1. Inhaltsverzeichnis

1. Inhaltsverzeichnis 1

2. Einleitung 4

2.1. Allgemeines 4
2.2. Aufgabenstellung 4
2.2.1. Realisierung der Aufgabenstellung 5
2.2.2. Vorteile und Konzepte von MS Windows 3.0 6
2.2.3. Die Programmiersprache 'C' 6

3. Das Bildverarbeitungssystem 8

3.1. Die Funktionseinheit 03 8
3.2. Die Verbindung zum PC 8

4. Neuronale Netze 11

4.1. Einleitung 11
4.2. Modellbildung eines Neurons 12
4.3. Netzformen 14
4.4. Funktionsweise eines neuronalen Netzes 15
4.5. Backpropagation of Errors 16
4.5.1. Bei zwei Layern 16
4.5.2. Bei n Layern 20
4.5.2.1. Herleitung 21
4.5.2.2. Zusammenfassung 24

5. Bedienungsanleitung 25

5.1 Installation von NeuroRob 25
5.2. Starten des Programms NeuroRob 25
5.3. Menuepunkte von NeuroRob 27
5.3.1. Info 28
5.3.2. RoboterInit 28
5.3.3. RoboterSteuerung 29
5.3.4. RoboterBefehlsliste 30
5.3.5. RoboterSende_File 31
5.3.6. RoboterPositionsanalyse 32
5.3.7. RoboterVisualisierung 33
5.3.8. RoboterBeenden 33
5.3.9. VideomatInit/Abgleich 34
5.3.10. VideomatSteuerung 34
5.3.11. DemoGelenkdemo 35

5.3.12. BrainInit 35
5.3.13. BrainInit_DateiNeu 35
5.3.14. BrainInit_DateiLaden 38
5.3.15. BrainInit_DateiSpeichern 39
5.3.16. BrainInit_DateiSpeichernals 39
5.3.17. BrainInit_Brainbeenden 39
5.3.18. BrainInit_TrainingHohePriorität 39
5.3.19. BrainInit_TrainingMittlerePriorität 39
5.3.20. BrainInit_TrainingNiedrigePriorität 39
5.3.21. BrainInit_TrainingStart 39
5.3.22. BrainInit_TrainingAnzeigen 40
5.3.23. BrainInit_TrainingReset 40
5.3.24. BrainInit_ErkennungStart 40
5.3.25. BrainInit_ErkennungTestphase 41
5.3.26. BrainInit_ErkennungRobotereinsatz 42
5.3.27. BrainInit_OptionEdit_Teiledaten 42
5.3.28. BrainInit_OptionAnzeige 42
5.4. Bedienungsbeispiel 42

6. Zusammenfassung 48

6.1 Ergebnis und Beurteilung der Arbeit 48
6.2 Anwendungsmöglichkeiten 49
6.3. Anregungen für weitere Untersuchungen 49

7. Programmlisting 51

7.1. Das Projekt-File NEUROROB.PRJ 51
7.2. Die Moduldefinitionsdatei NEUROROB.DEF 52
7.3. Die Header-Dateien 53
7.3.1. NEUROROB.H 53
7.3.2. NEUROMEN.H 54
7.3.3. N_EXTERN.H 54
7.3.4. BRAINMEN.H 56
7.3.5. NDL_ABO.H 56
7.3.6. NDL_EDIT.H 56
7.3.7. NDL_FILE.H 56
7.3.8. NDL_LIST.H 57
7.3.9. NDL_NDAT.H 57
7.3.10. NDL_NNET.H 57
7.3.11. NDL_POS.H 58
7.3.12. NDL_STEU.H 58
7.3.13. NDL_TDAT.H 59
7.3.14. NDL_ROB.H 59
7.3.15. NDL_VID.H 59
7.3.16. NDL_VISU.H 59
7.4. Der C-Quellcode 60

7.4.1. NEUROROB.C 60
7.4.2. N_ABOUT.C 70
7.4.3. N_BRAIN.C 72
7.4.4. N_FILE.C 108
7.4.5. N_LIST.C 110
7.4.6. N_MASTER.C 116
7.4.7. N_POSITI.C 119
7.4.8. N_ROBINI.C 125
7.4.9. N_STEU.C 132
7.4.10. N_VIDIOI.C 137
7.4.11. N_VISUAL.C 141
7.5. Der Resourcequellcode NEUROROB.RC 145
7.6. Das Sivips-Programm NEUROROB.PRS 153

8. Literaturverzeichnis 158

2. Einleitung

2.1. Allgemeines

Die Verarbeitung von visuellen Informationen ist ein wichtiges Merkmal höherer Lebensformen. So ist es nicht verwunderlich, daß zunehmend auch Datenverarbeitungssysteme in diesem Bereich eingesetzt werden. Die Hauptforschungsgebiete bei den Computersichtsystemen sind dabei die Bildverarbeitung, die Mustererkennung und das Bildverständnis.

Die Künstliche Intelligenz (KI) beschäftigt sich mit der maschinellen Repräsentation und Verarbeitung menschlicher Wissensstrukturen als Grundlage für komplexe Informationsverarbeitungsprozesse. Mit Hilfe von Verfahren der Künstlichen Intelligenz können die für anspruchsvolle informationstechnische Anwendungen relevanten Aspekte intelligenten Verhaltens maschinell verfügbar gemacht werden. Die bekanntesten Vertreter solcher Systeme sind künstliche neuronale Netzwerke.

Neuronale Netzwerke erleben derzeit einen unbeschreiblichen großen Aufschwung. Die praktischen Anwendungen reichen heute bereits von der Bilderkennung und -interpretation über die Lösung kinematischer und dynamischer Probleme in der Robotik bis hin zur Prognose von Aktienkursen. **Marktforscher sagen der Technik bis weit in die 90er Jahre jährlich zweistellige Wachstumsraten voraus.**

Einer der Gründe für diesen Aufschwung ist, daß neuronale Netzwerke in ihrem Aufbau und ihrer Konzeption stärker an der Funktionsweise des menschlichen Gehirns orientiert sind als an der Arbeitsweise konventioneller Rechner der klassischen von Neumann-Architektur. Neuronale Netze können deshalb leichter dazu genutzt werden, wichtige geistige Fähigkeiten des Menschen wie das Lernen aus Beispielen, das Verallgemeinern von Beispielen, das Abstrahieren, das schnelle Erkennen und Vervollständigen komplizierter Muster, das assoziative Speichern und Abrufen von Informationen etc. nachzubilden und zu simulieren.

2.2. Aufgabenstellung

Die Diplomarbeit von Heinz Ennen und Matthias Rhein [3] beschäftigte sich mit der Steuerung eines Gelenkarmroboters unter Einsatz eines Bilderfassung- und Auswertesystems der Firma Siemens. Nach der Ergänzung des Bildauswertesystems zur Verarbeitung von Graubildern bot sich uns eine Folgediplomarbeit an.

Das Thema wurde wie folgt formuliert:

> Programmierung einer Windows-Applikation zur Steuerung eines Gelenkarmroboters mit Hilfe einer grauwerterfassenden Bildverarbeitungsanlage unter Anwendung neuronaler Netze.

Der Roboter soll mit Hilfe der Bilderfassungsanlage in der Lage sein, vorher 'eingelernte' Gegenstände anhand ihrer Grauwerte zu erkennen, ihre Position und Orientierung festzustellen, die Gegenstände zu greifen und anschließend definiert abzulegen. Unsere Schwerpunkte lagen vor allem in der Programmierung einer Microsoft Windows-Applikation unter der Programmiersprache 'C' und die Auswertung eines Grauwertbildes durch ein neuronales Netz.

2.2.1. Realisierung der Aufgabenstellung

Da unsere Diplomarbeit auf der bereits erwähnten Arbeit [3] aufbaut, wurden fast die gleichen Hardwarekomponenten benutzt.

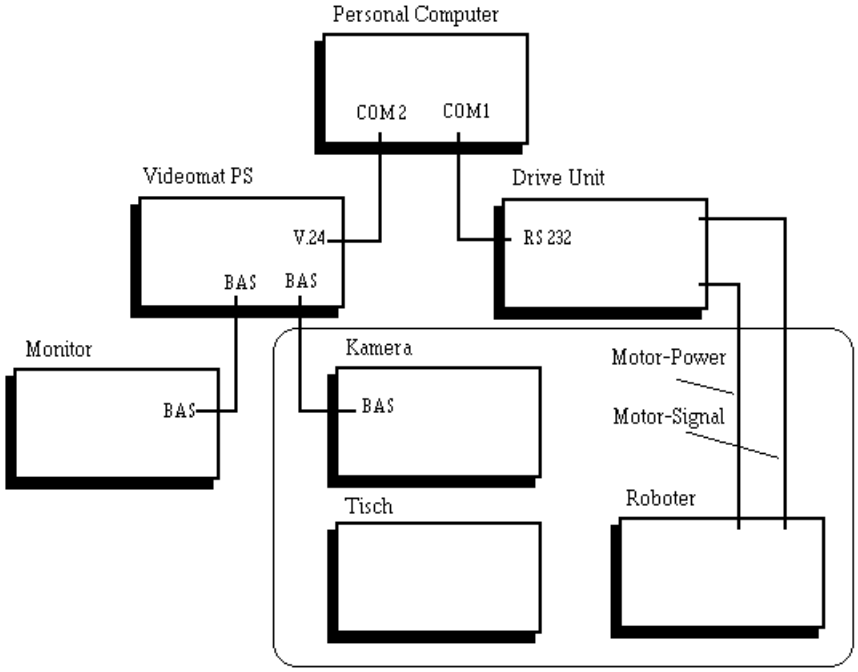

Bild 1
Systemkonfiguration

Wie aus der Abbildung (Bild 1) ersichtlich ist, sind sowohl das Bildauswertesystem 'VIDEOMAT PS' als auch der Roboter ('Drive Unit') über einen PC mit 80486 CPU gekoppelt. Die Bildverarbeitungsanlage der Firma Siemens wurde durch eine Funktionseinheit (F03) zur Grauwerterverarbeitung ergänzt. Außerdem wurde das Greifsystem des Roboters durch uns erweitert.

Technische Details des Roboters und der Bilderfassungsanlage können in der oben genannten Diplomarbeit [3] nachgelesen werden. Hier findet man auch eingehende Informationen zur Installation und Programmierung.

Unsere Hauptaufgabe war es nun, die Theorie über neuronale Netze, welche wir zur Klassifizierung von Objekten einsetzen, in ein 'C'-Programm umzusetzen und weiterhin die Kommunikation zwischen den einzelnen Komponenten untereinander zu gewährleisten.

2.2.2. Vorteile und Konzepte von MS Windows 3.0

Das Programm 'NeuroRob' wurde für die Verwendung unter der graphischen Benutzeroberfläche WINDOWS entwickelt. Diese graphische Oberfläche der Firma Microsoft bietet eine quasi-standardisierte Bedienung der Programme, so daß der Aufwand für die Erlernung neuer Programme auf ein Minimum reduziert werden kann. Ist die grundlegende Funktionsweise von WINDOWS einmal verstanden, müssen nur die programmspezifischen Funktionen neu erlernt werden. Aus programmtechnischer Sicht ergibt sich durch die Verwendung von WINDOWS der Vorzug der geräteunabhängigen Programmierbarkeit; das Programm ist auf einer Vielzahl von PC-Rechnern unterschiedlichster Konfiguration lauffähig. Insbesondere die unterschiedlichen Grafik-Standards (Hercules, EGA, VGA usw.) müssen bei der Programmierung nicht weiter berücksichtigt werden. Voraussetzung ist, daß für die gewünschte Grafikkarte ein passender Treiber zur eingesetzten WINDOWS-Version verfügbar ist. Ein weiterer Vorteil liegt in der Multitaskingfähigkeit von WINDOWS. Hiermit ist es möglich, komplexe Rechenvorgänge im Hintergrund zu bearbeiten (Ein wichtiges Argument für den Einsatz von Windows in Anbetracht des großen Rechenaufwandes beim Training von neuronalen Netzen). Somit bleibt der PC dem Anwender für weitere Aufgaben verfügbar. Weiterhin bietet Microsoft im Lieferumfang zu WINDOWS leistungsstarke Programme, die eine effiziente Arbeit am Rechner ermöglichen. Durch die schon angesprochene Multitaskingfähigkeit ist eine Kommunikation der einzelnen WINDOWS-Applikationen untereinander möglich. So bot sich für uns die Möglichkeit, den 'Notizblock' zur Erstellung von Roboterprogrammen zu nutzen.

