eine Kombination (UND-, ODER-Verknüpfung) mehrerer Inputs besteht, so werden die Wahrheitswerte der Teilbedingungen (siehe Schritt 1: Fuzzifikation) in diesem Schritt so kombiniert, dass man genau einen Wahrheitswert aus dem Intervall $[0,1]$ für die gesamte Bedingung erhält. Für UND-Verknüpfungen wird eine t-Norm, für ODER-Verknüpfungen eine s-Norm verwendet.

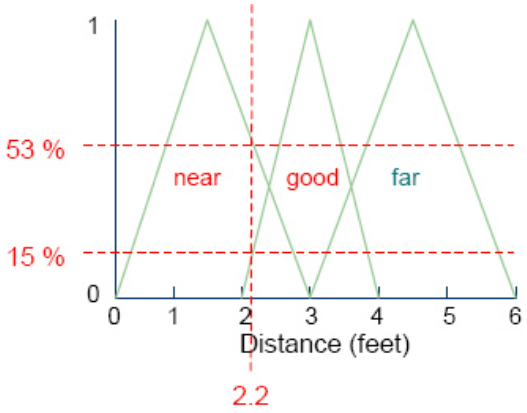


Abbildung 2.5: Fuzzifikation

In Tabelle 2.3 ist dargestellt, wie man die beiden Teilbedingungen *distance is near* und *distance is good* in der Aggregation kombiniert. In diesem Beispiel wird als t-Norm der Minimum-Operator und als s-Norm der Maximum-Operator verwendet.

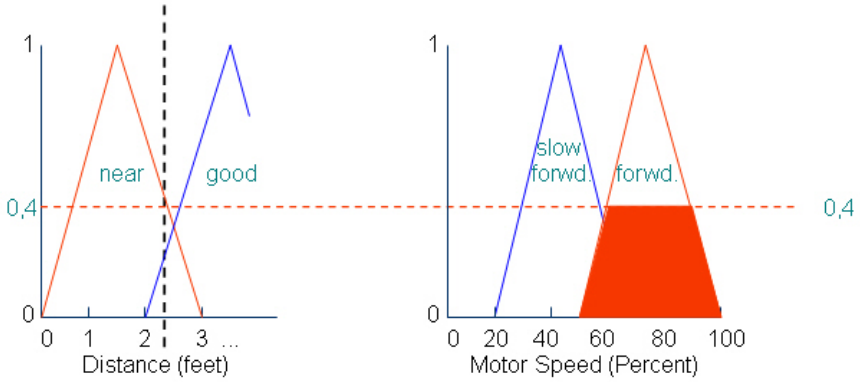
Bedingung der Fuzzy-Rule	**Fuzzifikation für *near***	**Fuzzifikation für *good***	**Aggregierter Wert**
near AND good	0.53	0.20	0.20
near OR good	0.53	0.20	0.53

Tabelle 2.3: Aggregation

Schritt 3: Implikation

Die Implikation überträgt den durch die Aggregation ermittelten Wahrheitswert der Bedingung einer Fuzzy-Rule auf das Output-Fuzzy-Set der Schlussfolgerung der Regel. Es wird eine t-Norm (z.B. der Minimum-Operator) verwendet um den Wahrheitswert auf das Output-Fuzzy-Set anzuwenden. Abbildung 2.6 zeigt das anhand eines Fuzzy-Systems mit zwei Regeln. Bei Verwendung des Minimum-Operators, wie in diesem Beispiel, wird das Output-Fuzzy-Set einfach in der Höhe abgeschnitten, die als Ergebnis aus der Aggregation hervorgeht.

Rule 1: `IF distance IS near THEN motor speed IS forward`



Rule 2: `IF distance IS good THEN motor speed IS slow forward`

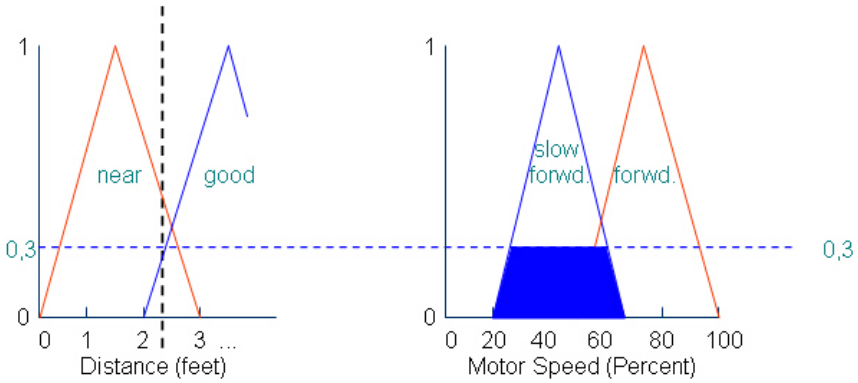


Abbildung 2.6: Implikation

Schritt 4: Akkumulation

Falls mehrere Fuzzy-Rules auf die gleiche Output-Variable schließen, wird eine Methode benötigt, welche die Implikationsergebnisse der betroffenen Regeln kombiniert. Die „abgeschnittenen“ Fuzzy-Sets werden unter Verwendung einer s-Norm kombiniert. Im Beispiel in Abbildung 2.7 wird der Maximum-Operator verwendet.

Schritt 5: Defuzzifikation

Das Ergebnis der Akkumulation ist ein Fuzzy-Set bzw. eine Membership-Funktion, die das Fuzzy-Set beschreibt. Ein Gerät, das über das Fuzzy-

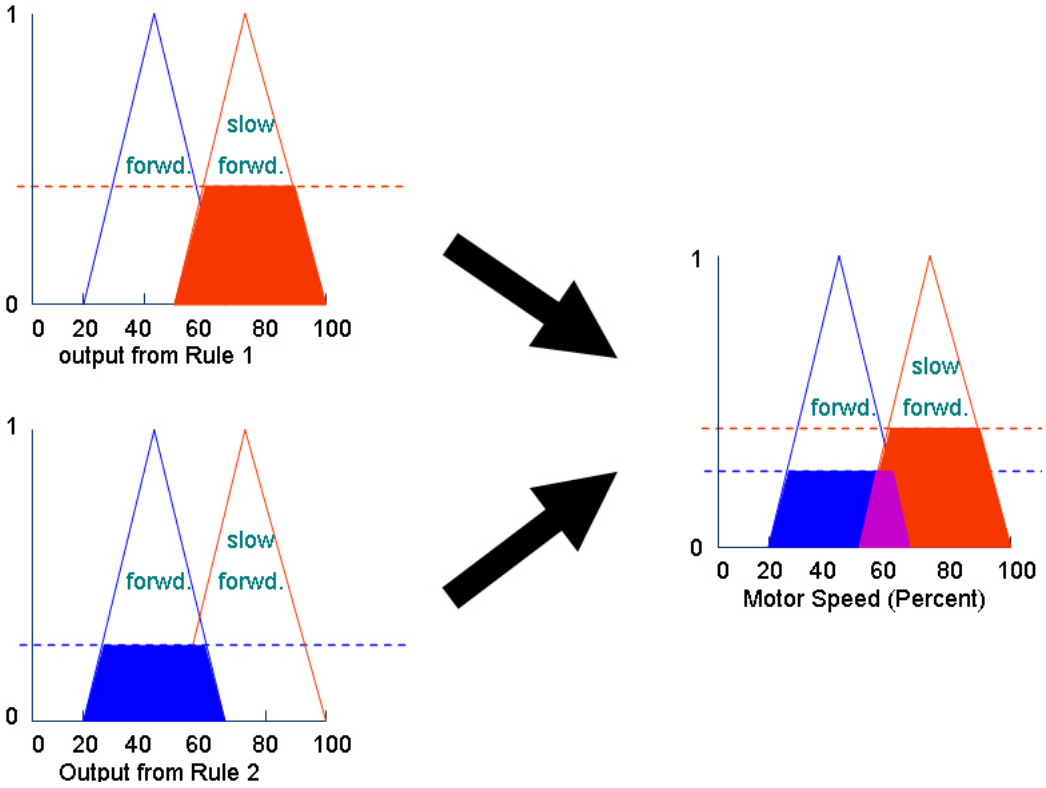


Abbildung 2.7: Akkumulation

System gesteuert werden soll, kann üblicherweise mit einem Fuzzy-Set nichts anfangen, sondern benötigt einen exakten Wert. Die Aufgabe der Defuzzifikation ist es nun, das Fuzzy-Set auf einen einzelnen Wert (crisp Value) zu reduzieren, der es am besten repräsentiert.

Es gibt viele verschiedene Defuzzifikationsmethoden. Die meistverwendete ist die *Schwerpunkt-Methode* (*Center of Gravity Method, CoG*). Bei dieser Methode wird der Ausgabewert bestimmt, indem zuerst der Schwerpunkt unter dem Graphen der Membership-Funktion ermittelt wird, und dieser dann auf die Abszisse projiziert wird (siehe Abbildung 2.8).

Der Ausgabewert x_{crisp} kann rechnerisch mittels folgender Formel ermittelt werden:

$$x_{crisp} = \frac{\int_{x=min}^{max} x \cdot \mu(x) dx}{\int_{x=min}^{max} \mu(x) dx}$$

mit X als Ergebnis der Defuzzifikation, x als Ausgabewert, μ als Membership-

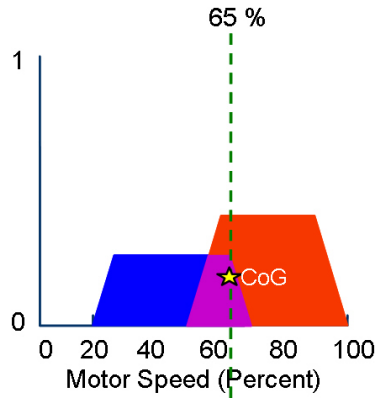


Abbildung 2.8: CoG Defuzzifikation

Funktion nach der Akkumulation und min, max als untere und obere Grenzen für die betroffenen Output-Variable.