2.2.3. Die Programmiersprache 'C'

Für die Erstellung einer Windows-Applikation ist ein spezieller Compiler notwendig, der die ca. 600 Windowsfunktionen unterstützt. Im Labor für Regelungstechnik und Prozeßlenkung wurde uns der Borland 'C++'-Compiler zur Verfügung gestellt.

Da Microsoft Windows ebenfalls in 'C' programmiert wurde, ist es vorteilhaft, die Windows-Applikationen ebenfalls in dieser Sprache zu erstellen.

Windows 3.0 gestattet es nicht, 'Lowlevel-Funktionen' auf DOS-Ebene zu benutzen. Windows arbeitet ereignisorientiert, deshalb mußte eine völlig neue Programmstruktur in der Programmiersprache 'C' entwickelt werden.

3. Das Bildverarbeitungssystem

3.1. Die Funktionseinheit 3

Die Funktionserweiterung (F03) besteht aus Bildrechner und Bildspeicher (5 Baugruppen Bildspeicher mit Echtzeit ALU - Arithmetik Logik Unit - und RGB-Ausgang, 1 Baugruppe Bildrechner) [5]. Zur Verarbeitung von Graubildern enthält sie insgesamt 16 Bildspeicher und einen Bildrechner. Die maximale Auflösung der Bildspeicher beträgt 512*512 Bildpunkte mit je 64 Graustufen. Der Bildrechner dient zur schnellen Bildvorverarbeitung und kommuniziert direkt mit den Bildspeichern. Das Ergebnis der Bildvorverarbeitung kann auf einen anschließbaren RGB-Monitor in Falschfarben dargestellt werden.

Durch die Erweiterung des Videomatsystems von Siemens wurde es uns ermöglicht, einen Gegenstand durch seine Graustufenverteilung zu erkennen. Damit steigert sich der Leistungsumfang der Bilderkennung enorm, da selbst Gegenstände, die den gleichen Umriß haben, sich jedoch in der Oberflächenstruktur voneinander unterscheiden, klassifiziert werden können.

In unserer Diplomarbeit war es nun wichtig, die Gegenstände bzw. Werkstücke vorwiegend durch ihre Grauwertverteilung voneinander zu unterscheiden und danach durch Binärisierung die Position und Orientierung festzustellen.

3.2. Die Verbindung zum PC

Um eine effektive Arbeit mit dem VIDEOMAT PS zu gewährleisten, ist ein gutes Softwarekonzept nötig. Wir haben uns für eine Lösung entschieden, die auf einer 'Server - Client - Architektur' basiert. Hier ist der Rechner der Client und das Videomatsytem der Server. Der Client gibt dem Server Aufgaben, die dieser selbstständig ausführt und danach dem Client das Ergebnis übermittelt. Im folgenden seien alle 'Aufträge', die der PC dem Videomaten erteilen kann, aufgeführt:

Auftrag	**Ergebnis des Auftrags**
'A'	Binärbilder und Grauwertbilder werden in den Bildspeichern (vgl. Bild 2) abgelegt und anschließend vom Videomatsystem ausgewertet. Die ermittelten Bildmerkmale werden dem Rechner gesendet, der die für ihn wichtigen Merkmale selektiert und dann weiterverarbeitet.
'P'	Binärisierungsschwelle wird heraufgesetzt (voreingestellt ist der Wert 58).
'M'	dito, es erfolgt jedoch eine Herabsetzung.
'C'	Speicherung und Darstellung des Grauwertvollbildes in G1.
'E'	Speicherung und Darstellung des Binärvollbildes in B3.1.

'H' Speichern eines Grauwerthalbbildes in G2.1, daraus wird das Grauwerthistogramm berechnet, in B4.11 gespeichert und auf dem Sivips-Monitor abgebildet.

'O' Abspeichern des Kamerabildes in den Lauflängenspeicher und in B4.12. Daraus wird die Orientierung und der Schwerpunkt des binärisierten Teiles berechnet und dargestellt.

'D' Anzeigen des Live-Binärbildes.

'B' Anzeigen des Live-Graubildes.

'T' Speicherung der Szene in den Lauflängenspeicher. Daraufhin erfolgt eine Überprüfung des Bildes auf Bewegung und auf Anzahl der vorhandenen Teile. Bei Bewegung oder einer Anzahl von Teilen ungleich eins, teilt das VIDEOMAT PS dem PC durch Senden eines 'N' mit, daß eine Merkmalextrahierung im Runtime-Modus nicht sinnvoll ist. Genügt das Bild den gestellten Anforderungen, wird dem Rechner ein 'J' übermittelt.

'X' Dieser Auftrag ermittelt den Schwerpunkt des in der Szene liegenden Teils und teilt diesen dem PC mit.

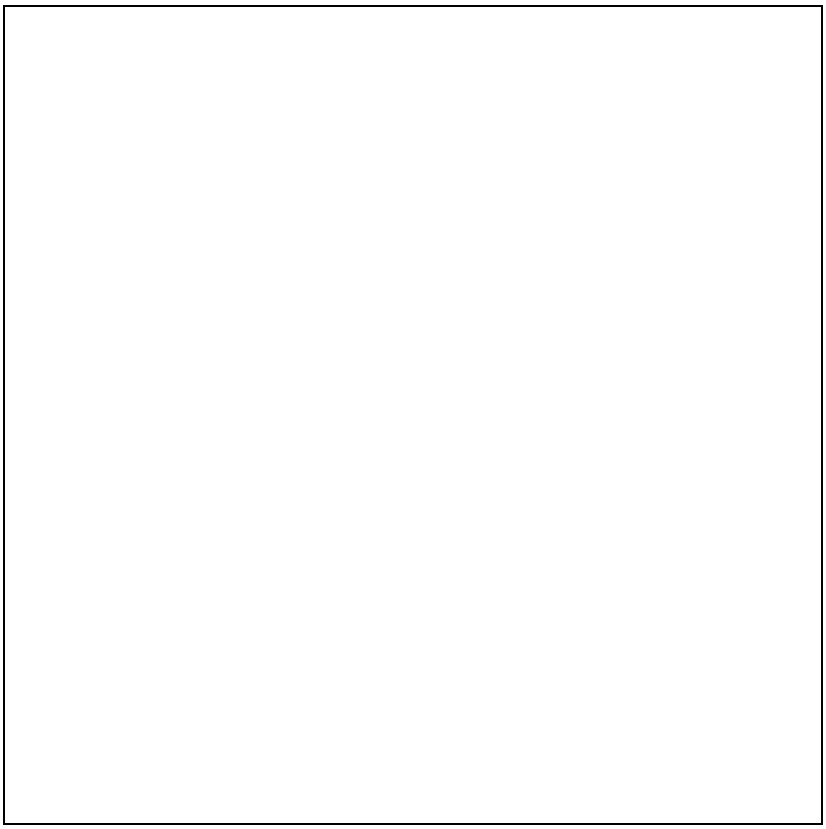

Bild 2
Aufteilung des Bildspeichers im VIDEOMAT PS

Leider war es uns nicht möglich, eine Implementation weiterer Aufträge (vor allem umfangreichere) in das Programm NEUROROB.PRS einzubinden. Die Ursache hierfür ist der interne

Programmspeicher des VIDEOMAT PS, dessen Kapazität nach Angaben der Firma Siemens nur 20 kByte beträgt. Dieser Speicherplatz wird durch die jetzige Version von NEUROROB.PRS voll ausgenutzt. Eine Erweiterung des Programmspeichers ist nach Auskunft der Firma Siemens nicht möglich.

4. Neuronale Netze

4.1. Einleitung

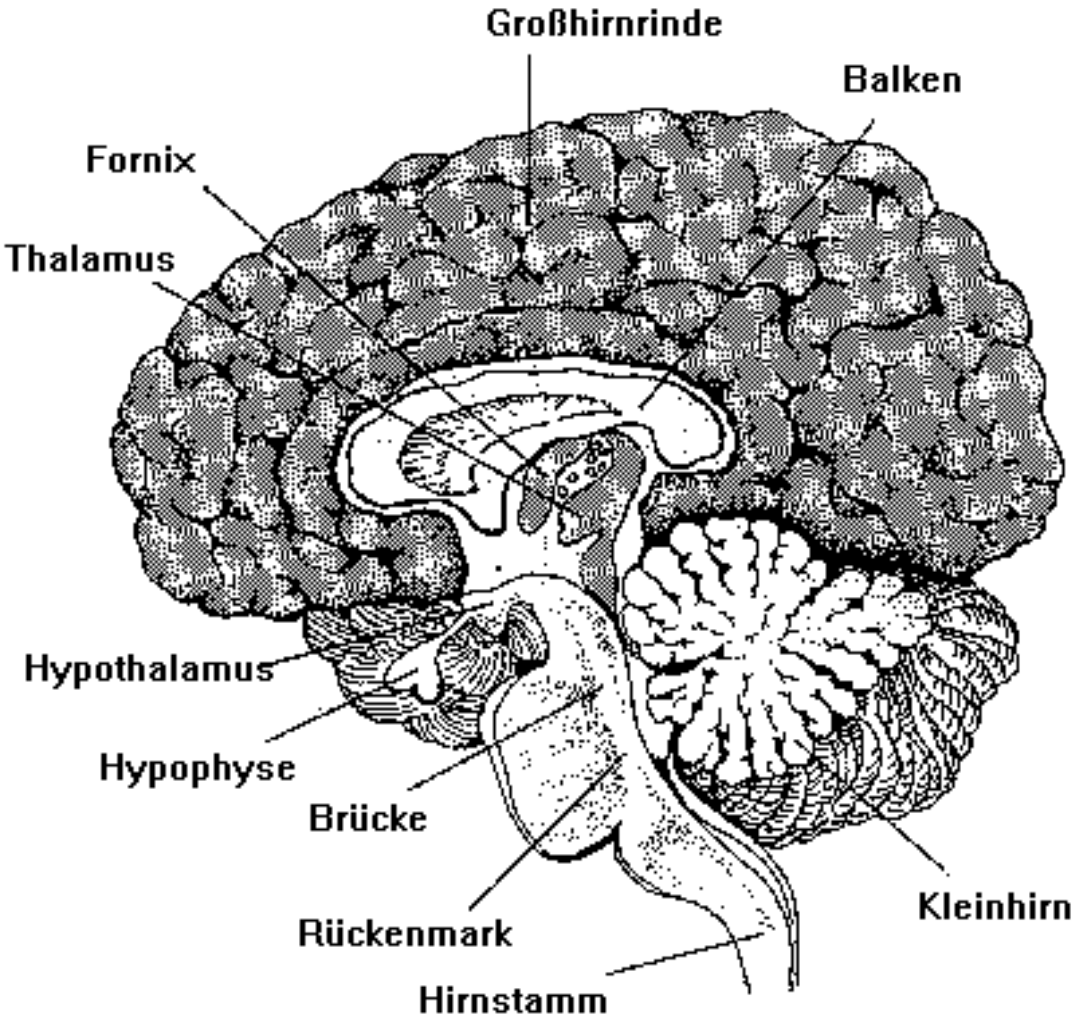

Bild 3
Schnitt durch das menschliche Gehirn [10]

Das Gehirn besteht aus Milliarden von Zellen, die man Neuronen nennt. Jede dieser Zellen ähnelt einem winzigen Computer mit sehr beschränkten Fähigkeiten - miteinander verbunden bilden diese Zellen jedoch das intelligenteste System, das wir kennen. Künstliche neuronale

Netze bestehen aus simulierten Neuronen, die untereinander ganz ähnlich wie die Zellen des Gehirns miteinander verbunden sind.