2.2 Kohonen Self-Organizing-Feature-Map

Die im Jahr 1982 von *Tuevo Kohonen* entwickelte *Self-Organizing-Feature-Map (SOFM)* gehört zur Klasse der selbstorganisierenden Netze [14]. Man spricht in diesem Zusammenhang auch von *unüberwachtem Lernen (unsupervised learning)*. Das überwachte Lernen basiert darauf, dass dem Netz den ermittelten Output mit dem durch einen „Lehrer“ mitgeteilten richtigen Ergebnis vergleicht, und versucht, den Fehler zu minimieren. Ein Beispiel für ein neuronales Netz mit überwachtem Lernen ist das Multilayer-Perceptron mit Backpropagation-Algorithmus. Das hier verwendete Kohonen-Netzwerk hat im Gegensatz dazu keinen „Lehrer“ . Es weiß also nicht, welches Ergebnis das richtige wäre. Die Aufgabe dieses Netzes ist vielmehr die verschiedenen Inputs zu kategorisieren. Es werden also - vereinfacht gesagt - ähnliche Inputs als gleichartig erkannt. Oder, etwas technischer ausgedrückt, die Aufgabe des Netzes besteht darin, allein durch Eingabe von zufällig ausgewählten Vektoren eine innere Repräsentation des Eingaberaums zu bilden.

Die SOFM verfügt über eine bestimmte Anzahl von Eingabeeinheiten, deren Anzahl der Dimension der Trainingsvektoren entspricht. Zusätzlich besteht sie aus einer ein- oder zweidimensionalen Schicht von gitterförmig

angeordneten Neuronen (die eigentliche Feature-Map), die jeweils mit allen Eingabeeinheiten verbunden sind.[1][9] Die Aktivitäten werden von der Eingabeschicht hin zur Ausgabeschicht ausgebreitet. Abbildung 2.9 zeigt den Grundaufbau einer SOFM. Alle Neuronen der Eingabeschicht sind mit jedem anderen Neuron der Ausgabeschicht (Kartenschicht) verbunden. Jedes Neuron in der Ausgabeschicht ist eindeutig durch seine x und y Koordinaten bestimmt.[1]

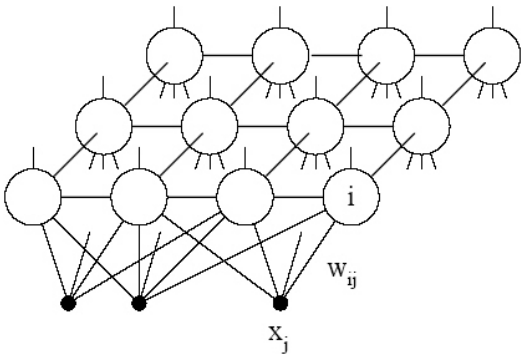


Abbildung 2.9: Aufbau einer SOFM [9]

Die Eingabeneuronen nehmen das externe Eingabesignal auf und leiten es weiter zur Ausgabeschicht. Das Hauptziel beim Lernen ist es die zu lernenden Muster auf die SOFM abzubilden, das heißt, es muss für jedes Muster ein Zentrum geben welches einem gelernten Muster entspricht. Um dieses Zentrum herum befinden sich Neuronen welche diesem Muster ähnlich sind. Daraus ergibt sich, dass ähnliche Eingabemuster auf die selbe Region in der SOFM abgebildet werden.

2.2.1 Trainieren einer SOFM

Beim Lernalgorithmus des Kohonen-Netzwerks handelt es sich um *kompetetives Lernen*. Der Name kommt daher, dass die Neuronen in einem Wettbewerb stehen. Nur genau ein Neuron - das sogenannte *Winner-Neuron* - wird bei der Eingabe aktiviert. Es werden bei jedem Trainingsschritt die Gewichte w_{ij} (siehe dazu Abbildung 2.9) in einer definierten Umgebung rund um das Winner-Neuron angepasst. Diese definierte Umgebung wird auch Nachbar-

schaft genannt und ist über den Radius r definiert (siehe Abbildung 2.10).

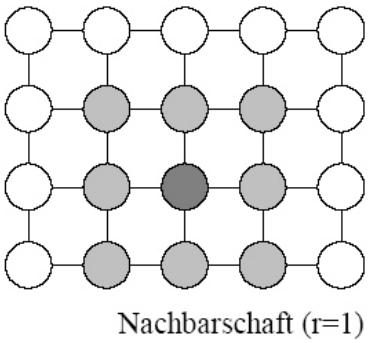


Abbildung 2.10: Winner-Neuron und Nachbarschaft

Ziel des Lernalgorithmus ist es, jedes Neuron der Ausgabeschicht auf einen bestimmten Bereich des Eingaberaums zu spezialisieren. Die Erregung des betroffenen Neurons ist bei Eingaben aus eben diesem bestimmten Bereich des Eingaberaums maximal. Dabei sollen benachbarte Bereiche des Eingaberaums auch benachbarten Neuronen zugeordnet werden. Diese Strukturierung wird dadurch erreicht, dass nicht nur das Gewicht des Winner-Neurons angepasst wird, sondern auch die Nachbarschaft des Winner-Neurons an jedem Lernschritt teilnimmt.

Konkret besteht der Kohonen-Lernalgorithmus aus folgenden Schritten[9]:

1. *Initialisierung:* Die Gewichtsvektoren w_i aller Neuronen i werden mit kleinen Werten initialisiert.

2. *Stimuluswahl:* Aus dem Eingaberaum $X \subseteq \mathbf{R}^n$ wird ein Vektor x gemäß einer gegebenen Wahrscheinlichkeitsverteilung zufällig ausgewählt.

3. *Bestimmung des Erregungszentrums:* Das Erregungszentrum ist dasjenige Neuron k, dessen Gewichtsvektor w_k dem Eingabevektor x am nächsten liegt:
$$\|w_k - x\| = \min \|w_i - x\|$$

4. *Anpassung der Gewichte:* Die Gewichtsvektoren aller Neuronen i in der Nachbarschaft des Erregungszentrums k mit Radius r werden um die

Differenz

$$\Delta w_i = \alpha(x - w_i)$$

verändert, wobei $\alpha > 0$ die Lernrate ist. Dies bedeutet, dass die Gewichtsvektoren in Richtung des Eingabevektors gezogen werden.

5. *Wiederholung:* Die Schritte 2 bis 4 werden wiederholt und dabei die Lernrate α sowie der Nachbarschaftsradius r sukzessive verkleinert, bis sich das Netz stabilisiert.

Kapitel 3

Implementierung

Die wesentlichen Bestandteile des Systems sind

- ein Roboter mit zwei unabhängig voneinander betriebenen Elektromotoren für Geschwindigkeitsregulierung und Navigation,
- eine handelsübliche Webcam mit USB-Anschluß,
- ein PC (Notebook) zur Bildverarbeitung und zur Anwendung der KI-Methoden. Beide Aufgaben übernimmt *MATLAB®*, ein Softwarepaket, das von der Herstellerfirma *The Mathworks®* wie folgt beschrieben wird: *„MATLAB ist eine intuitive Sprache und eine Oberfläche für technische Berechnungen. Es besteht aus einem mathematischen Kern und modernen Grafik-Werkzeugen für Datenanalyse, Visualisierung sowie die Entwicklung von Algorithmen und Anwendungen. Weltweit verlassen sich eine große Zahl von Wissenschaftlern und Ingenieuren auf MATLAB als strategische Entwicklungsplattform mit ihren über 600 mathematischen, statistischen und ingenieurwissenschaftlichen Funktionen.“* [25]
- und ein Mikrocontroller (*Microchip® PIC16F877®*) zur Kommunikation mit dem PC und zur Ansteuerung der Motoren.

3.1 Wege der Signalverarbeitung

Abbildung 3.1 gibt einen schematischen Überblick über Aufbau und Zusammenspiel der wesentlichen Komponenten des Systems.

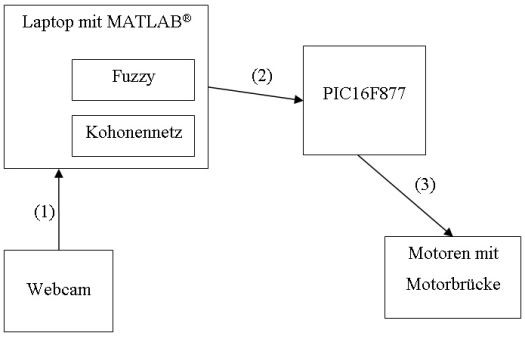


Abbildung 3.1: Wege der Signalverarbeitung

1. Die Webcam nimmt Bilder auf und sendet diese an *MATLAB®*, das am Laptop läuft.

2. *MATLAB®* analysiert die Bilder, erkennt eine Linie und sendet die Lagedaten der Linie als Eingangsparameter an ein Fuzzy Inferenz System, welches als Ausgabe die Daten für die Ansteuerung der beiden Motoren liefert. Weiters wird im Bild nach Verkehrszeichen gesucht, die auf der Linie angebracht sind und dem Roboter die Fahrtgeschwindigkeit vorgeben. Die Motorsteuerungsdaten - ermittelt aus der Richtung der Linie und der Art des Verkehrszeichens - werden über eine serielle Schnittstelle an den *PIC16F877®* Mikrocontroller übermittelt. Die dafür notwendigen Routinen hab ich in M®, der MATLAB®-eigenen Programmiersprache implementiert. Die wesentlichen Teile des Source-Codes sind im Anhang beigefügt.

3. Der Mikrocontroller übernimmt die Ansteuerung der Motorbrücke. Die Geschwindigkeit jedes einzelnen der beiden Motoren wird über *Pulsweitenmodulation (PWM)* reguliert. Diesen Teil der Software habe ich in PIC-C, einer reduzierten C-Implementierung speziell für PIC® Mikrocontroller, geschrieben.

3.2 Hardware

Ein Chassis auf Rädern ist eines der einfachsten Konstruktionen eines fahrenden Roboters. Die Anzahl der Räder spielt dabei eine bedeutende Rolle. Die gängigsten Radroboter haben drei oder vier Räder. Im Bezug auf die Bodenhaftung kann die Anzahl der Räder sehr wesentlich sein. Ein Vehikel mit sehr vielen (angetriebenen) Rädern verhält sich ähnlich einem Kettenfahrzeug. Obwohl auf unebenem Terrain nicht alle Räder zu jedem Zeitpunkt Bodenkontakt haben ist ein Fortkommen möglich. Schwieriger gestaltet sich allerdings die Steuerung der Fahrtrichtung.