Neuronale Netze sind statisch basierte, nicht lineare Systeme, die ihr Wissen innerhalb bestimmter Grenzen selbsttätig aus den angebotenen Eingaben extrahieren. Sie sind den biologischen neuronalen Netzen nachempfunden und besitzen einige ähnliche Eigenschaften, wie Robustheit und Fehlertoleranz, so daß sie für viele technische Aufgaben prädestiniert sind. Die Entwicklung der neuronalen Netze befindet sich erst in den Anfängen. Da sie jedoch als Mustererkenner in vielen realen Anwendungen nicht zu übertreffen sind und sich zudem noch für die parallele Datenverarbeitung eignen, ist ihr Siegeszug nicht aufzuhalten.

4.2. Modellbildung eines Neurons

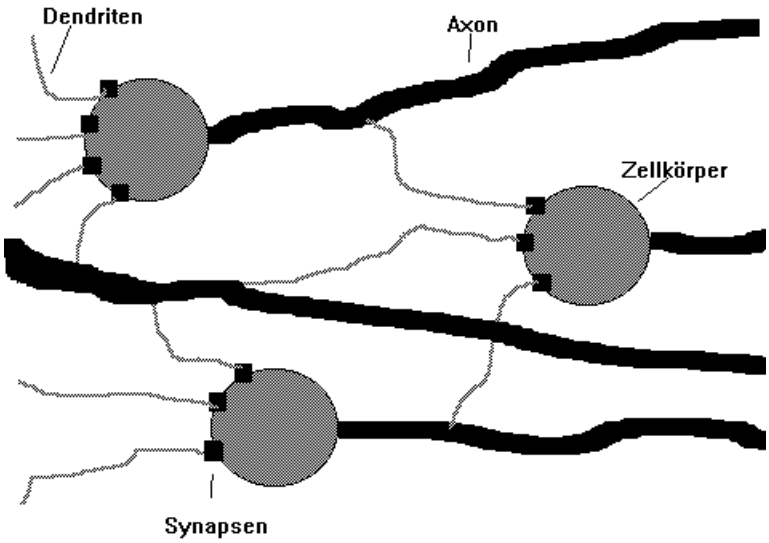

Bild 4
Abstrakte Darstellung eines biologischen neuronalen Netzes

Die Modellbildung wird auf der Basis eines extrem vereinfachten Neuronenkomplexes dargestellt (siehe Bild 4). Obwohl die für künstliche neuronale Netze verwendete Terminologie sich geringfügig von der biologischer Netze unterscheidet, funktionieren die Neuronen in einem künstlichen neuronalen Netzwerk auf ähnliche Weise. Für die Umsetzung zur Simulation neuronaler Netzwerke sind folgende Strukturelemente von Bedeutung:

- Der Zellkörper selbst dient als Informationsträger

- Das Axon dient zur Weitervermittlung des Zellzustandes.

- Die Synapse bestimmt, wie die sich über ein Axon vermittelte Erregung auf eine andere Zelle auswirken soll.

- Dendriten stellen die Verbindungen von den Axonen zu den Synapsen dar.

Ein hieraus abgeleitetes mathematisches Modell wird in Bild 5 dargestellt:

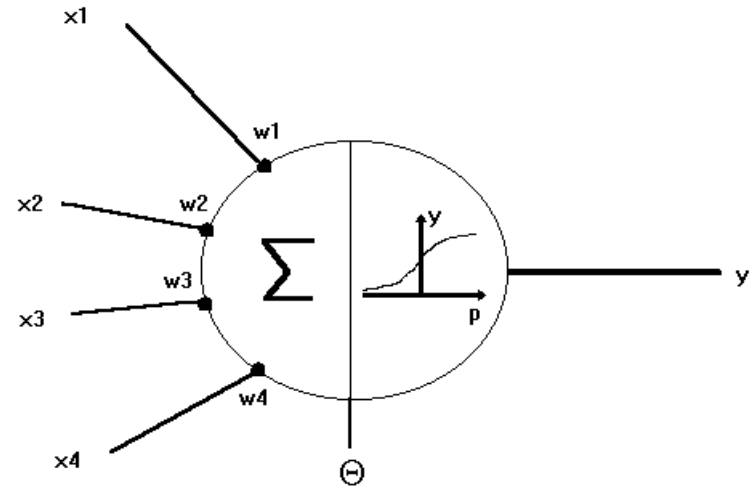

Bild 5
Schematische Darstellung eines technisch simulierbaren Neurons

Dieses mathematische Modell wird wie folgt beschrieben:

$$p = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 - \Theta$$

$$p = \sum_i (x_i w_i) - \Theta$$

Hierbei repräsentiert 'p' das innere Potential des Neurons, 'x' die jeweiligen Eingänge des Neurons (Dendriten), 'w' die zu den Eingängen gehörenden Gewichte (Synapsen) und 'Θ' ein konstanter Wert, der von der Summe subtrahiert wird. Dieses innere Potential wird durch eine Übertragungsfunktion an den Ausgang 'y' weitergegeben. Die Übertragungsfunktion hat die Aufgabe, 'p' auf ein Intervall von (0,1) abzubilden. Als Übertragungsfunktion haben wir die Sigmoid-Funktion (siehe Bild 6) gewählt:

$$y = \frac{1}{1+e^{-p}}$$ Sigmoid-Funktion

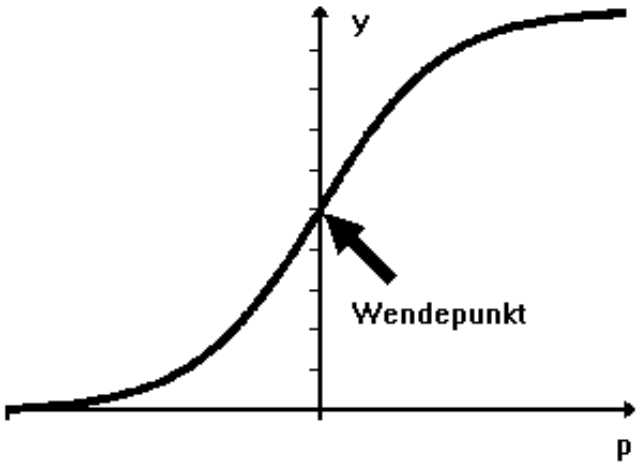

Bild 6
Die Sigmoid-Funktion

Diese Modellbildung eines Neurons kann sehr gut auf einem Computer implementiert werden.

4.3. Netzformen

Für neuronale Netze gibt es verschiedene Netzstrukturen. Jedes einzelne Netz besitzt besondere Fähigkeiten. Das von uns gesuchte Modell sollte in der Lage sein, anhand von Trainingsvektoren Objekte zu klassifizieren und später durch Anlegen eines Eingangsvektors diesen mit einer bestimmten Klasse zu assoziieren. Ein Netz, welches diesen Anforderungen genügt, ist das sogenannte 'FEEDFORWARD NETZ' (siehe Bild 7).

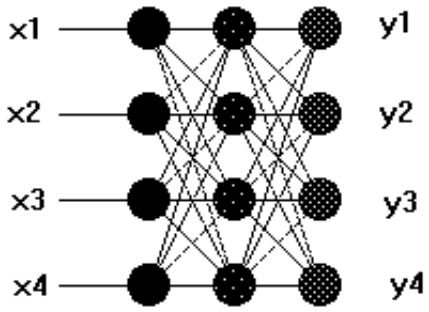

Bild 7
Feedforward-Netz mit 3 Layern und jeweils 4 Neuronen pro Layer

Das abgebildete Netz besteht aus drei Neuronenspalten. Diese Spalten, auch Layer genannt, teilen sich wie folgt auf:

- Der Eingangslayer ist kein echter Layer, da er keine Kombinatorik ermöglicht, er ist eine Art Latch - er gibt seine Eingänge direkt weiter zu seinem Ausgang.
- Die zweite Schicht ist der sogenannte Hiddenlayer. Dieser wird so genannt, weil er nach außen nicht in Erscheinung tritt.
- Die letzte Schicht ist der Ausgangslayer. Hier wird das zum Eingangsvektor assoziierte Ergebnis ausgegeben.

4.4. Funktionsweise eines neuronalen Netzes

Entscheidend ist natürlich die Frage, was denn nun die Inkarnation der künstlichen Intelligenz solcher neuronalen Netze ausmacht. Diese Intelligenz liegt in der Kombinatorik begründet, d.h. in der Frage, wie die Gewichte w_i die Eingangswerte x_i durchgeben. Weiterhin entscheidend ist, daß man Wege gefunden hat, diese Gewichte so zu trainieren, daß ein neuronales Netz in der gewünschten Weise reagiert.

Der Lernvorgang in einem neuronalen Netz stellt sich wie folgt dar:

a) Initialisierung der Gewichte des Netzes mit Zufallswerten.
b) Auswahl der zu unterscheidenen Objekte.
c) Aufnahme der Trainingssamples von den bekannten Objekten.
d) Anlegen eines Trainingssamples an den Eingangslayer.
e) Berechnung des Ausgangslayer.
f) Vergleich des Istausgangswertes mit dem bekannten Sollausgangswert.
g) Anpassung der Gewichte mittels Fehlerrückführung.
h) Wiederholung der Punkte d) bis g) solange bis eine ausreichende Erkennung gewährleistet ist.

Wir haben in NeuroRob folgende Definition für den Sollausgangswert festgelegt:

> Das Neuron des Ausgangslayers, welches zu dem zu erkennenden Gegenstand gehört, soll eine Ausgangsaktivation von '1' haben. Alle übrigen Ausgangsneuronen sollen eine Ausgangsaktivation von '0' haben. Im Programm NeuroRob wurde die Fehlerrückführung mittels des 'Backpropagation of Errors'-Algorithmus realisiert, um diesem Sollwert gerecht zu werden.

4.5. Backpropagation of Errors

4.5.1. Bei zwei Layern

Widrow und Hoff stellten 1960 das Lernproblem auf 'saubere mathematische Füße', indem sie es als ein Optimierungsproblem begriffen. Hierzu sei ein Zwei-Layer-Netz gegeben (vgl. Bild 8).

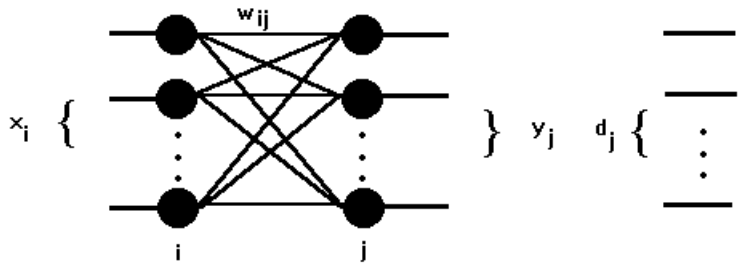

Bild 8
Zwei-Layer-Netz

$$p_j = \sum_i (w_{ij}\, x_i) - \Theta$$

$$y_j = f(p_j)$$

Sie definieren eine Fehlerfunktion E , die es durch Veränderung der Gewichte zu minimieren gilt.

$$E = f(w) = \frac{1}{2} \sum_s \sum_j (y_j - d_j)^2$$

x_i = angelegter Trainingsvektor
y_j = berechneter Ausgangsvektor (Ist-Wert)
d_j = bekannter Soll-Wert des Ausgangsvektors
s = Anzahl der aufgenommenen Beispiele
i = Anzahl der Eingangsneuronen
j = Anzahl der Ausgangsneuronen

Gezeichnet sei eine willkürliche Funktion E=f(w) (siehe Bild 9), die der Einfachheit halber nur von einer Variablen w abhängt:

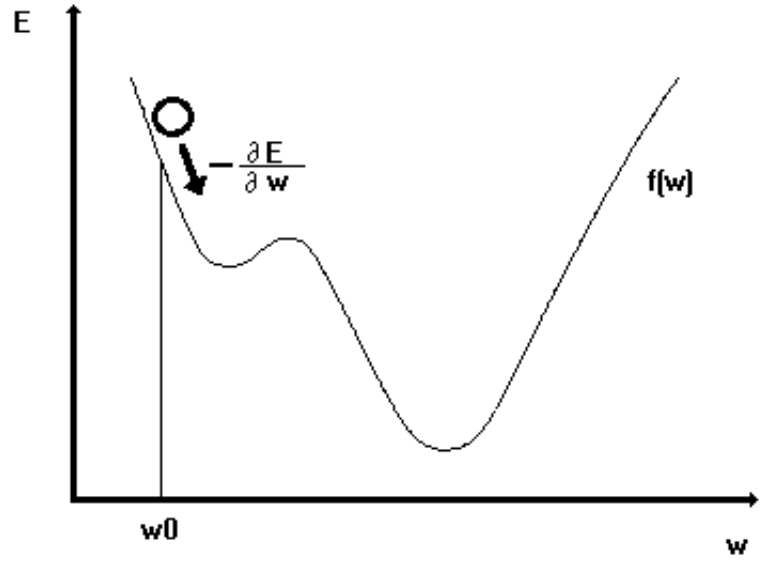

Bild 9
Fehlerfunktion E=f(w)

Hierbei sei w_0 der momentane Wert des Gewichtes. Der Gradient $\frac{\partial E}{\partial w}$ zeigt die Steigung der Funktion f(w) im Punkt w_0 an. Der negative Gradient $-\frac{\partial E}{\partial w}$ zeigt also in Richtung Minimum. Um also das Minimum des Fehlers E zu finden, muß man ein Stück in Richtung des negativen Gradienten gehen, d.h.

$$w_{ij}(t+1) = w_{ij}(t) - \alpha \frac{\partial E(t)}{\partial w_{ij}(t)} \qquad \Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}}$$

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \ \Delta w_{ij}$$

Durch α kann die Schrittweite in Richtung Minimum gewählt werden.