Abbildung 3.2: Radanzahl und Bodenkontakt [20]

Abbildung 3.2 illustriert die stabilere Lage eines dreirädrigen gegenüber einem vierrädrigen Roboter auf unebenem Untergrund. Bei vier Rädern kann es passieren, dass ein Antriebsrad keine Bodenhaftung mehr hat. Ein Chassis mit drei Rädern dagegen hat immer mit allen Rädern Bodenkontakt, da durch beliebige drei Punkte im Raum eine Ebene gelegt werden kann. Ein weiterer Vorteil dieser Bauart ist die einfache und effektive Steuerung. Zwei voneinander unabhängig angetriebene Räder am Heck und ein frei bewegliches Stützrad vorne ermöglichen sehr enge Radien in jede Richtung.

Für den hier beschriebenen Roboter wird das dreirädrige Chassis aus der Zeitschrift *Real Robots*®[20] inklusive Getriebe, zwei Elektromotoren und einer Motorbrücke zum voneinander unabhängigen Betrieb der beiden Hinterräder verwendet.

Die Hauptbestandteile der Motorbrücke sind vier im Stromkreis angeord-

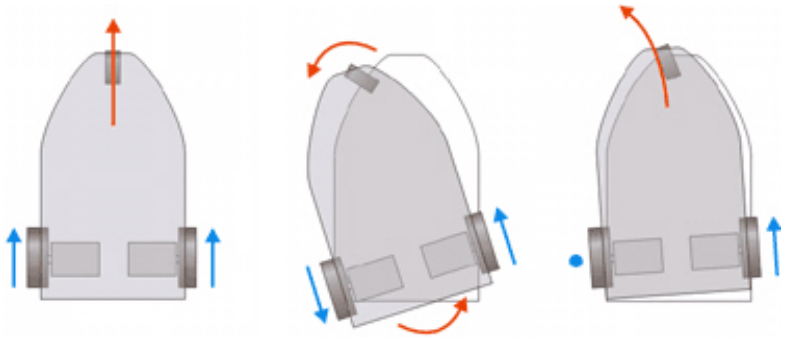


Abbildung 3.3: Bewegung eines dreirädrigen Roboters [6]

nete Transistoren, die wie simple Schalter funktionieren. Jeweils zwei Schalter werden, wie in Abbildung 3.4 dargestellt, gemeinsam geschalten. Aus den beiden Kombinationen *SW1* + *SW4* (Abbildung 3.4(b)) bzw. *SW2* + *SW3* (Abbildung 3.4(c)) ergibt sich die Laufrichtung des Motors, der in der Abbildung 3.3 durch den Buchstaben *m* gekennzeichnet ist.

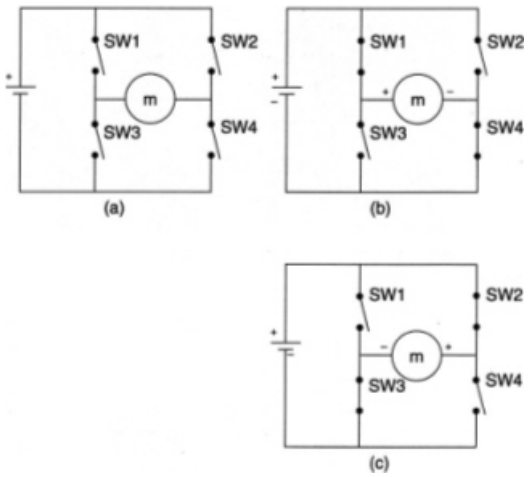


Abbildung 3.4: Motorbrückenschaltung [12]

Die Motorbrücke besitzt Pins sowohl für eine +6V als auch für eine +9V Stromversorgung und weitere vier Pins für die Ansteuerung der Motoren. Zur Regelung der Laufgeschwindigkeit der Motoren wird Pulsweitenmodulation (PWM) eingesetzt. PWM wird sowohl von der Motorbrücke als auch vom Mikrocontroller PIC16F877® unterstützt. Der PWM-Ausgang des Mikrocontrollers ermöglicht das einfache Erzeugen von Impulsen mit einem ein-

stellbaren Taktverhältnis.[21]

3.3 Kommunikation

Für die Kommunikation zwischen dem Notebook und dem Mikrocontroller am Roboter habe ich *DCP* (*Device Communication Protocol*), ein speziell für den einfachen Datenaustausch zwischen verschiedenen Devices optimiertes Protokoll, entwickelt und implementiert.

Ein für diese Anwendung geeignetes Kommunikationsprotokoll muss meiner Meinung nach folgende Bedingungen erfüllen:

Einfachheit: Das Protokoll muss einfach implementierbar sein. Eine Implementierung auf einem Mikrocontrollers mit sehr beschränkten RAM- und CPU-Ressourcen muss möglich sein.

Effizienz: Der Kommunikationsoverhead soll möglichst gering sein, um nicht unnötig Bandbreite zu verschwenden.

Mehrere Devices: Über ein und dieselbe Verbindung müssen mittels des Protokolls Daten für mehrere Devices übertragen werden können.

Symmetrie: Die Implementierung auf Sender- und Empfängerseite muss ident sein.

Variable Datengröße: Die Länge des Datenpakets per Device muss variabel sein, da die Daten unterschiedliche Devices auch unterschiedliche Datentypen haben können.

Flexibilität: Nicht jedes Device muss zu jedem Zeitpunkt einen Wert liefern.

3.3.1 Device Communication Protocol (DCP)

Abbildung 3.5 beschreibt die Struktur eines DCP-Paketes. Es besteht aus Daten für mehrere Devices. Für jedes dieser Devices enthält es ein Header

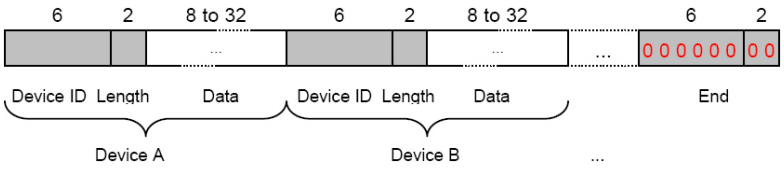


Abbildung 3.5: DCP Paketaufbau

Byte, bestehend aus Device Identifier (DID) und Länge der folgenden Daten (LoD). Durch ein Null-Byte wird das Paket abgeschlossen.

Im Detail setzt sich ein DCP Paket aus folgenden Elementen zusammen:

Device Identifier (DID): 6 bit -> bis zu 64 Devices ($000000_{bin}..111111_{bin}$)

Länge der Daten (LoD): 2 bit -> Variable Datenlänge für jedes Device, bis zu 4 Bytes:

- $\mathbf{01}_{bin}$ ... byte (8 bit)
- $\mathbf{10}_{bin}$ ... word (16 bit)
- $\mathbf{11}_{bin}$ ... double (32 bit)
- $\mathbf{00}_{bin}$ ... Ende der Daten (falls DID = 0)

Daten: 8, 16 or 32 bit (siehe "Länge der Daten") -> bis zu $2^{32} \approx 4.3 * 10^9$ verschiedene Werte

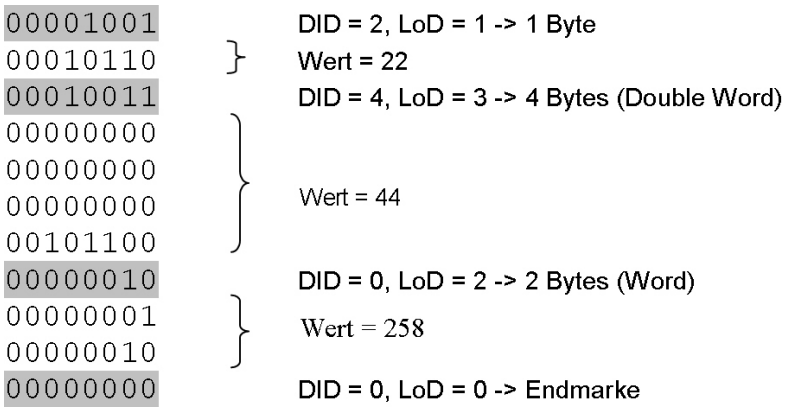


Abbildung 3.6: DCP Beispielpaket

Während der Kommunikation zweier Kommunikationspartner unter Verwendung von DCP ist zu einem bestimmten Zeitpunkt immer einer der beiden Partner Sender, der andere Empfänger. Nach Übertragung eines vollständigen DCP-Pakets tauschen die Kommunikationspartner die Rollen und ein Paket wird in die andere Richtung übertragen. Der genaue Ablauf des Algorithmus ist in Abbildung 3.7 ersichtlich.