Der Einfachheit wegen fassen wir Θ als ein Gewicht zu einem Eingangselement auf, dessen Wert immer 1 ist. Das heißt wir können einfach schreiben:

$$p_j = \sum_i (w_{ij}\, x_i)$$

Mit der Kettenregel ergibt sich:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial p_j} \frac{\partial p_j}{\partial w_{ij}}$$

wobei

$$\frac{\partial p_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_i w_{ij} x_i \right) = x_i$$ nur ein Term bleibt übrig

$$\frac{\partial y_j}{\partial p_j} = \frac{\partial}{\partial p_j} (f(p_j)) = f'(p_j)$$ die Ableitung der Übertragungsfunktion

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j} E(w) = \frac{\partial}{\partial y_j} \left(\frac{1}{2} \sum_s \sum_j (y_j - d_j)^2 \right) = \frac{1}{2} \sum_s 2 (y_j - d_j) = \sum_s (y_j - d_j)$$

Daraus folgt:

$$\frac{\partial E}{\partial w_{ij}} = \sum_s (y_j - d_j) \; f'(p_j) \; x_i$$

um also den 'wirklichen Gradienten' zu finden, müßte man eine Summation über alle Trainingssamples vollziehen, was manchmal nicht praktisch ist. Widrow & Hoff haben nachgewiesen, daß man auch die Anpassung pro Sample durchführen darf.

Für die Sigmoid-Funktion (siehe Bild 6) bleibt, als letztes noch die Ableitung zu bilden:

$$\frac{\partial y_j}{\partial p_j} = \frac{\partial}{\partial p_j} (f(p_j)) = \frac{\partial}{\partial p_j} \left(\frac{1}{1+e^{-p_j}} \right) = \frac{\partial}{\partial p_j} \left(1+e^{-p_j} \right)^{-1} = \left(1+e^{-p_j} \right)^{-2} \; e^{-p_j}$$

$$\frac{\partial y_j}{\partial p_j} = \frac{e^{-p_j}}{\left(1+e^{-p_j} \right)^2} = f(p_j) \left(1-f(p_j) \right) = f'(p_j)$$

Daraus folgt:

$$\frac{\partial E}{\partial w_{ij}} = (y_j - d_j)\, f'(p_j)\, x_i = \delta_j\, x_i \qquad \text{mit } \delta_j = (y_j - d_j)\, f'(p_j)$$

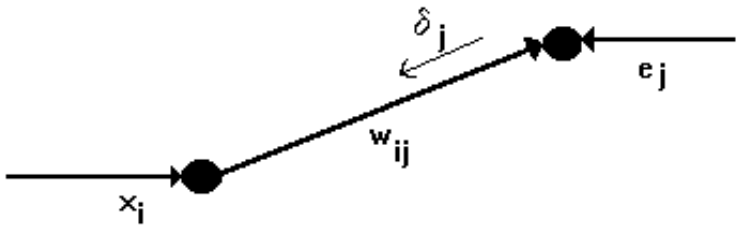

Bild 10
Rückführung des Fehlers e_j auf das Gewicht w_{ij}

wobei $e_j = y_j - d_j$, die Differenz zwischen Ist- und Sollwert ist.

Wenn also die Neuronen eine nicht-lineare Übertragungsfunktion haben, dann wird ein modifizierter Fehler δ_j berechnet, indem man den eigentlichen Fehler e_j mit der abgeleiteten Übertragungsfunktion multipliziert. Daraus folgt auch, daß die Übertragungsfunktion differenzierbar sein muß. Dies ist, wie oben gezeigt, bei der Sigmoid-Funktion der Fall.

4.5.2. Bei n Layern

Die Leistungsfähigkeit bei 2-Schicht Netzen ist sehr begrenzt. Dies wird anhand folgender Tabelle gezeigt:

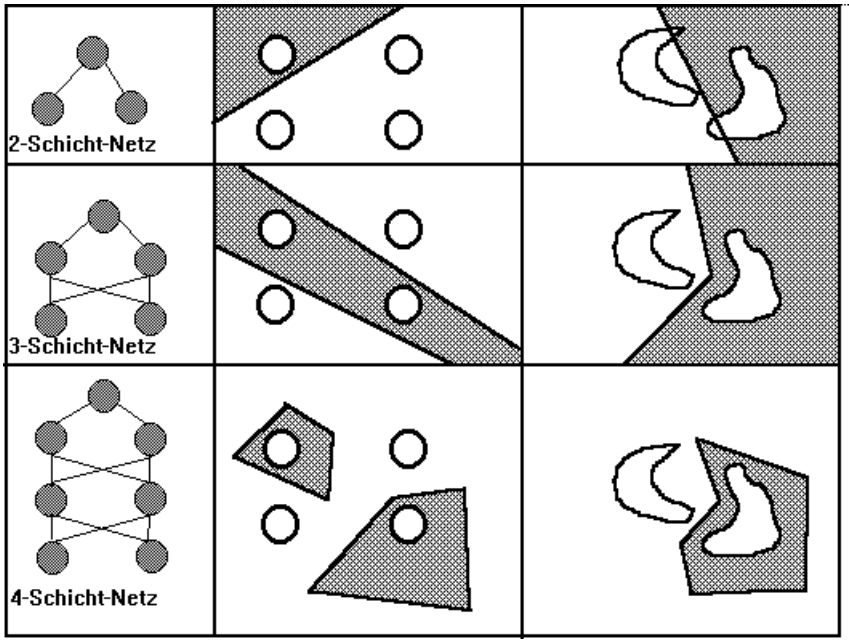

Bild 11
Zusammenhang zwischen der Netzstruktur und der Separierungsfähigkeit

Die Separierungsfähigkeit des 2-Schicht-Netzes ist dadurch begrenzt, daß hier nur eine Separierungslinie gezogen werden kann. In 3-Schicht-Netzen können je nach Anzahl der Hidden-Neuronen dementsprechend viele Linien gezogen werden, wobei zu beachten ist, daß diese zueinander konvex sein müssen. Ein 4-Schicht-Netz kann mit einer genügend großen Anzahl von Hidden-Neuronen beliebig separieren, wobei die Separationslinien auch konkav sein dürfen. In einem 4-Layer-Netz mit genügend Neuronen kann man also jedes Entscheidungsproblem fehlerfrei lösen, wenn es keine Überlappungen gibt.

Es stellt sich allerdings für reale (überlappende) Aufgaben das Problem der Überspezifikation.

Die Frage bei n-Layer-Netzwerken (n>2) ist, wie trainiere ich die Gewichte der versteckten Neuronen. Eine direkte Fehlerrückführung ist bei den Hidden-Neuronen nicht möglich, da es nicht bekannt ist, welche Ausgangswerte diese Neuronen einnehmen sollen. Aus diesem Grund ist der Algorithmus von 2-Layer-Netzen zu überarbeiten.

4.5.2.1. Herleitung

Wir nehmen an, daß wir das folgende Netz (Bild 12) haben:

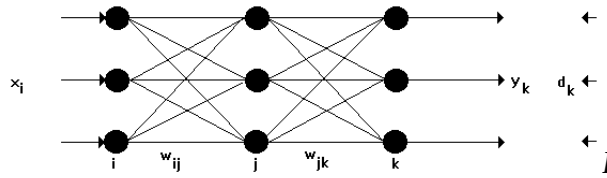

Bild 12
Drei-Layer-Netz

Für den Layer k gilt nach wie vor:

$$\Delta w_{jk} = - \frac{\partial E}{\partial w_{jk}} = - \delta_k x_j \qquad \text{mit } \delta_k = (y_k - d_k) f'(p_k) = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial p_k}$$

wobei d_k der Sollwert ist.

Für den Hidden-Layer j können wir schreiben:

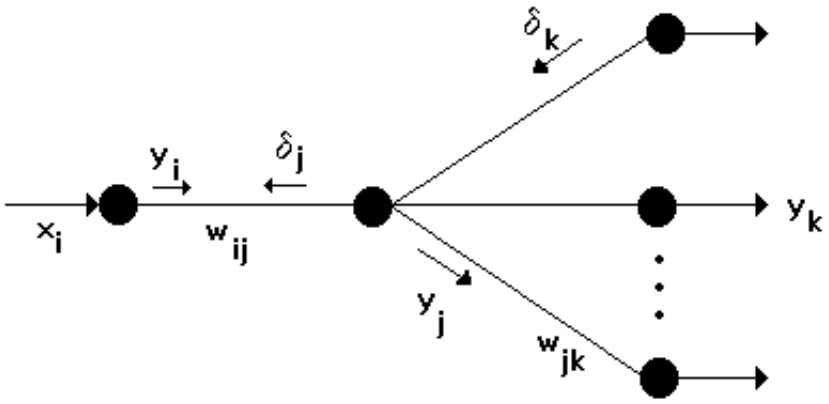

Bild 13
Ausschnitt aus einem Drei-Layer-Netz zur Veranschaulichung einer Gewichtsanpassung

$$\Delta w_{ij} = - \frac{\partial E}{\partial w_{ij}}$$

unter Berücksichtigung der k Vorgänger-Neuronen und durch erweitern erhält man

$$\Delta w_{ij} = - \sum_k \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial p_k} \frac{\partial p_k}{\partial y_j} \frac{\partial y_j}{\partial p_j} \frac{\partial p_j}{\partial w_{ij}}$$

mit:

$$\delta_k = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial p_k} = (y_k - d_k)\, f'(p_k)$$

$$w_{jk} = \frac{\partial p_k}{\partial y_j} \qquad \text{da} \quad p_k = \sum_j w_{jk}\, y_j \quad \Rightarrow \quad \frac{\partial p_k}{\partial y_j} = w_{jk}$$

$$f'(p_j) = \frac{\partial y_j}{\partial p_j} \qquad \text{Ableitung der Sigmoiden}$$

und

$$y_i = \frac{\partial p_j}{\partial w_{ij}} \qquad \text{da} \quad p_j = \sum_i w_{ij}\, y_i \quad \Rightarrow \quad \frac{\partial p_j}{\partial y_j} = y_i$$

folgt daraus:

$$\Delta w_{ij} = -\sum_k \delta_k\, w_{jk}\, f'(p_j)\, y_i$$

$$\Delta w_{ij} = -\, \delta_j\, y_i \qquad \text{mit} \qquad \delta_j = \left(\sum_k \delta_k\, w_{jk} \right) f'(p_j)$$

Die Gewichtsanpassung ergibt sich also wiederum aus einem durchgegebenen Fehler δ_j, der mit der Aktivation des vorangehenden Neurons y_i multipliziert wird. Wir müssen also nur den durchgegebenen Fehler berechnen. Dieser ergibt sich aus der Summe der durchgegebenen Fehler der Ausgangs-Neuronen multipliziert mit dem jeweiligen Gewicht. Dieser Summenwert wird durch das Neuron mit der Abgeleiteten der Übertragungsfunktion zurücktransferiert.