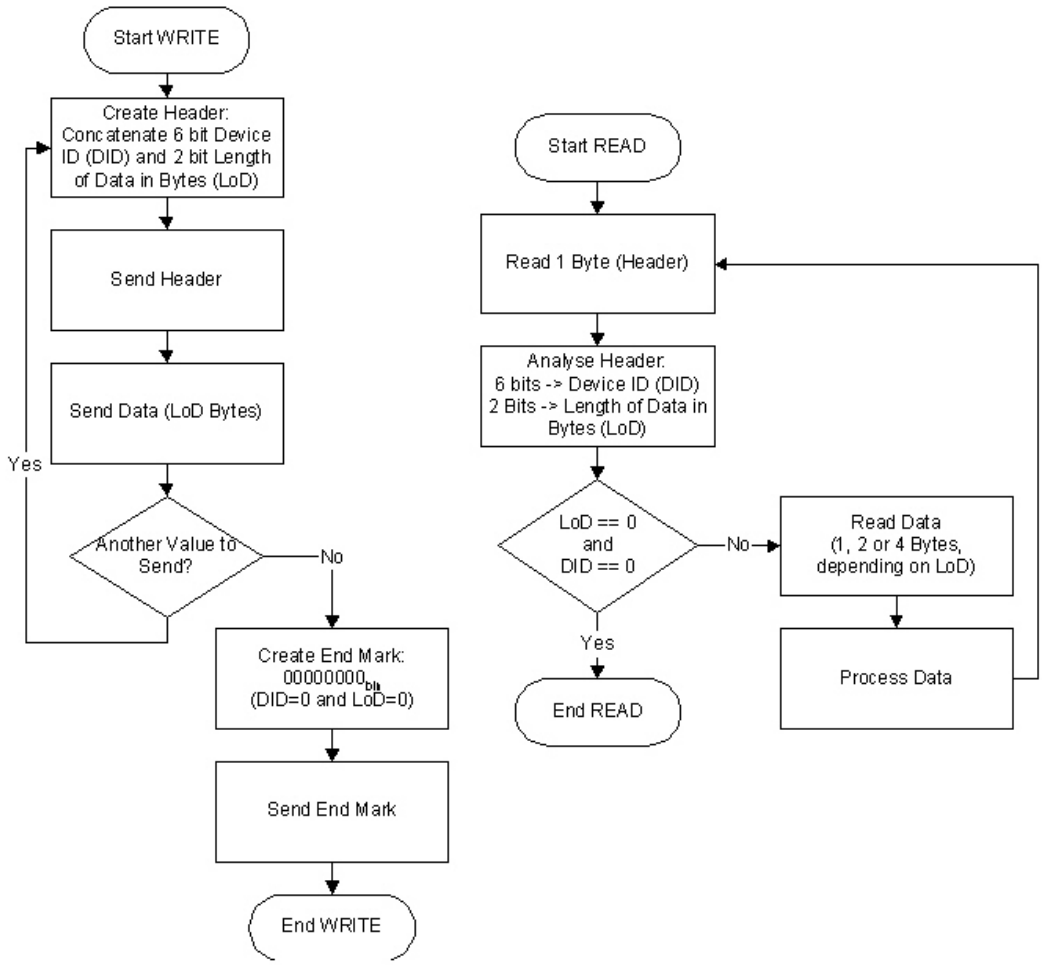


Abbildung 3.7: DCP Programmstruktur

Kapitel 4

Navigation entlang einer Linie

Die Steuerung mobiler Roboter anhand von Sensoren ist eine grundlegende Aufgabenstellung in der Robotik. Der Roboter wird über die Geschwindigkeit jedes der beiden Antriebsräder kontrolliert. Ein frei bewegliches Stützrad im vorderen Teil des Roboters hat keinen Einfluss auf den Weg des Roboters. Die beiden unabhängig voneinander betriebenen Gleichstrommotoren werden folglich für beides verwendet, für Antrieb und Steuerung (siehe Abbildung 3.3).

4.1 Identifizieren der Linie

Um einer Linie folgen zu können, muss diese erstmal „gefunden“ werden. Dieser Prozess ist in drei Teilschritte untergliedert. Im ersten Schritt wird von der Kamera, die an der Front des Vehikels montiert ist, ein Bild aufgenommen. Dieses Bild wird im zweiten Schritt bezüglich des Farbmodells transformiert, so dass für die Bildanalyse maximale Unabhängigkeit von den vorherrschenden Lichtverhältnissen erreicht wird. Eine *Sliding Window* (gleitendes Fenster) Analyse liefert letztlich die erforderlichen Daten über den Verlauf der gesuchten Linie, welche dann an das Fuzzy-System weitergegeben werden.

4.1.1 Schritt 1: Aufnahme eines Bildes

Die Kamera liefert bis zu 30 Bildern pro Sekunde. Die Anzahl der tatsächlich pro Sekunde verarbeiteten Bilder hängt letztlich von der Leistungsfähigkeit des Rechners an dem die Kamera hängt, der geforderten Bildqualität und der Komplexität der angewendeten Bildverarbeitungsalgorithmen ab.

Mittels einer sogenannten *Frame-Grabber-Software* wird ein einzelnes Bild aufgenommen und für die weitere Analyse als Bitmap im RGB-Farbmodell zwischengespeichert. Als guter Kompromiss zwischen Bildqualität und Verarbeitungszeit hat sich eine Auflösung von 352×288 Bildpunkten herausgestellt.

4.1.2 Schritt 2: Transformation in das HSB-Farbmodell

Farben sind in der Regel aus mehreren Bestandteilen zusammengesetzt. Je nach verwendetem Farbmodell kann man eine Farbe in ihre Primärfarben *Rot*, *Grün*, *Blau* (additives Farbmodell) oder *Cyan*, *Magenta*, *Gelb* (subtraktives Farbmodell) aufsplitten. Im Gegensatz dazu definiert das HSB-Farbmodell einen Bildpunkt über die Informationen *Farbwert*, *Sättigung* und *Helligkeit*.[26]

In dieser Arbeit wird lediglich auf das additive und das HSB-Farbmodell näher eingegangen, da die anderen Farbmodelle für die beschriebene Anwendung keine Relevanz haben.

Additives Farbmodell (RGB)

RGB steht für die drei Primärfarben des *additiven Farbmodells*, nämlich rot, grün und blau. Das bedeutet, dass ein Bildpunkt über drei Zahlen definiert wird, wobei jede dieser Zahlen die Intensität jeweils einer der drei Farben repräsentiert.

Das *RGB-Farbmodell* ist das am häufigsten verwendete Modell zur Beschreibung von Farben im Computer. Additives Farbmodell bedeutet, dass die Beiträge der einzelnen RGB-Primärfarben übereinandergelegt - also addiert - werden und so das Gesamtergebnis liefern. Das RGB-Modell wird

beispielsweise bei Bildschirmen verwendet.

Innerhalb des RGB-Systems können Designer auch durch die Verschmelzung dieser drei Primärfarben eine Farbspektrum erzeugen. Die Kombination jeweils zweier Primärfarben erschafft drei Sekundärfarben: türkis, fuchsinrot und gelb. Wie bereits erwähnt, erzeugt die Kombination aller drei Primärfarben weißes Licht. Ein RGB-Wert von 255,255,255 erzeugt also Weiß. Die vollständige Abwesenheit der drei Primärfarben (RGB: 0,0,0) erzeugt Schwarz. Abbildung 4.1 veranschaulicht die Beschreibung von Farben über ihre RGB-Werte.

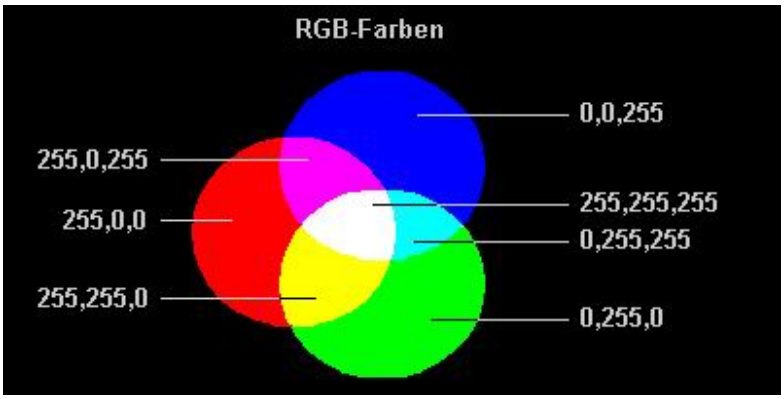


Abbildung 4.1: RGB-Farben

HSB-Farbmodell

Im Gegensatz zum RGB-Modell basiert das *HSB-Farbmodell* auf den intuitiven Methoden, die etwa von Malern eingesetzt werden. Anstelle der Angabe des Rot-, Grün- und Blauanteils findet eine Qualifizierung von Farben mittels *Farbton* (***h**ue*), *Sättigung* (***s**aturation*) und *Helligkeit* (***b**rightness*) statt. Da der Helligkeitswert oftmals auch schlicht mit ***V**alue* bezeichnet wird, wird dieses Modell auch *HSV-Farbmodell* genannt. Dabei wird die Farbe durch den Farbton bestimmt. Als Wert für die Farbe wird oft der entsprechende Winkel (0° bis 360°) im HSB-Farbenkreis (siehe Abbildung 4.2) herangezogen. Er besagt, ob es sich zum Beispiel um einen Blau-, Grün- oder Gelbton handelt. Der Grauanteil einer Farbe wird durch die Sättigung bestimmt. Wird beispielsweise die Sättigung verringert, so erhöht sich der Grauanteil. Bei

keiner Sättigung erscheint jede Farbe als Grau. Farben mit nur einem sehr geringem Sättigungsgrad bezeichnet man auch als trübe Farben. Erhöht man den Sättigungsgrad, so erscheint eine Farbe reiner. Für den Begriff Helligkeit gilt folgendes: Ohne Helligkeit wird jeder Farbton zu Schwarz, und mit maximaler Helligkeit wird jeder Farbton zu Weiß.[8]

Abbildung 4.2: HSB-Farbenkreis

Abbildung 4.2 zeigt einen Snapshot aus *CorelDraw*® der einen Querschnitt des HSB-Zylinders, auch HSB-Farbenkreis genannt, darstellt. Der markierte Farbpunkt hat dabei die Koordinaten $H = 0$, $S = 0$ und $B = 0$. Im Kreis können dabei die Werte für H (Winkel im Farbenkreis) und S (Abstand vom Mittelpunkt) dargestellt werden; die Größe B wird auf dem Balken daneben dargestellt.

Umwandlung von RGB nach HSB

Um bei der Bildanalyse möglichst unabhängig von den Beleuchtungseinflüssen aus der Umgebung zu sein, wird das Bild vom RGB- in das HSB-Farbmodell umgewandelt. Das HSB-Modell eignet sich wesentlich besser dafür, gleiche Farben unter unterschiedlicher Beleuchtung als gleich zu identifizieren, als das RGB-Modell. Der in Abbildung 4.3 beschriebene Algorithmus zeigt die Umrechnung eines Bildpunktes von RGB nach HSB.

```
var_R = ( R / 255 )                    //RGB values = From 0 to 255
var_G = ( G / 255 )
var_B = ( B / 255 )

var_Min = min( var_R, var_G, var_B )   //Min. value of RGB
var_Max = max( var_R, var_G, var_B )   //Max. value of RGB
del_Max = var_Max - var_Min            //Delta RGB value

V = var_Max

if ( del_Max == 0 )                    //This is a gray, no chroma...
{
   H = 0                               //HSV results = From 0 to 1
   S = 0
}
else                                   //Chromatic data...
{
   S = del_Max / var_Max

   del_R = ( ( ( var_Max - var_R ) / 6 ) + ( del_Max / 2 ) ) / del_Max
   del_G = ( ( ( var_Max - var_G ) / 6 ) + ( del_Max / 2 ) ) / del_Max
   del_B = ( ( ( var_Max - var_B ) / 6 ) + ( del_Max / 2 ) ) / del_Max

   if      ( var_R == var_Max ) H = del_B - del_G
   else if ( var_G == var_Max ) H = ( 1 / 3 ) + del_R - del_B
   else if ( var_B == var_Max ) H = ( 2 / 3 ) + del_G - del_R

   if ( H < 0 ) ; H += 1
   if ( H > 1 ) ; H -= 1
}
```

Abbildung 4.3: Umwandlung von RGB nach HSB (HSV)[7]

4.1.3 Schritt 3: Sliding Window Analyse

Um die Richtung zu bestimmen, in die der Roboter seine Fahrt fortsetzen soll, muss die Linie im Bild gefunden, und deren Lage ermittelt werden. Die zwei wesentlichen Parameter sind Winkel und Offset der Linie. Unter Offset versteht man die horizontale Abweichung der Linie von der Bildmitte.

Um die Lage der Linie zu bestimmen, muss nicht das ganze Bild durchsucht werden. Unter der Annahme, dass der Abschnitt der Linie, der sich im aktuellen Bildbereich befindet gerade ist, wird die Analyse auf den oberen und den untere Bildrand beschränkt. Es werden also nur die beiden Endpunkte des in Bild sichtbaren Linienabschnitts ermittelt.

Um beispielsweise die x-Koordinate der Linie am oberen Bildrand zu er-

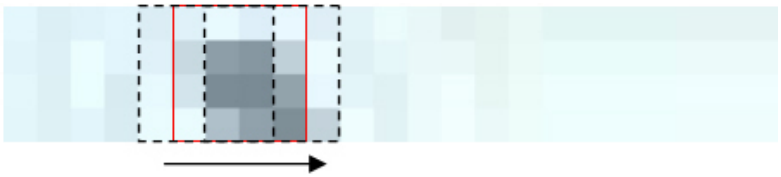


Abbildung 4.4: Sliding Window

mitteln läuft ein *Sliding Window* von $n \times n$ Pixel so wie in Abbildung 4.4 dargestellt von links nach rechts entlang des oberen Bildrandes pixelweise über das ganze Bild. Die Farbwerte (HSB) des jeweiligen Quadrats werden gemittelt und mit definierten Schwellwerten für Farbton und Sättigung verglichen. Auf diese Weise wird ermittelt, ob sich das Sliding Window gerade auf der Linie befindet oder nicht.

Die Mitte aller Sliding Windows die auf der Linie liegen ergeben die obere Linienkoordinate x_{top}. Das gleiche Verfahren wird am unteren Bildrand angewendet um die Koordinate x_{bottom} zu bestimmen. Abbildung 4.5 zeigt, wie aus diesen beiden Koordinaten die wesentlichen Parameter Winkel und Offset berechnet werden. *height* und *width* stehen für die Höhe und die Breite des Bildes in Pixel.

$$\Delta x = \frac{x_{bottom} - \frac{width}{2}}{width} \cdot 200 \quad \text{(siehe obere Skala von Abbildung 4.5)}$$

$$\alpha = \arctan\left(\frac{x_{bottom} - x_{top}}{height}\right) \quad \text{(siehe untere Skala von Abbildung 4.5)}$$

4.2 Fuzzy Logic für die Roboternavigation

Um den Roboter auf Kurs zu halten, wird ein Fuzzy System mit Mamdani-Inferenz verwendet. Das System hat zwei Eingänge und einen Ausgang. In der Fuzzy-Aggregation wird das logische *und* durch den Minimum-, das logische *oder* durch den Maximum-Operator repräsentiert. Für die Implikation wird ebenfalls das Minimum, und für die Akkumulation das Maximum verwendet. Defuzzifiziert wird nach der Schwerpunktmethode.

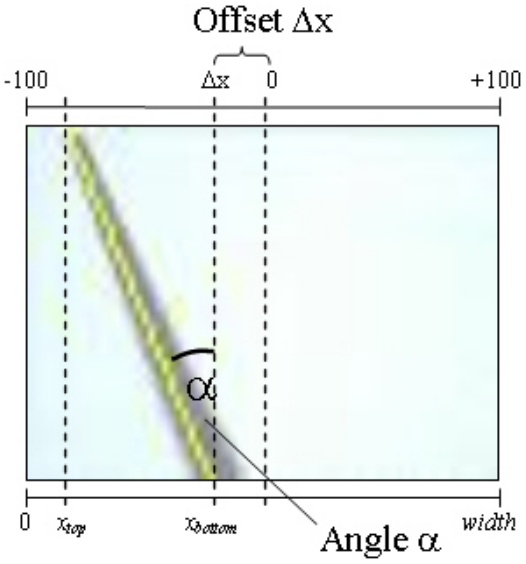


Abbildung 4.5: Winkel und Offset der Linie

4.2.1 Input Fuzzy Sets

Die zwei Input-Fuzzy-Sets des Systems (siehe Abbildung 4.6) beschreiben die linguistischen Variablen *Angle* (Winkel) und *Offset* (Versatz, Abweichung von der Bildmitte). Beide Werte kommen als Ergebnis aus der Bildanalyse. Die linguistischen Terme für den Angle sind: *farneg, neg, zero, pos, farpos*; und für Offset: *farleft, left, centre, right,* farright.

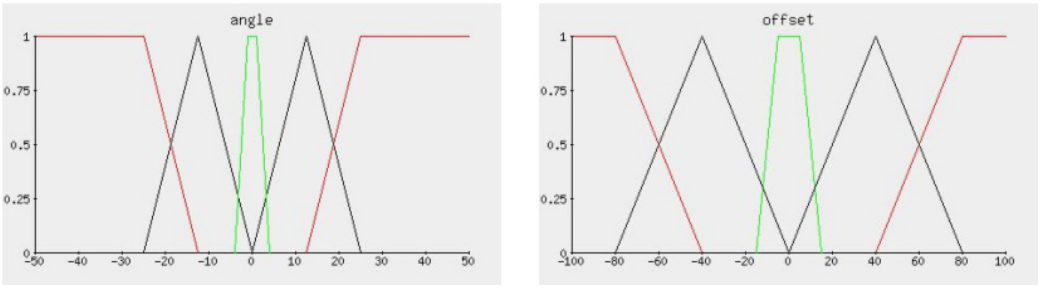


Abbildung 4.6: Fuzzy Input Variable: Winkel and Offset

4.2.2 Output Fuzzy Sets

Das Fuzzy-System liefert als Ergebnis die *Richtung*, die der Roboter einschlagen soll (siehe Abbildung 4.7). Die linguistischen Terme der Variable *direction* (Richtung) heißen: *lefter, left, straight, right, righter*. Das Ergebnis liegt im Intervall $[-100, 100]$, und beschreibt Richtungen von *stark nach links* bis *stark nach rechts*.

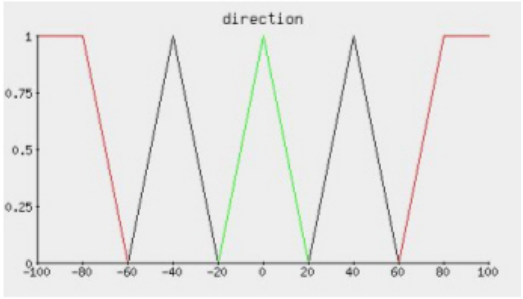


Abbildung 4.7: Fuzzy Output Variable: Richtung

4.2.3 Fuzzy Regeln

Folgende fünf Fuzzy-Regeln werden verwendet um den Roboter entlang der Linie zu navigieren.

- IF angle IS farneg OR offset IS farleft THEN direction IS righter
- IF angle IS neg OR offset IS left THEN direction IS right
- IF angle IS zero OR offset IS centre THEN direction IS straight
- IF angle IS pos OR offset IS right THEN direction IS left
- IF angle IS farpos OR offset IS farright THEN direction IS lefter

4.2.4 Navigationsfunktion

Das oben beschriebene Fuzzy-Inferenz-System (FIS) beschreibt die Funktion:

$$f(angle, offset) = direction$$

Abbildung 4.8 zeigt eine grafische Repräsentation dieser Navigationsfunktion.

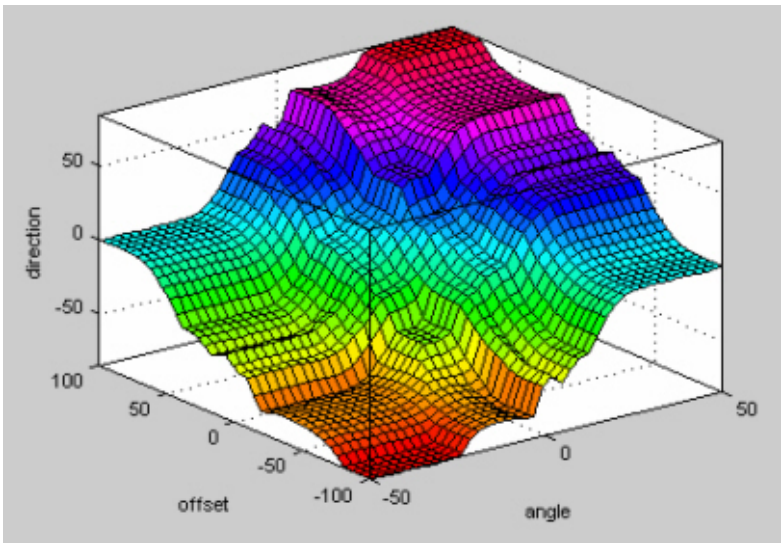


Abbildung 4.8: Navigationsfunktion

4.2.5 Ergebnistransformation

Der Ausgabewert des Fuzzy-Systems ist eine reelle Zahl im Intervall $[-100, 100]$. Der Roboter wird vom Mikrocontroller über zwei vorzeichenlose 8-Bit-Integerwerte gesteuert. Ein Wert gibt die Geschwindigkeit des rechten, der andere die des linke Motors an. 0 bedeutet Stillstand, 255 maximale Geschwindigkeit.