Damit haben wir eine rekursive Rechenvorschrift, die wir für Netze mit beliebig vielen Layern verwenden können !

Während der Klassifizierung und dem ersten Schritt im Training betrachten wir also, wie die Aktivationen von links nach rechts durchgegeben wird (Bild 14). Dadurch wird in den Neuronen die Übertragungsfunktion verwendet.

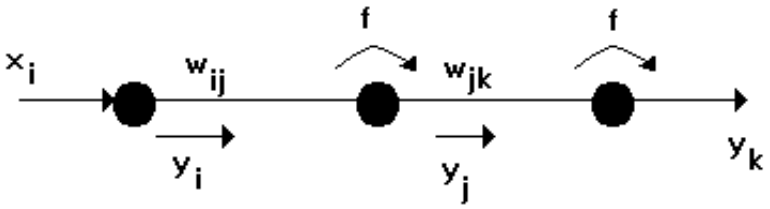

Bild 14
Durchgabe eines Einganges x_i zum Ausgang y_k mittels der Übertragungsfunktion f

Beim Training wird im zweiten Schritt der Fehler von rechts nach links durchgegeben (siehe Bild 15). Dabei wird in den Neuronen die Abgeleitete der Übertragungsfunktion verwendet.

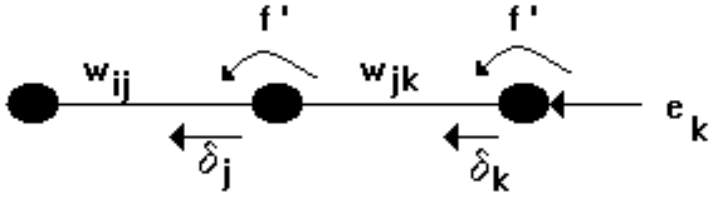

Bild 15
Durchgabe des Fehlers e_k zum Eingang über die Ableitung der Übertragungsfunktion f

Aus diesem Grund nennt man den beschriebenen Algorithmus auch:

Backpropagation of Errors

4.5.2.2. Zusammenfassung

Die wichtigsten verwendeten Formeln, die zur Implementierung des Backpropagation - Algorithmus in NeuroRob notwendig waren und im vorhergehenden Kapitel hergeleitet worden sind, seien hier noch einmal aufgeführt:

(1) $f(p) = \frac{1}{1+e^{-P}}$

(2) $f'(p) = \left(1+e^{-p}\right)^{-2} \; e^{-p}$

(3) $\Delta w_{jk} = - \delta_k \, x_j$ mit $\delta_k = (y_k - d_k) \, f'(p_k)$

(4) $\Delta w_{ij} = - \delta_j \, y_i$ mit $\delta_j = \left(\sum_k \delta_k \, w_{jk} \right) f'(p_j)$

Man berechnet zuerst die Gewichtsanpassungen der Neuronen des Ausgangslayers. Danach ist es möglich, eine Gewichtsanpassung der Neuronen durchzuführen, die im davorliegenden Layer verborgen sind. Die Rekursion des Backpropagation - Algorithmus liegt also in den Gleichungen (3) und (4).

5. Bedienungsanleitung

5.1. Installation von NeuroRob

Für die Installation, sowie die Bedienung von NeuroRob wird vorausgesetzt, daß der Anwender den Umgang mit MS Windows 3.0 beherrscht. Außerdem setzen wir voraus, daß das SIVIPS Programmiersystem installiert ist und der Anwender in der Lage ist, dieses zu bedienen (Hier sei auf die oben erwähnte Diplomarbeit [3] auf Seite 24 verwiesen).

Zuerst ist es notwendig, NEUROROB.CMD mit Hilfe von Sivips zum VIDEOMAT PS zu 'LADEN'. Das eingeladene Programm kann dann auch sofort gestartet werden.

Die Installation von NeuroRob ist sehr einfach. Man kopiert die Dateien NEUROROB.EXE und NEUROROB.POS von der Diskette auf die Festplatte des Rechners auf dem das Programm später laufen soll. Dann startet man Microsoft Windows im erweiterten 386'er Modus. Mit Hilfe des Dateimanagers verschiebt man nun das Programm NEUROROB.EXE in ein Fenster des Programm-Managers.

Die Verbindungen der einzelnen Hardwarekomponenten ist in der oben erwähnten Diplomarbeit [3] auf Seite 33 (3.1 Systemkonfiguration) nachzulesen.

5.2. Starten des Programms NeuroRob

Beim Starten von NeuroRob ist es wichtig, daß MS Windows im erweiterten 386'er Modus läuft. NeuroRob wird durch einen Doppelklick auf das Roboterikon (s. Bild 16) aktiviert.

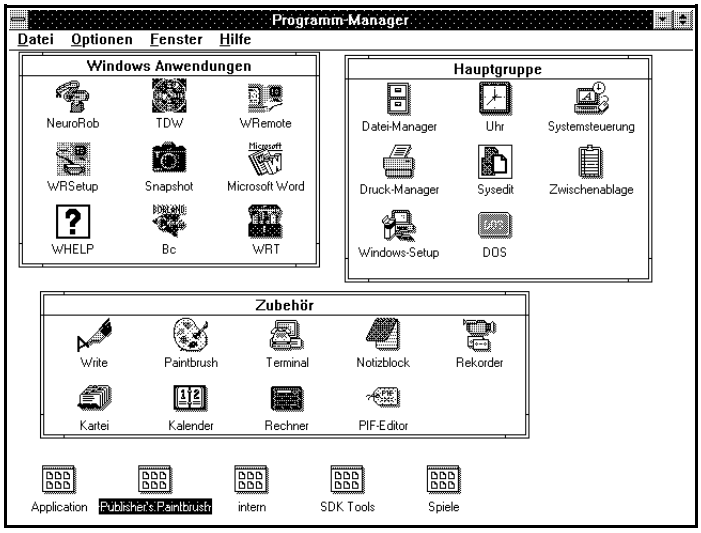

Bild 16
Erscheinungsbild nach Start von Windows

Daraufhin erscheint das Hauptmenue (Bild 17), daß im einzelnen nachfolgend beschrieben wird.

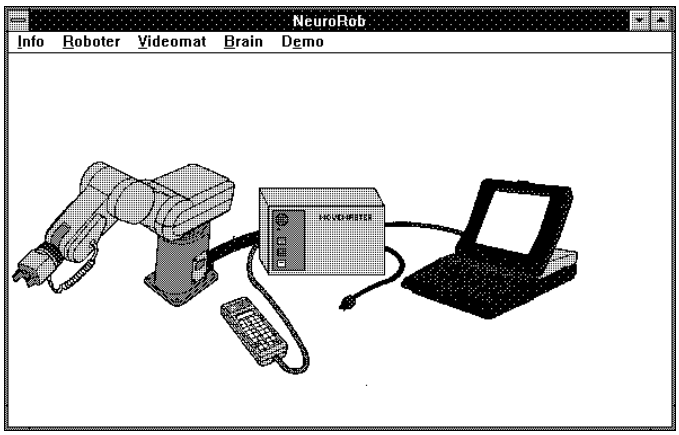

Bild 17
Erscheinungsbild von NeuroRob nach Programmstart

5.3. Menuepunkte von NeuroRob

Menuepunkte vom Hauptfenster:

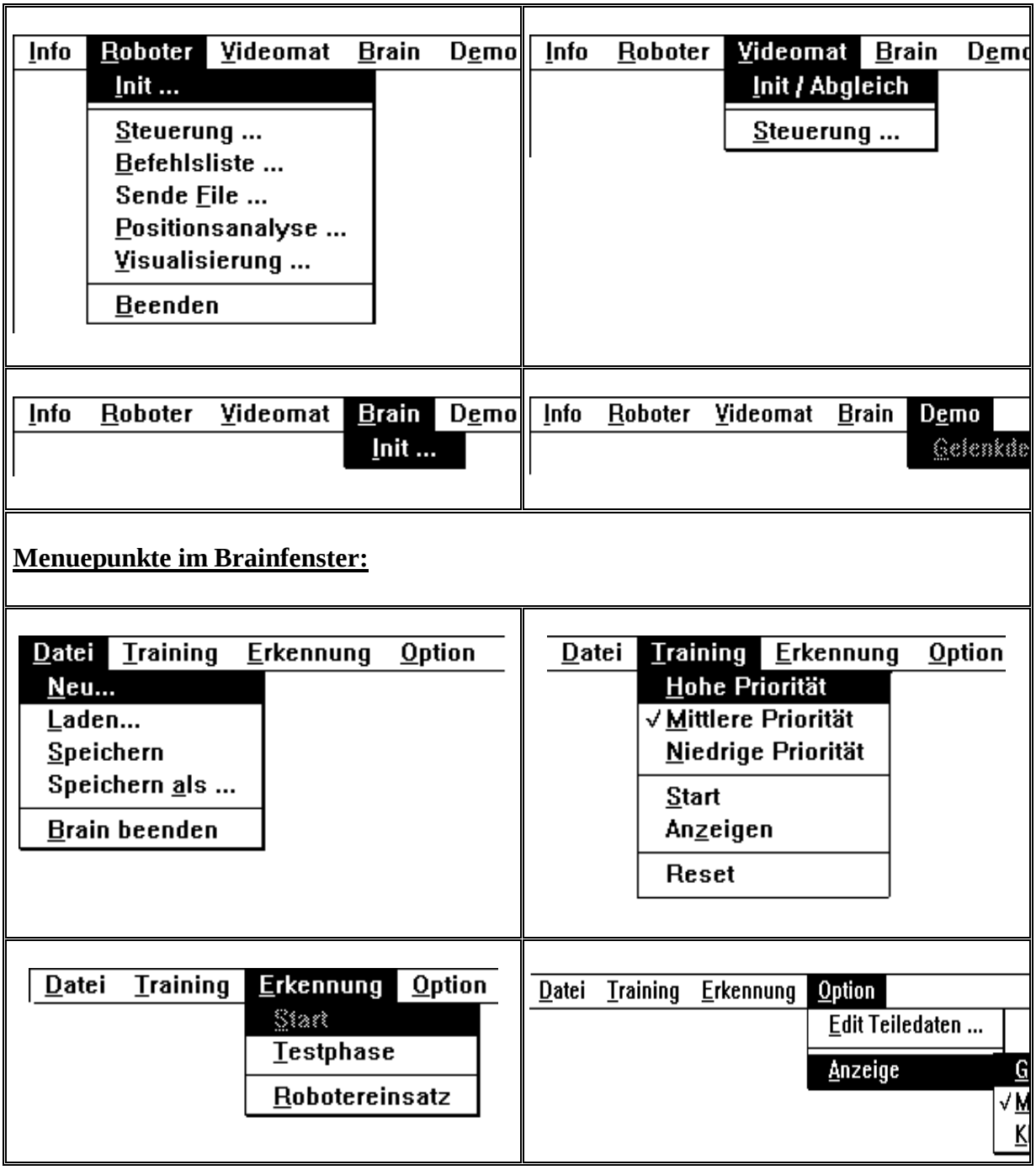

Bild 18
Die Menuepunkte von NeuroRob

Um eine Übersicht über die vorhandenen Menuepunkte von NeuroRob zu bekommen, seien diese in Bild 18 dargestellt. Die nachfolgenden Kapitelüberschriften richten sich nach der Bezeichnung der Menuepunkte. Die Menuepunkte vom 'Brain'-Fenster sind durch ein vorgestelltes 'BrainInit_' gekennzeichnet, da dieses Fenster erst nach dem Betätigen dieses Menuepunktes geöffnet wird.

5.3.1. Info

Nach Anwahl des Menuepunktes Info erscheint folgende Informationsbox (Bild 19).