Daraus ergibt sich folgender Zusammenhang zwischen Richtung (*direction*) und Geschwindigkeit der einzelnen Motoren ($speed_L$ und $speed_R$):

$$speed_L = \begin{cases} 255 + round\left(direction \cdot \frac{255}{100}\right) & \text{if } direction < 0 \\ 255 & \text{if } direction \geq 0 \end{cases}$$

$$speed_R = \begin{cases} 255 & \text{if } direction < 0 \\ 255 - round\left(direction \cdot \frac{255}{100}\right) & \text{if } direction \geq 0 \end{cases}$$

Falls beispielsweise die Richtung 0 ist (geradeaus), werden nach obiger Formel beide Motoren gleich, und zwar mit 255 (Maximalgeschwindigkeit)

angesteuert. Bei einer Richtung kleiner als 0 (nach links) behält der rechte Motor Maximalgeschwindigkeit bei, und der linke Motor läuft entsprechend langsamer. Bei einer Richtung größer als 0 (nach rechts) ist das Verhalten genau umgekehrt.

Die Formeln liefern also die maximale Geschwindigkeit für die angegebene Richtung. Zur Verringerung der Geschwindigkeit werden beide Geschwindigkeitswerte ($speed_L$ und $speed_R$) mit dem gleichen Faktor aus dem Intervall $[0, 1]$ multipliziert.

Kapitel 5

Erkennung von Verkehrszeichen

Die Verkehrszeichen, die erkannt werden sollen, sind flach liegend auf der Linie positioniert (siehe Abbildung 1.1, die vom Roboter mittels Fuzzy-Logic verfolgt wird.

5.1 Finden des Verkehrszeichen und Bildtransformation

Die Webcam am vorderen Ende des Roboters nimmt Bilder der Linie auf. Da die Verkehrszeichen genau auf der Linie liegen, erscheint es im Bild, unmittelbar bevor der Roboter es passiert.

Sobald das Verkehrszeichen vollständig im Bild zu sehen ist, muss es für die Verarbeitung in einem neuronalen Netzwerk aufbereitet werden. Dafür sind einig Transformationsschritte notwendig, die im Folgenden beschrieben werden.

5.1.1 Schritt 1: Drehen

Abbildung 5.1 zeigt das Bild so, wie es von der Kamera übermittelt wird. Das Verkehrszeichen ist in seiner Lage genau nach der Linie ausgerichtet. Folglich ist es Bei jedem Winkel der Linie ungleich 0 verdreht im Bild zu sehen. Um diese Schieflage zu korrigieren wird das gesamte Bild um den Winkel der

Abbildung 5.1: Originalbild

Linie (Wert α in Abbildung 4.5) gedreht. Die Linie ist dann genau senkrecht im Bild, und das Verkehrszeichen ist gerade ausgerichtet. Des Ergebnis dieser Drehung ist in Abbildung 5.2 dargestellt.

Abbildung 5.2: Bild nach der Drehung

5.1.2 Schritt 2: Ausschneiden des Verkehrszeichens

Nachdem das Bild nun in der gerade gerichtet ist, muss die genaue Position und die Größe des Verkehrszeichens ermittelt werden. Dazu wird - genau wie zur Identifizierung der Linie - ein Sliding-Window-Verfahren verwendet. Diesmal wird das gesamte Bild zeilenweise analysiert. Es werden die extremen (oberster, unterster, linkster, rechtester) roten Punkte im Bild gesucht (siehe Abbildung 5.3). Es wird vorausgesetzt, dass im Bild außerhalb des Verkehrszeichens keine roten Punkte existieren.

Das Verkehrszeichen wird im nun exakt ausgeschnitten. Das Ergebnis dieses Schrittes ist in Abbildung 5.4 zu sehen.

Abbildung 5.3: Extremstellen des Verkehrszeichens

Abbildung 5.4: Ausgeschnittenes Verkehrszeichen

5.1.3 Schritt 3: Größe normieren

Wie in Abbildung 5.4 ersichtlich, ist der ausgeschnittene Teil des Bildes kein Quadrat, sondern ein Rechteck. Folglich ist das Verkehrszeichen selbst aufgrund der Kameraposition stark verzerrt. Der Grad der Verzerrung ist unterschiedlich, je nachdem wie weit der Roboter noch von dem Verkehrszeichen entfernt ist.

Um die weitere Verarbeitung zu vereinfachen werden alle aufgenommenen Bilder auf eine Standardgröße von 20 x 20 Pixel, wie in Abbildung 5.5 dargestellt, normiert.

Abbildung 5.5: Größennormiertes Verkehrszeichen

5.1.4 Schritt 4: Entfärben

In diesem Schritt wird das Farbbild aus Abbildung 5.5 zu einem Bild aus Grauwerten reduziert. Die Standardtransformation bringt nur ein mäßig gutes Ergebnis, da nicht das volle Spektrum an verfügbaren Grauwerten ausgereizt wird. Je nach Lichtverhältnissen liegen die Werte in einem anderen Bereich. Doch nie werden alle 256 Grauwerte verwendet.

Um unabhängig von den Lichtverhältnissen und geringfügigen Farbunterschieden in den Verkehrszeichen vergleichbare Bilder als Input für das neuronale Netz zu erhalten, werden alle Werte so normiert, dass der gesamte Wertebereich von 0 bis 255 verwendet wird. Das hellste Pixel erhält dann den Wert 255 (weiß), das dunkelste den Wert 0 (schwarz), auch wenn ihre Werte im Originalbild wesentlich näher beieinander liegen. Die Werte aller anderen Pixel liegen entsprechend linearer Transformation dazwischen. In der folgenden Transformationsformel beschreibt h den Wert des hellsten und d der Wert des dunkelsten Pixels des Quellbildes. Die Variable w steht für den Wert des jeweils zu Bildpunktes vor, das Ergebnis w' ist der Wert des Pixels nach der Transformation.

$$w' = \frac{(w - d) \cdot 255}{h - d}$$

Alle Aufnahmen des Verkehrszeichens liefern nach diesen Umwandlungsschritten, unabhängig von Lichtverhältnissen, annähernd das gleiche Ergebnis. Bildstörungen werden stark reduziert.

Abbildung 5.6: Input für das neuronale Netz

Das so erstellte Bild ist der Input für das das neuronale Netz, in diesem

Fall eine Self-Organizing-Map nach Tuevo Kohonen.

5.2 Klassifizierung der Verkehrszeichen mittels neuronalem Netz

In diesem Projekt werden drei verschiedene Verkehrszeichen verwendet, zwei Geschwindigkeitsbeschränkungen (30 und 50) und ein Stopp-Schild. Da das gleiche Verkehrszeichen unter verschiedenen Bedingungen (z.B. wechselnde Lichtverhältnisse, unterschiedliche Kameraposition) verschiedene Bilder liefert, gestaltet sich die Identifizierung des Verkehrszeichen schwierig. Mittels der oben beschriebenen Bildtransformationsschritte werden gleiche Zeichen zwar annähernd gleich, aber keineswegs identisch, dargestellt.

Um die aufgenommenen Bilder trotz dieser Unterschiede richtig einer der drei Kategorien von Verkehrszeichen zuzuordnen, wird eine SOFM nach Tuevo Kohonen , verwendet. Die SOFM ist eine Methode des unüberwachten Lernens. Das Netz wird erst mehrmals mit einer erheblichen Anzahl unterschiedlicher Aufnahmen der Verkehrszeichen trainiert. Mit der Zeit bekommt das System ein „Gefühl“ für die Gleichheiten und die Unterschiede der einzelnen Bilder und ist in der Lage, die wesentlichen Charakteristika der drei verschiedenartigen Verkehrszeichen zu erkennen. Ähnliche Inputvektoren (Bilder von Verkehrszeichen) werden auf die selbe Region der SOFM abgebildet. Die Bilder werden in maximal so viele Kategorien eingeteilt, wie das Kohonen-Netzwerk Neuronen besitzt. Da in diesem Fall drei verschiedene Verkehrszeichen existieren, wird ein Netzwerk mit genau drei Neuronen erstellt.

Nachdem das Netz ausreichend trainiert wurde, kann es im Produktivbetrieb eingesetzt werden. Ab diesem Zeitpunkt ändern sich die Eigenschaften des Netzes nicht mehr, das Lernen ist also abgeschlossen. Wird nun ein neues Bild dem „ausgelernten“ Netz übergeben, so ermittelt dieses, in welche der im Lernprozess identifizierten drei Kategorien das Bild am besten passt. In der Software des Roboters ist hinterlegt, welche Aktion bei welcher Kategorie von Verkehrszeichen auszuführen ist.

Die Implementierung erfolgte mit Hilfe der *Neural Network Toolbox* in MATLAB®. Die Dimension der SOFM ist [1 3], also eine Matrix mit einer Spalte und drei Zeilen (drei Neuronen). Nach dem Lernprozess wird also jede Art von Verkehrszeichen auf ein bestimmtes der Neuronen - dem jeweiligen Winner-Neuron abgebildet. Die Distanz zwischen zwei benachbarten Neuronen ist mit 1 festgelegt. Der Inputvektor hat die Dimension 400. Das ergibt sich aus der Größe des Bildes von 20×20 Pixel.

Kapitel 6

Schlussfolgerungen

Die Überlegungen zu Beginn dieser Arbeit haben gezeigt, dass das Problem des Navigierens entlang einer Linie unter Verwendung der Fuzzy-Logic durch nur wenige einfache Regeln in natürlicher Sprache (IF-THEN-Rules) beschreiben.

Bei der Identifikation der Verkehrszeichen wurde nach einer Möglichkeit gesucht, Verkehrszeichen zu unterscheiden, ohne deren Aussehen definieren zu müssen. Das wurde durch Verwendung der SOFM nach Tuevo Kohonen erreicht. Durch erlernen vieler unterschiedlicher Fotos von Verkehrszeichen erkennt das System selbständig Unterschiede und Gemeinsamkeiten der einzelnen Bilder. Immer, wenn es darum geht visuelle Daten zu kategorisieren, sollte ein Kohonen-Netzwerk als Lösungsmethode in Betracht gezogen werden.

Durch die KI-Methoden FL und SOFM war es möglich, mit vergleichsweise geringem Modellierungsaufwand ein System erschaffen, das unter Laborbedingungen das gewünschte Ergebnis liefert. Anfängliche Schwierigkeiten durch wechselnde Lichtverhältnisse und qualitativ minderwertige Hardware konnten durch Verwendung des HSB-Farbmodells und entsprechendes Tuning des Fuzzy-Systems beseitigt werden.