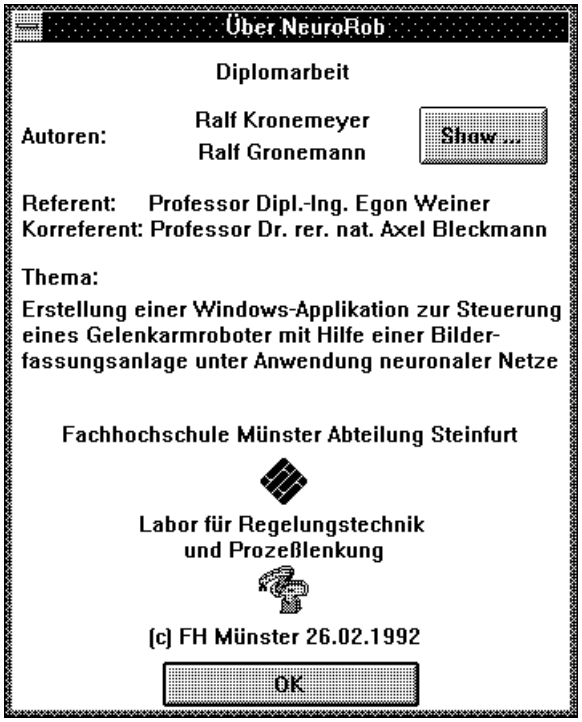

Bild 19
Die Informationsbox

5.3.2. RoboterInit

Bevor eine Arbeit mit dem Roboter möglich ist, muß dieser als erstes initialisiert werden. Hierbei sind zwei Fälle zu unterscheiden.

a) der Movemaster ist eingeschaltet worden, und es wurde noch kein NEST-Kommando gesendet; in diesem Fall ist die entsprechende Option im Dialogfenster (Bild 20) anzuwählen.

b) es wurde bereits ein NEST-Kommando gesendet; in diesem Fall kann direkt das 'OK'-Button gedrückt werden.

Außerdem wird während der Initialisierung die Datei NEUROROB.POS geladen und dem Movemaster gesendet. Dadurch werden für den Roboter vorher festgelegte Positionen (Ablagefächer) definiert, die dann im weiteren Verlauf angefahren werden können.

Bild 20
Roboterinitialisierungbox

5.3.3. RoboterSteuerung

Beim Anklicken des Menuepunktes RoboterSteuerung kann die Roboterkontrolle vom PC übernommen werden. Hier können verschiedene Parameter wie Greifstärke, Schrittweite, Schrittwinkel und Geschwindigkeit eingestellt werden. Bewegt werden kann der Roboter durch die im rechten Abschnitt des Bildes 21 dargestellte Steuerung durch lineare Interpolation, oder mittels die im linken Abschnitt dargestellte Steuerung durch Achseninterpolation.

Außerdem bietet diese Dialogbox die Möglichkeit, Roboterpositionen zu definieren. Hierbei ist folgendermaßen vorzugehen:

Man bewegt den Roboter mittels Steuerung zur gewünschten Position, gibt die Positionsnummer ein, und drückt die Taste 'Definieren'. Zu beachten ist hierbei, daß NeuroRob die Positionsnummern 1 und 900...999 als Systempositionen benutzt, diese sollten nach Möglichkeit nicht geändert werden.

Mit der Taste 'Ursprung' fährt der Roboter zur Position 999, diese liegt ca. 20 cm über dem Koordinatenursprung des auf dem Tisch liegenden Koordinatenkreuzes.

Ein Druck auf 'Hinzufügen' bewirkt, daß die augenblickliche Position des Roboters in die Befehlsliste übertragen wird. Ist diese noch nicht vorher geöffnet worden, so erscheint die dafür zuständige Dialogbox (siehe Menuepunkt RoboterBefehlsliste).

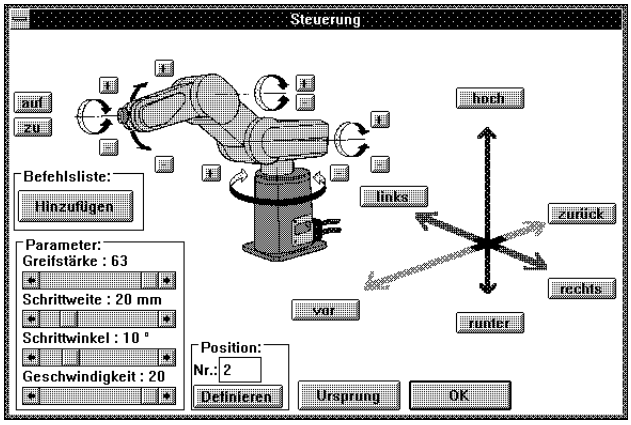

Bild 21
Roboter-Steuerungsbox

Da Windows alle Mausoperationen zwischenspeichert, ist eine vorsichtige Bedienung dieser Dialogbox wichtig, um einen Hardware-Error zu vermeiden.

5.3.4. RoboterBefehlsliste

Eine weitere Dialogbox findet man unter dem Menuepunkt RoboterBefehlsliste (Bild 22). Hier können einfache Befehlsfolgen für den Roboter erstellt werden. Wichtig ist hier, daß der Roboter nur im 'Onlinebetrieb' läuft. Das heißt, daß keine Zeilennummern vor den Befehlen stehen dürfen und außerdem Befehle, die vom Movemaster lesen, nicht verwendet werden können. Die Befehlsfolgen können durch drei verschiedene Methoden erstellt werden. Einmal kann das Programm im 'Notizblock' von MS Windows geschrieben und abgespeichert werden. Durch die Taste 'Laden' kann diese ASCII-Datei dann in die Befehlsliste geladen werden. Die zweite Methode wird mit 'Hinzufügen' realisiert. Hier wird dann, wie in der Steuerungsbox, die augenblickliche Position des Roboters in die Befehlsliste übernommen. Lediglich steht vor jeder Position dann ein 'MP', gefolgt von einem 'GO' bzw. 'GC'. Letztlich bleibt die Möglichkeit, die Option 'Befehl' zu wählen, welche eine weitere Dialogbox öffnet (Bild 23). Hier kann nun eine Befehlszeile eingegeben werden, die der Befehlsliste durch Drücken der 'OK'-Taste angehängt wird. Durch das Button 'Speichern' wird die gesamte Liste in eine Datei gespeichert und kann bei Bedarf wieder geladen werden. Mit 'Löschen' werden markierte Befehle aus der Liste entfernt. Die ganze Liste wird mit 'Markiere alles' markiert, und mit 'Demarkiere alles' demarkiert.

Um einzelne Befehle aus der Befehlsliste dem Roboter zu senden, werden diese manuell markiert und mit der Taste 'Ausführen' dem Movemaster geschickt. Vier Betriebsmodi sind in der Befehlsliste zu unterscheiden:

a) Einzelschrittmodus: Es werden alle markierten Befehle nacheinander gesendet. Bevor ein Befehl übermittelt wird, wird dieser noch durch eine Messagebox angezeigt.

b) Wiederholmodus: Nach Ablauf der Befehlssequenz wird diese von neuem gesendet. Ein Verlassen des Wiederholmodus erreicht man durch Drücken der rechten Maustaste.

c) Kombination von a) und b).

d) weder a) noch b) sind gewählt.

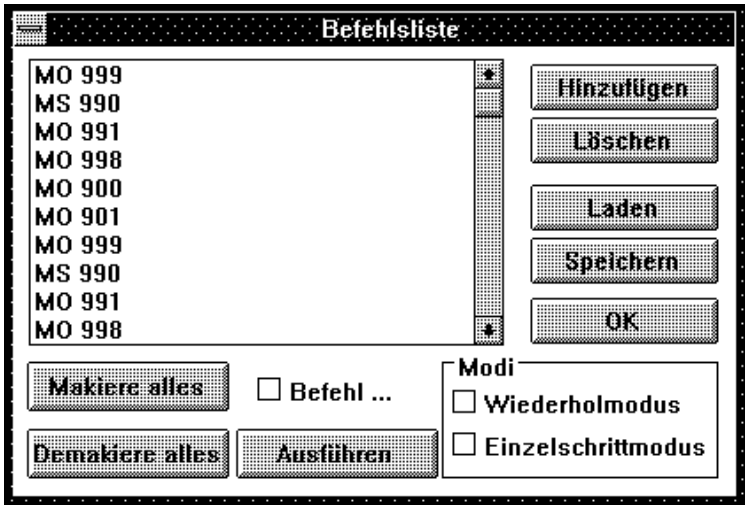

Bild 22
Dialogbox zur einfachen Erstellung einer Befehlsliste für den Roboter

Bild 23
Dialogbox zur Hinzufügung beliebiger Befehle

5.3.5. RoboterSende_File

Durch das Betätigen des Menuepunktes RoboterSende_File wird eine Dialogbox geöffnet, die den üblichen Standarddialogboxen für Fileoperationen entspricht (s. Bild 24). Hier kann man eine Textdatei auswählen, die ein Roboterprogramm repräsentiert. Diese wird geladen und zum Movemaster geschickt. Hier ist ebenfalls zu beachten, daß keine Befehle benutzt werden, die den Roboter zu einer Antwort auffordern.

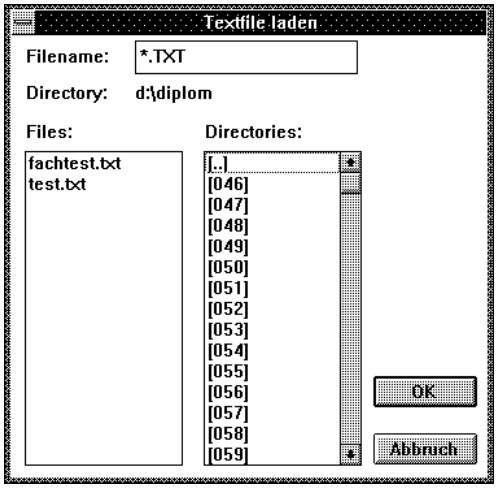

Bild24
Fileselectorbox

5.3.6. RoboterPositionsanalyse

Eine Erweiterung für das Handling des Roboters ist die Positionsanalyse.

Man sieht in Bild 25 zur linken eine Listbox. Diese kann durch zwei Möglichkeiten mit 'Leben' gefüllt werden. Durch das Drücken des Knopfes 'Transfer' sendet der Movemaster dem Rechner die schon definierten Roboterpositionen (von 1...999). Dieser Vorgang dauert ca.3 min.. Die zweite Möglichkeit ist die Taste 'Laden', hier wird eine Datei, die mit dem Suffix '.POS' endet, in der Listbox sichtbar gemacht, während sie außerdem im Movemaster-RAM gespeichert wird. So werden dann neue Roboterpositionen festgelegt. Mit 'Speichern' wird der komplette Boxinhalt als ASCII-Datei gespeichert. Defaultmäßig ist auch hier das Suffix '.POS' eingestellt.

Wird eine Positionsnummer markiert, erscheinen ihre Positionsdaten in den dafür vorgesehenen Feldern. Zudem kann eine angeklickte Nummer durch 'Move' angefahren werden, und mit 'Neu' bei Bedarf wieder gelöscht werden. Verlassen kann man die Dialogbox 'Positionsanalyse' durch 'OK'.

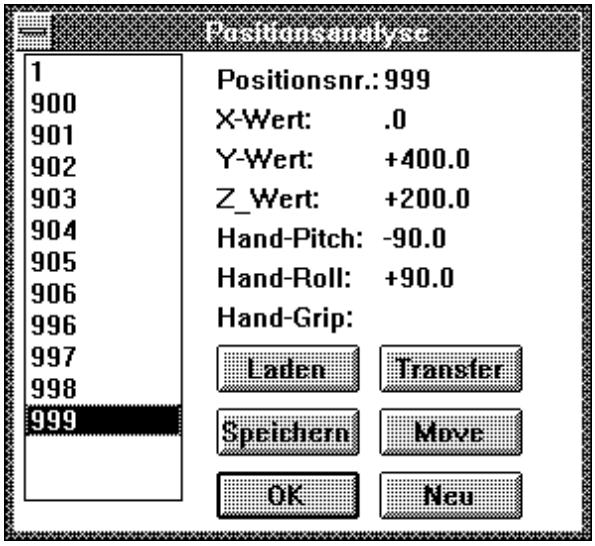

Bild 25
Dialogbox für eine Positionsanalyse des Movemasters

5.3.7. RoboterVisualisierung

Im Menuepunkt RoboterVisualisierung erscheinen drei Koordinatenkreuze (x-y, x-z, y-z). In diesen Grafiken wird die Position des Tool-Center-Points dargestellt (Bild 26).