Denkbare Erweiterungen für das System wären etwa Funktionen zur Interaktion mit anderen Robotern, wie etwa Abstand halten, Überholen, oder das befolgen von Vorrangregeln an Kreuzungen. Weiters könnte die Navi-

gation, die derzeit rein auf FL basiert, durch Kombination mit adaptiven Systemen, wie etwa neuronalen Netzen oder genetischen Algorithmen, weiter verbessert werden. Die Basis für die Navigation wäre dann weiterhin ein Fuzzy-System, das auf Regeln aufbaut, die dann durch Erfahrung vom Roboter selbst optimiert werden.

Bei der Suche nach Einsatzgebieten der Erkenntnisse aus dieser Arbeit sollten die beiden Bereiche *Navigation entlang einer Linie* und *Erkennen von Verkehrszeichen* gesondert betrachtet werden.

In vielen Krankenhäusern sind die Wege zwischen hochfrequentierten Abteilungen durch farbige Linien am Boden gekennzeichnet. Diese dienen dazu, den Patienten den Weg zu weisen. Nach genau diesem System könnte ein Roboter etwa Proben aus dem Labor in die entsprechende Abteilung transportieren. Vorteil der Navigation entlang einer Linie ist, dass bei Veränderung des Kurses lediglich die Linie verlegt werden muss und die Software am Roboter unverändert bleibt.

Nach entsprechender Adaptierung könnte ein ähnliches System auch genutzt werden und ein Fahrzeug auf einer Autobahn selbständig lenken zu lassen. Natürlich wäre dabei das Risiko bei einer Fehlnavigation aufgrund der hohen Geschwindigkeiten enorm. Schwierig würde sich sicher auch die Bilderkennung gestalten, besonders in der Nacht oder bei schlechten Witterungsverhältnissen. Auch schlechte oder mehrdeutige Bodenmarkierungen könnten Probleme verursachen. Als Vorstufe zur selbständigen Navigation würde sich anbiete, ein Warnsystem zu entwickeln. Dieses könnte ein akustisches Warnsignal von sich geben, wenn es bemerkt, dass das Fahrzeug von seiner Fahrspur abzukommen droht. So könnten etwa schwere Unfälle, verursacht durch Sekundenschlaf, vermieden werden.

Aufbauend auf die in dieser Arbeit beschriebene Verkehrszeichenerkennung wäre es denkbar, Geschwindigkeitswarner für KFZ zu entwickeln, indem man das System mit dem Tachometer des Fahrzeugs koppelt. Auch hier sehe ich die großen Schwierigkeiten in der Bilderkennung in der Nacht oder bei schlechten Witterungsverhältnissen. Auch das herausfiltern der wirklich relevanten Verkehrszeichen aus einem „Schilderwald“ im städtischen Bereich ist eine besondere Herausforderung.

Anhang A

Source Code

In diesem Kapitel sind die wesentlichen Teile des Source-Codes zu finden. Die Routinen habe ich in M®, der in MATLAB® integrierten Programmiersprache, geschrieben. Neben den Standardfunktionen von MATLAB® wurde auch auf Funktionalitäten folgender MATLAB®-Toolboxes zurückgegriffen:

- MathWorks Fuzzy-Logic Toolbox
- MathWorks Neural Network Toolbox
- MathWorks Image Processing Toolbox

A.1 robot.m

robot.m ist das Hauptprogramm der gesamten Verarbeitung. Ein Bild wird von einer Kamera eingelesen und analysiert. Zuerst wird eine Linie gesucht. Die Lagekoordinaten der Linie werden an ein Fuzzy-System weitergegeben, welches die Werte für die Ansteuerung der Motoren des Roboters ermittelt. Zusätzlich wird das Bild nach einem Verkehrszeichen durchsucht. Wird eines gefunden, wird es transformiert und an eine SOFM übergeben um es zu identifizieren. Je nach Verkehrszeichen werden die zuvor vom Fuzzy-System ermittelten mit einem Faktor zwischen 0 und 1 angepasst. Die so ermittelten Motorgeschwindigkeiten werden mittels DCP an dem Mikrocontroller am Roboter übergeben.

Die in diesem Programm aufgerufenen Routinen getlinepos.m und findsign.m werden in den nächsten Kapiteln beschrieben.

Listing A.1: robot.m

```
%%%%%%%%%%%%%%%%%%%%%%%%
% Autor: Roland Stelzer %
% Datum: 03/2004        %
%%%%%%%%%%%%%%%%%%%%%%%%



% Beschreibung:
%   Ein Bild wird von einer Kamera eingelesen und analysiert.
%   Zuerst wird eine Linie gesucht. Die Lagekoordinaten der Linie
%   werden an ein Fuzzy-System weitergegeben, welches die Werte
%   fuer die Ansteuerung der Motoren des Roboters ermittelt.
%   Zusaetzlich wird das Bild nach einem Verkehrszeichen
%   durchsucht. Wird eines gefunden, wird es transformiert und an
%   eine SOFM übergeben um es zu identifizieren. Je nach
%   Verkehrszeichen werden die zuvor vom Fuzzy-System ermittelten
%   mit einem Faktor zwischen 0 und 1 angepasst. Die so
%   ermittelten Motorgeschwindigkeiten werden mittels DCP an dem
%   Mikrocontroller am Roboter übergeben.

squaresize=20; % Seitenlaenge des normierten Verkehrszeichens
somvect=[];    % Input-Vector üfr SOFM
Fuzzy = readfis('Steering.fis'); % Fuzzy-System initialisieren
speed=50;      % Motorgeschwindigkeit von 0-255
linesize=2; signsize=1;

% Serielle Schnittstelle üfr
% Kommunikation zum Mikrocontroller oeffnen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
s=serial('COM1'); s.Terminator = '0'; s.BaudRate = 9600;
fopen(s);

for i=1:100 % "Endlos"-Schleife

    % Bild im RGB-Format von der Kamera einlesen
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
rgb=vfm('grab',1);

% Groesse des Bildes ermitteln
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[imgh,imgw,Z]=size(rgb);

% Bild in das HSB-Farbmodell umwandeln
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hsv=rgb2hsv(rgb);

% Lage der Linie ermitteln (obere und untere x-Koordinate)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
opos=getlinepos(hsv,1,linesize,50,135,50,255,0,255);
upos=getlinepos(hsv,imgh-linesize,linesize,
                    50,135,50,255,0,255);
line=[(opos(1)+opos(2))/2,(upos(1)+upos(2))/2];

% Winkel und Offset berechnen
%%%%%%%%%%%%%%%%%%%%%%%%%%%
dx=line(2)-line(1);
angle=atan(dx/imgh);
angle=-1*angle*360/(2*pi);
offset=(line(2)-imgw/2)/imgw*200;

% falls Linie gefunden, Verkehrszeichen suchen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (line(1)~=-1) && (line(2)~=-1)

    % Bild so drehen, dass Linie vertikal ist
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    hsv=imrotate(hsv,angle,'nearest','crop');

    % Position des Verkehrszeichens ermitteln,
    % Übergabe der HS(B)-Schwellwerte
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    pos=findsign(hsv,signsize,235,15,30,255,0,255);

    % Verkehrszeichen ausschneiden
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        hsv=hsv(round(pos(2)):round(pos(4)),
                round(pos(1)):round(pos(3)),:);

        % Verkehrszeichen der Groesse nach normieren
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        hsv=imresize(hsv,[squaresize squaresize],'nearest');

        % Umwandeln auf grayscale
        %%%%%%%%%%%%%%%%%%%%%%%
        rgb=hsv2rgb(hsv);
        gr=rgb2gray(rgb);

        % Kontrast maximieren
        %%%%%%%%%%%%%%%%%%%
        maximum=max(max(gr));
        minimum=min(min(gr));
        deltagr=(maximum-minimum);
        gr=(gr-minimum)*(1/deltagr);

        % Input-Vektor üfr die SOFM setzen
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        singlesomvect=[];
        for i=1:squaresize
            singlesomvect=[singlesomvect gr(i,:)];
        end
        somvect=[somvect;singlesomvect];
    end

    % Richtung durch das Fuzzy-System ermitteln,
    % anhand von Winkel und Offset
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    steer=evalfis([angle offset],Fuzzy)

    % Geschwindigkeit der einzelnen Motoren ermitteln,
    % auf Ganzzahl gerundet und auf 8 Bit skaliert
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (steer<0)
        R=255;
        L=255+steer;
```

```
    else
        L=255;
        R=255-steer;
    end
    R=R*speed/255;
    L=L*speed/255;

    % Motorsteuerungsdaten an Mikrocontroller schicken,
    % DCP-Format
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fwrite(s,65,'int8');     %left engine
    fwrite(s,L,'int8');      %speed on left engine
    fwrite(s,69,'int8');     %right engine
    fwrite(s,R,'int8');      %speed on right engine
    fwrite(s,0,'int8');      %terminator
    pause(0.1);
    fwrite(s,65,'int8');     %left engine
    fwrite(s,0,'int8');      %speed on left engine
    fwrite(s,69,'int8');     %right engine
    fwrite(s,0,'int8');      %speed on right engine
    fwrite(s,0,'int8');      %terminator
    pause(0.1);

end

% Serielle Schnittstelle schliessen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fclose(s); delete(s); clear s
```

A.2 getlinepos.m

getlinepos.m durchsucht das gesamte Bild zeilenweise mittels „Sliding Window“ nach einem Farbbereich, der innerhalb der übergebenen Schwellwerte liegt.

Listing A.2: getlinepos.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Autor: Roland Stelzer %
% Datum: 03/2004        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function xpos=getlinepos(image, y ,width, hu, ho, su, so, vu, vo)

% Beschreibung:
%   Durchsucht eine Bildzeile mittels "Sliding Window" nach
%   einem Farbbereich, der innerhalb der uebergebenen
%   Schwellwerte liegt.
%
% Parameter:
%   image ... Bild im HSB-Format
%   y     ... Zeile die durchsucht werden soll
%   width ... Groesse des "Sliding Window" (width x width)
%   hu,ho ... Unterer und oberer Schwellwert fuer den Farbwert
%   su,so ... Unterer und oberer Schwellwert fuer die Saettigung
%   vu,vo ... Unterer und oberer Schwellwert fuer die Helligkeit
%
% Rueckgabewerte:
%   Koordinaten des ersten und des letzten passenden Pixel in der
%   durchsuchten Bildzeile. Wenn keiner gefunden wird: [-1,-1].