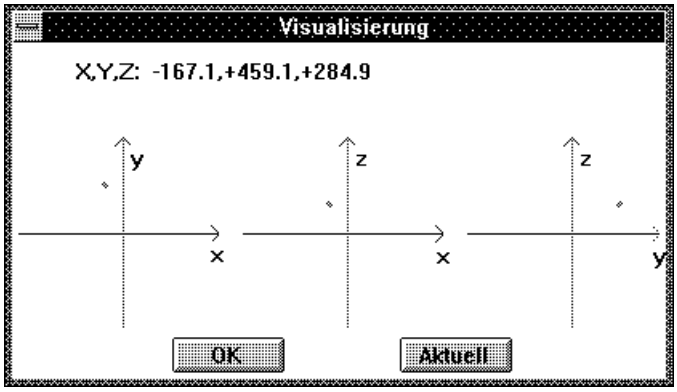

Bild 26
Visualisierungsbox

5.3.8. RoboterBeenden

Diese Funktion beendet das Programm NeuroRob. Sollte ein neuronales Netz erstellt oder verändert worden sein, wird vor Beendigung des Programms die Möglichkeit eingeräumt, dieses abzuspeichern.

5.3.9. VideomatInit/Abgleich

Beim Starten des Programms NeuroRob ist es wichtig, Movemaster und Bildverarbeitungsanlage zu initialisieren. Initialisieren dient hauptsächlich zur Konfiguration der Geräteschnittstellen. Zudem wird hier ebenfalls ein Abgleich der Roboterkoordinaten auf die Koordinaten der Videomatanlage durchgeführt. Nach Anklicken dieses Menuepunktes, wird man mittels einer Messagebox dazu aufgefordert, einen Abgleichgegenstand (dies ist bei uns ein 1-Pfennigstück) auf den markierten Punkt des Tisches zu legen. Nach erfolgreicher Initialisierung sollte man diesen Gegenstand sofort wieder entfernen.

5.3.10. VideomatSteuerung

Durch diesen Menueunterpunkt (Bild 27) wurde es möglich, vom PC aus die Bildverarbeitungsanlage zu steuern. Durch die verschiedenen, sich selbst erklärenden Knöpfe der Dialogbox, ist eine andere Darstellungsart des Bildes auf dem VIDEOMAT-Monitor möglich. So kann z.B. die Binärisierungsschwelle nachträglich (im Sivips-Programm ist die Schwelle auf 58 voreingestellt) verändert werden.

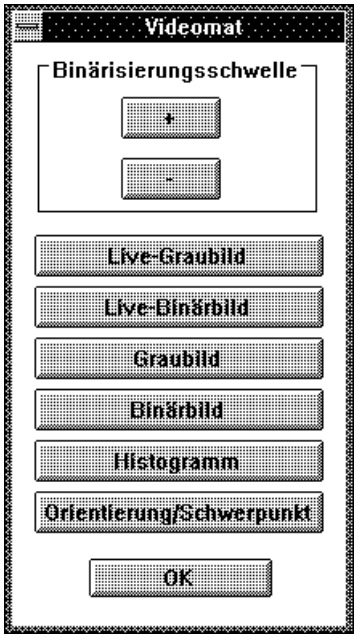

Bild 27
Dialogbox zur Steuerung des VIDEOMAT PS

5.3.11. DemoGelenkdemo

Zur Demonstration der Beweglichkeit des Roboters wurde eine kleine Gelenkdemo geschrieben. Diese deutet unter anderem auch den Aktionsradius an.

5.3.12. BrainInit

Dieser Menuepunkt ist ein sehr wichtiger Bestandteil des Programms NeuroRob. Nach Anwahl von BrainInit erscheint das in Bild 28 dargestellte Fenster BRAIN. Von hier aus wird das neuronale Netz aufgebaut und trainiert. Weiterhin werden hier Optionen bezüglich des Robotereinsatzes und der Grafik bereitgestellt.

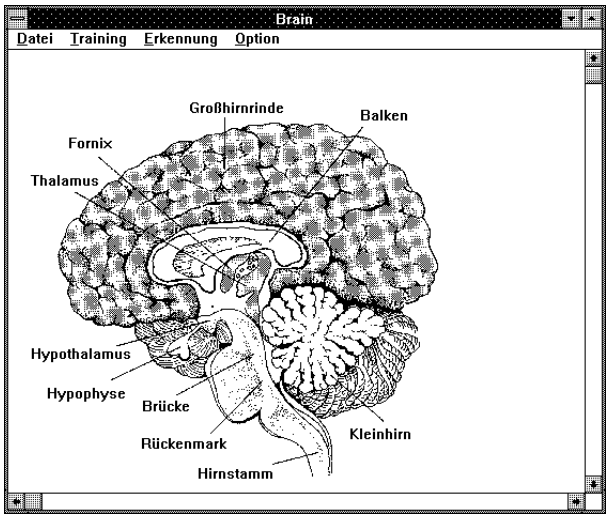

Bild 28
Brain-Fenster nach der 'Erstinitialisierung'

5.3.13. BrainInit_DateiNeu

Um ein neues neuronales Netz zu erstellen, muß der Untermenuepunkt BrainInit_DateiNeu angewählt werden. Es erscheint die Dialogbox 'Neues Netz anlegen' (Bild 29), in der die für ein Netz wichtigen Parameter einzustellen sind. Dazu zählen die Erkennungsmerkmale für das Inputlayer (max. 26), Anzahl der zu erkennenden Teile (max. 16) für das Outputlayer und die Größe der Hiddenlayer (max. 32). Die Hiddenlayer können in ihrer waagerechten Ausdehnung durch die Parametereinstellung 'Hiddenlayer' und in ihrer senkrechten durch 'Neuronenanzahl' varriert werden. Ein weiterer Punkt in dieser Box ist 'Sample pro Teil'. Hier kann angegeben werden, wieviele Beispiele (max. 8) von einem Gegenstand aufgenommen werden sollen. Von diesem Gegenstand werden dann mit oder ohne der Hilfe des Roboters Eigangsvektoren für das neuronale Netz eingelesen.

Eine Selektion von 'Grauwert-Histogramm' schlägt beim Eingangslayer mit 16 Neuronen zu Buche (aus Speicherplatzgründen werden 4 Grauwerte in einem Neuron zusammengefaßt). Somit ergibt sich eine maximale Neuronenanzahl von 26. Alle anderen Merkmale benötigen jeweils genau ein Eingangsneuron.

Die Optionen für den Robotereinsatz können unten rechts festgelegt werden.

kein: der Roboter wird nicht eingesetzt

nur Training: der Roboter wird nur zum Training benutzt
komplett: der Roboter wird zum Training und während des Run-Modus benutzt

Bild 29
Dialogbox zum erstellen eines neuen neuronalen Netzes

Nachdem BrainInit_DateiNeu durch 'OK' verlassen wurde, erscheint eine weitere Dialogbox mit dem Namen 'Festlegung der Teiledaten'. Hier werden die Namen, Grösse und die eventuelle Fachnummer eingetragen:

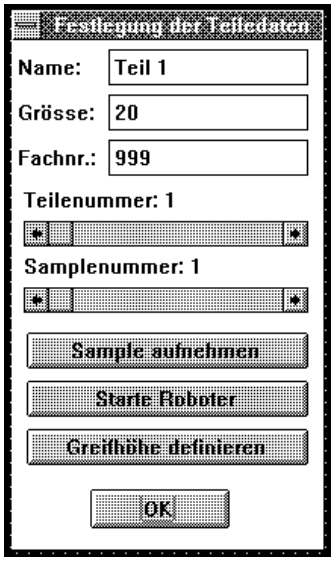

Bild 30
Dialogbox zur Festlegung teilespezifischer Daten

Wenn der Robotereinsatz beim Aufnehmen der Samples gewünscht wird, so ist es notwendig, die Teile auf den Ursprung des aufgezeichneten Koordinatensystems zu legen, die maximale Ausdehnung des Teils abzulesen und in die Dialogbox einzutragen. Danach wird der Roboter mittels der Steuerungsbox auf die gewünschte Greifhöhe gefahren. Hiernach muß man den Knopf 'Greifhöhe definieren' drücken. Jetzt kann man, mit 'Starte Roboter', die Samples automatisiert aufzunehmen. Optional ist es auch möglich, einzelne Trainingsvektoren direkt mittels 'Sample aufnehmen' einzulesen. Hierbei ist es notwendig, über die Scrollbalken sowohl 'Samplenummer' und als auch 'Teilenummer' einzustellen.

Nachdem von jedem Gegenstand alle Samples aufgenommen worden sind, kann man mit 'OK' die Dialogbox verlassen. Nun erscheint im Fenster 'Brain' das neuronale Netz (Bild 31) auf dem Bildschirm.

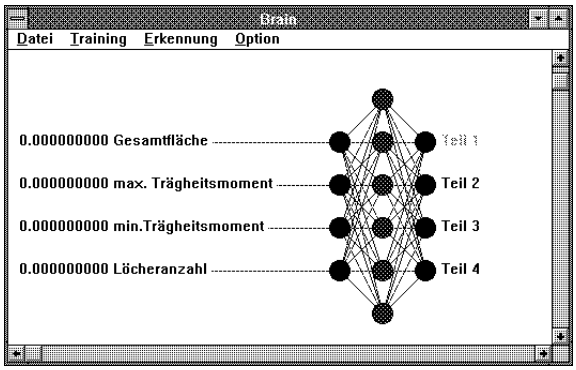

Bild 31
Darstellung eines neuronalen Netzes im 'Brain-Fenster'

Bei einem Doppelklick auf einzeln Neuronen erscheint folgende Dialogbox (s. Bild 32):

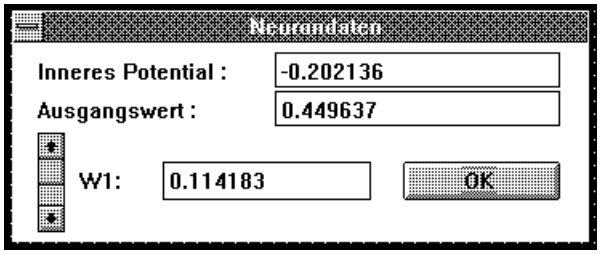

Bild 32
Dialogbox zur Untersuchung neuroneninterner Parameter

Hier kann man die inneren Neuronendaten, wie sie im Kapitel über neuronale Netze beschrieben sind, untersuchen und editieren.

5.3.14. BrainInit_DateiLaden

Um ein schon vorhandenes und vielleicht schon trainiertes Netz zu laden, geht man in das Menue BrainInit_DateiLaden. Diese Dialogbox ähnelt den Standardboxen für Fileoperationen (siehe auch unter Menuepunkt RoboteSende_File).

5.3.15. BrainInit_DateiSpeichern

Mit BrainInit_DateiSpeichern können neuronale Netze gespeichert werden. Interessant ist, daß auch schon trainierte Netze gespeichert werden können. So ist es möglich ein schon trainiertes Netz zu laden, weiter zu trainieren und anschließend wieder zu speichern.

5.3.16. BrainInit_DateiSpeichernals

Wie BrainInit_DateiSpeichern, nur wird hier noch einmal nach dem Dateinamen gefragt unter dem gespeichert werden soll.

5.3.17. BrainInit_Brainbeenden

Durch diesen Unterpunkt wird das Fenster von Brain geschlossen. Man kehrt in das Hauptfenster von NeuroRob zurück.

5.3.18. BrainInit_TrainingHohePriorität

In diesem Menuepunkt kommt dem Training des neuronalen Netzes eine hohe Priorität zu. Das bedeutet, daß man für andere Programmaufgaben (Multitasking) weniger Zeit zur Verfügung hat.