[imgh,imgw,Z]=size(image);     %getting size of image
xhelp=-1; xfirst=-1;

% von links nach rechts
%%%%%%%%%%%%%%%%%%%%%%%%%%%
for x=1:(imgw-width)
    counter=0;
    rc=0;
```

```
    gc=0;
    bc=0;
    for tx=0:(width-1)
        for ty=0:(width-1)
            help=double(image(y+tx,x+ty,1));
            rc=rc + help;
            help=double(image(y+tx,x+ty,2));
            gc=gc + help;
            help=double(image(y+tx,x+ty,3));
            bc=bc + help;
            counter=counter + 1;
        end
    end
    avgr=rc/counter;
    avgg=gc/counter;
    avgb=bc/counter;

    % Schwellwerte ueberpruefen
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (((ho<hu) && ((avgr>hu/255) || (avgr<ho/255))) ||
        ((ho>=hu) && (avgr>hu/255) && (avgr<ho/255))) &&
        (avgg<=so/255) && (avgb<=vo/255) && (avgg>=su/255) &&
        (avgb>=vu/255)
        if xhelp==-1
            xfirst=x;
        end
        xhelp=x;
    end
end if(xhelp~=-1)
    xfirst=xfirst+width/2;
    xhelp=xhelp+width/2;
end

% Rueckgabewert setzen
%%%%%%%%%%%%%%%%%%%%%%
xpos=[xfirst,xhelp];
```

A.3 findsign.m

findsign.m Durchsucht das gesamte Bild zeilenweise mittels „Sliding Window“ nach einem Farbbereich, der innerhalb der übergebenen Schwellwertes liegt. Im wesentlichen wird das Bild Zeile für Zeile unter Verwendung der Funktion *getlinepos.m* durchlaufen. Als Ergebnis liefert die Funktion die Koordinaten des Verkehrszeichens, falls sich eines im Bild befindet.

Listing A.3: findsign.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%
% Autor: Roland Stelzer %
% Datum: 03/2004        %
%%%%%%%%%%%%%%%%%%%%%%%%%

function pos=getlinepos(image ,width, hu, ho, su, so, vu, vo)

% Beschreibung:
%   Durchsucht das gesamte Bild zeilenweise mittels "Sliding
%   Window" nach einem Farbbereich, der innerhalb der
%   uebergebenen Schwellwerte liegt.
%
% Parameter:
%   image ... Bild im HSB-Format
%   width ... Groesse des "Sliding Window" (width x width)
%   hu,ho ... Unterer und oberer Schwellwert fuer den Farbwert
%   su,so ... Unterer und oberer Schwellwert fuer die Saettigung
%   vu,vo ... Unterer und oberer Schwellwert fuer die Helligkeit
%
% Rueckgabewerte:
%   Koordinaten des obersten, untersten, aeusserst linken und
%   des aeusserst rechten passenden Pixel in der durchsuchten
%   Bildzeile. Wenn keiner gefunden wird: [-1,-1,-1,-1].

[imgh,imgw,Z]=size(image);     %getting size of image
x1=-1; x2=-1; y1=-1; y2=-1;

% von oben nach unten (zeilenweise)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for y=1:(imgh-width)
```

```
    % ermittlen des aeusserst linken und aeusserst rechten
    % passenden Pixel der aktuellen Zeile
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    linepos=getlinepos(image,y,width,hu, ho, su, so, vu, vo);

    if (y1==-1) && (linepos(2)~=-1)
        y1=y;
    end
    if (y1~=-1) && (linepos(2)==-1)
        y2=y;
        break;
    end
    if(y1~=-1)
        if(x1==-1)
            x1=linepos(1);
            x2=linepos(2);
        else
            x1=min(x1,linepos(1));
            x2=max(x2,linepos(2));
        end
    end
end

% Rueckgabe der Verkehrszeichen-Koordinaten
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pos=[x1,y1,x2,y2];
```

Literaturverzeichnis

[1] Berger W., Grosse S., Lugauer U. (2004). *Beschreibung eines Kohonen Netzwerkes Zur Verwendung von Muster Erkennung (Kohonen-Feature-Map)*. FH Regensburg. Internet http://rfhs8012.fh-regensburg.de/šaj39122/begrolu/nn1.html

[2] Callan R. (2003). *Neuronale Netze im Klartext*. Verlag Pearson Studium, ISBN 382737071X.

[3] Cawsey A. (2003). *Künstliche Intelligenz im Klartext*. Verlag Pearson Studium, ISBN 382737068X.

[4] Cao J., Hall E., Liao X. (1999). *Reactive Navigation for Autonomous Guided Vehicle Using the Neuro-fuzzy Techniques*. Center for Robotics Research, ML 72, University of Cincinnati. Internet http://robotics.uc.edu/pdfdocs/spiejin/jin99.pdf

[5] Chen R. L. (1993). *A Fuzzy Inference Design on Hewlett-Packard Logic Synthesis System*. HP Laboratories Palo Alto External Research Program. Internet http://www.hpl.hp.com/techreports/93/HPL9370.pdf

[6] Dabrowski A. (2002). *The Turtle Principle*. Internet http://www.atrox.at/robots/inetturtle/index.html

[7] EasyRGB (2004). *Color Conversion Formulas*. Internet http://www.easyrgb.com/math.php?MATH=M20#text20

[8] Fellner W. (1991). *Computergrafik*. BI-Wissenschaftsverlag, Mannheim/Leipzig/Wien/Zürich, 2. Edition

[9] Fischli S. (2004). *Neuronale Netze: Backpropagation und Kohonen.* Berner Fachhochschule. Internet http://www.hta-be.bfh.ch/ fischli/kurse/eduswiss/nnet/skript.pdf

[10] Halter B. (2003). *Autonomous Robot.* Final Year Project, University of Derby.

[11] Honkela T. (1998). *Description of Kohonen's Self-Organizing Map.* Internet http://www.mlab.uiah.fi/ timo/som/thesis-som.html

[12] Iovine J. (2000), *PIC Microcontroller Project Book.* McGraw-Hill Publishing Co., ISBN 0071354794.

[13] Kaehler S. D. (2003). *Fuzzy Logic – An Introduction.* Seatlle Robotics Society. Internet http://www.seattlerobotics.org/encoder/mar98/fuz/fl_part1.html

[14] Kohonen T. *Self-organized formation of topologically correct feature maps* Biological Cybernetics, Vol. 43 (1982), S. 59 - 69

[15] Kohonen T. (1995), *Self-Organizing Maps.* Springer Series in Information Sciences, Vol. 30, Springer, Berlin, Heidelberg, New York.

[16] Krantz B. (2002). *A Crisp Introduction to Fuzzy Logic.* Computer Science Education Lab, University of Colorado At Boulder. Internet http://www-ugrad.cs.colorado.edu/c̃s3202/papers/Brigette_Krantz.html

[17] Müller L. (2003). *Colour Detection.* Assignment report for the module Ärtificial Neural Nets and Expert Systems", University of Derby.

[18] Ng K. C., Trivedi M. M. (1998). *A Neuro-Fuzzy Controller for Mobile Robot Navigation and Multirobot Convoying*, IEEE. Internet http://cvrr.ucsd.edu/aton/publications/pdfpapers/Nif-T_98.pdf

[19] Pawlikowski S. (1999). *Development of a Fuzzy Logic Speed and Steering Control System For an Autonomous Vehicle.* University of Cincinnati, Master of Science Research

Project, Department of Mechanical Engineering. Internet http://robotics.uc.edu/theses/theses1998/pawlikowski.pdf

[20] *Real Robots Magazin, Ausgabe 3* (2003), Eaglemoss International Ltd 2001.

[21] Sprut (2004). *Die Nutzung des 10-Bit PWM.* Internet http://www.sprut.de/electronic/pic/grund/pwm.htm

[22] Stelzer R. (2003)(a). *Digital Eye for Neuro-Fuzzy Robot Navigation.* Assignment report for the module „Artificial Neural Nets and Expert Systems", University of Derby.

[23] Stelzer R. (2003)(b). *FuzzyWeb: A Web-Based Approach to Fuzzy System Design.* Final Year Project, University of Derby. Internet http://fuzzyweb.atrox.at/FuzzyWeb.pdf

[24] The Mathworks (1995). *Fuzzy Logic Toolbox - For Use with MATLAB.* The Mathworks. Internet http://www.mathworks.com/access/helpdesk/help/pdf_doc/fuzzy/fuzzy_tb.pdf

[25] The Mathworks (2003). *The MathWorks Product Family.* The Mathworks Inc. http://www.mathworks.com/products/prodoverview.shtml (2003)

[26] TU Dresden (2004). *Farbmodelle / Farbtiefe.* Internet http://www.tu-dresden.de/urz/bilddtp/Theorie/body_theorie.html

[27] Tzafestas S. G., Zikidis K. C. (1997). *A 3-Level Neuro-Fuzzy Autonomous Robot Navigation System*, Intelligent Robotics and Automation Laboratory, National Technical University of Athens, Greece. Internet http://www.cds.caltech.edu/conferences/related/ECC97/proceeds/751_1000/ECC754.PDF

Abkürzungsverzeichnis

CoG	Center of Gravity (Schwerpunkt)
CPU	Central Processing Unit
DCP	Device Communication Protocol
DID	Device Identifier (im DCP-Header)
FIS	Fuzzy Inferenz System
FL	Fuzzy Logic
FLS	Fuzzy Logic System
HSB	Hue, Saturation, Brightness (Farbton, Sättigung, Helligkeit)
HSV	Hue, Saturation, Value (Farbton, Sättigung, Wert)
KFZ	Kraftfahrzeug
KI	Künstliche Intelligenz
LoD	Length of Data (im DCP-Header)
PWM	Pulsweitenmodulation
RGB	rot, grün, blau (Primärfarben des additiven Farbmodells
SOFM	Self-Organizing-Feature-Map (Kohonen-Netzwerk)
t-Norm	Triangular Norm
USB	Universal Serial Bus

Index