5.3.19. BrainInit_TrainingMittlerePriorität

dito, jedoch Zuteilung einer mittleren Priorität.

5.3.20. BrainInit_TrainingNiedrigePriorität

dito, jedoch Zuteilung einer niedrigen Priorität.

5.3.21. BrainInit_TrainingStart

TrainingStart startet den Trainingsvorgang des neuronalen Netzes mit den eingestellten Optionen. Wenn der Trainingsvorgang eingeschaltet ist, so sind einige Menuepunkte in 'Brain' nicht mehr anwählbar, auch ist es nicht möglich, während des Trainings über ein Doppelklick auf ein Neuron die internen Parameter der Neuronen abzufragen.

5.3.22. BrainInit_TrainingAnzeigen

Dabei werden interessante Trainingseigenschaften in der Grafik zum Netz eingeblendet (siehe Bild 33). Hierbei werden abwechselnd alle Trainingsvektoren an den Eingangs-Layer gelegt, daraus der Ausgang berechnet und angezeigt. In grüner Schrift wird das Teil geschrieben, dessen Neuron die höchste Aktivität besitzt, also auch am stärksten rot gefärbt ist. Der Trainingsvorgang ist dann abgeschlossen, wenn Soll- und Istwert des Ausgangs-Layers gleich groß sind. Das wird dadurch kenntlich gemacht, daß ein grüner Pfeil hinter dem grünen Text erscheint. Zu beachten ist, daß dies bei jedem Teil und Sample der Fall sein soll.

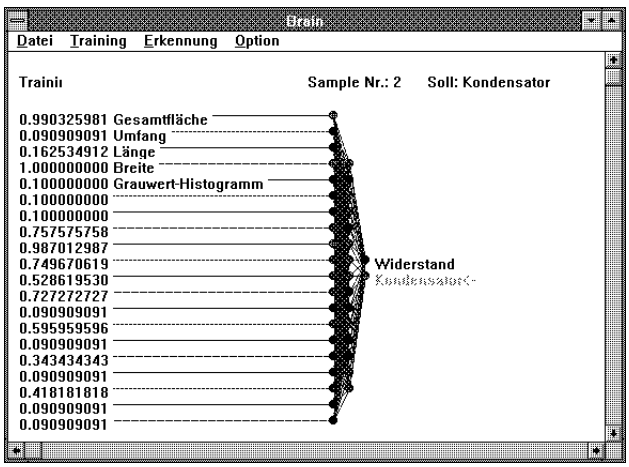

Bild 33
Das Brain-Fenster im Trainingsmodus

5.3.23. BrainInit_TrainingReset

Das neuronale Netz wird durch TrainingReset zurückgesetzt, d. h. alle Gewichte der Neuronen bekommen neue Zufallswerte. Dies ist dann Sinnvoll, wenn das neuronale Netz gegen ein ungünstiges Minimum konvergiert.

5.3.24. BrainInit_ErkennungStart

Durch diesen Menuepunkt wird die Erkennung gestartet. Während dieser Betriebsart (Run-Modus) erscheint auf dem Bildschirm das neuronale Netz, wo im Outputayer das Ergebnis der Bilderkennung steht. Abhängig davon, ob der Robotereinsatz selektiert wurde oder nicht, verhält sich das Erkennungssystem wie folgt:

a) ohne Roboter: NeuroRob holt sich ständig die Merkmale der Szene, ohne Berücksichtigung auf eine evtl. zu große Anzahl von Teilen (sollte das VIDEOMAT PS mehrere Teile binärisieren, so werden nur die Merkmale des ersten Teils ausgewertet). Die extrahierten Merkmale werden sofort in das neuronale Netz eingegeben. Dieses liefert dann das Ergebnis der Erkennung. Es ist darauf zu achten, daß das Ergebnis erst ca. 3 Sekunden nach Ablage in den Bilderkennungsbereich seine absolute Gültigkeit hat. Dieser Modus hat den Vorteil, daß man hier im Prinzip alle Objekte zur Bilderkennung heranziehen kann. Einschränkungen bezüglich der Binärisierbarkeit sind hier nicht gegeben, somit eignet sich dieser Modus ideal zur Demonstration der Leistungsfähigkeit neuronaler Netze.

b) mit Roboter: Bedingung für den Einsatz von diesem Modus ist, daß die eingelernten Teile binärisierbar sind, d.h. es sollten keine Gegenstände sein, die Grauwerte besitzen, die der Unterlage entsprechen, also keine sehr hellen Gegenstände. Mit Hilfe der Menueoption 'ErkennungTestphase' läßt sich dieses sehr gut überprüfen.

NeuroRob überprüft als erstes, ob sich in der Szene wirklich nur ein (binärisiertes) Teil befindet. Ist dies der Fall, so wird noch getestet, ob sich dieses Teil evtl. bewegt. Erst wenn sich nur ein sich nicht bewegendes Teil im Bilderfassungsbereich befindet, holt sich NeuroRob die Merkmale dieses Gegenstandes und wertet sie mittels des neuronalen Netzes aus. Danach ermittelt NeuroRob den Schwerpunkt und die Orientierung (Lage der Hauptträgheitsachse) des Gegenstandes und veranlaßt den Roboter, diesen zu greifen und in ein dafür vorgesehenes Fach abzulegen. Jetzt kann wieder ein Teil in die Szene gelegt werden.

5.3.25. BrainInit_ErkennungTestphase

Um ein evtl. unkorrektes Greifen des Roboters auszuschließen, sollte man beim erstmaligen Robotereinsatz diese Menueoption wählen. Hierbei führt der Roboter zwar den Greifvorgang aus, greift jedoch oberhalb des definierten Greifpunktes zu, so daß das Objekt nicht gefaßt wird. Ist eine korrekte Greifweise zu erkennen, so kann diese Option durch nochmaliges drücken dieses Menuepunktes abgeschaltet werden.

5.3.26. BrainInit_ErkennungRobotereinsatz

Es kann hier entschieden werden, ob der Roboter die erkannten Gegenstände wegräumen soll oder nicht. Bedingung dafür ist allerdings, daß beim Training auch die Option Robotereinsatz 'komplett' gewählt wurde (siehe auch unter Menuepunkt BrainInit_RoboterStart).

5.3.27. BrainInit_OptionEdit_Teiledaten

Die OptionEdit_Teiledaten ermöglicht, bei schon vorher eingelernten Gegenständen, die Festlegung der Teiledaten zu editieren, bzw. einen kompletten Sample neu einzulesen. Es erscheint also auch hier die Dialogbox 'Festlegung der Teiledaten'.

5.3.28. BrainInit_OptionAnzeige

In dem Menuepunkt OptionAnzeige wird die Darstellungsgröße des neuronalen Netzes eingestellt. Hierbei kann zwischen großer, mittlerer und kleiner Grafikausgabe gewählt werden. Sinnvoll ist dieser Punkt bei großen bzw. kleineren Netzen, um diese noch gut sichtbar zu machen.

5.4. Bedienungsbeispiel

Im folgenden sei die Bedienung von NeuroRob anhand eines kleinen Beispiels demonstriert. Die Bedienungsanleitung sollte vor dem Durchgehen des Beispiels gründlich gelesen worden sein. Hier möchten wir zwei Teile (einen Widerstand und einen Kondensator) erkennen und in definierte Fächer ablegen.Kurz beschrieben sind dazu folgende Schritte notwendig:

Durch Anklicken des Roboterikons befinden wir uns im Hauptfenster von NeuroRob (Bild 34).

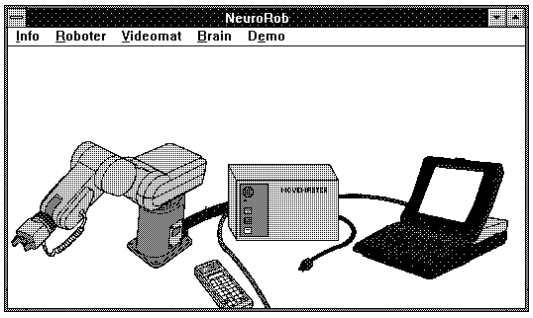

Bild 34

Es folgt die Initialisierung des Roboters durch Anwählen des Menuepunktes RoboterInit. In der erscheinenden Dialogbox ist die Option 'Nest - Kommando senden' zu selektieren. Danach drückt man 'OK' (s.Bild 35).

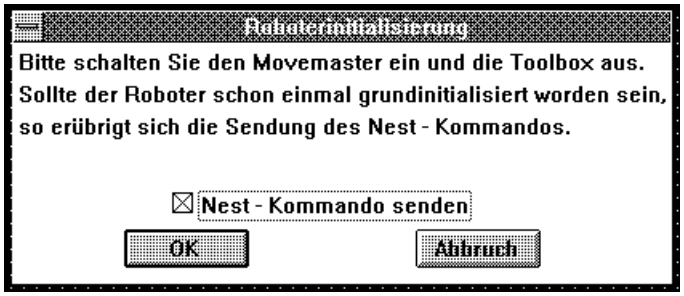

Bild 35

Nach der Initialisierung des Roboters folgt die des VIDEOMAT PS. Dazu wählt man 'VideomatInit/Abgleich', woraufhin eine Messagebox (s. Bild 36) erscheint, die dazu auffordert, den Abgleichgegenstand auf den markierten Ursprung der Videomat-Szene zu legen. Dieses wird mit 'OK' quittiert.

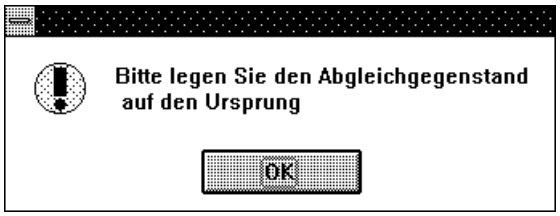

Bild 36

Nach dem Weglegen des Abgleichgegenstandes, sind die vorbereitenden Maßnahmen für die Arbeit mit NeuroRob abgeschlossen. Mit 'BrainInit' gelangt man in das Fenster von Brain (Bild 37).

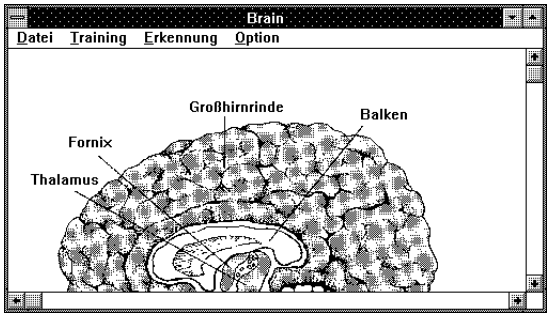

Bild 37

Hier wählt man 'DateiNeu' im Brain-Menue. In der erscheinenden Dialogbox wählt man die Optionen wie sie in Bild 38 dargestellt sind.

Bild 38

Nach 'OK' gibt man nun die teilespezifischen Daten ein (Bild 39):

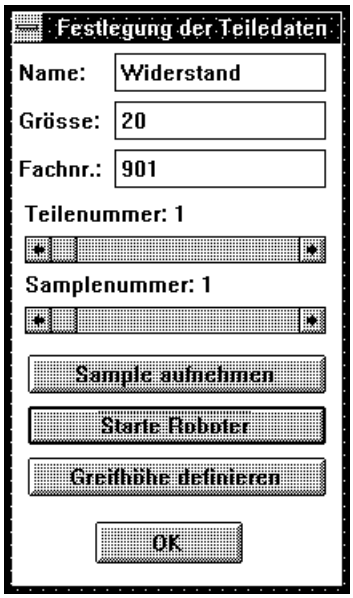

Bild 39

für denWiderstand sind folgende Eingaben zu machen:

Name: Widerstand
Grösse: 20
Fachnr: 901

nun fährt man mittels 'RoboterSteuerung' (Bild 40) vorsichtig den Greifpunkt des Widerstandes an. Dies bewerkstelligt man am besten mit 'runter'. Jetzt drückt man 'Greifhöhe definieren' in der Dialogbox 'Festlegung der Teiledaten' und danach 'Starte Roboter'. Der Roboter greift den Widerstand und der PC nimmt die Trainingsvektoren auf. Nachdem der Roboter den Widerstand auf den Roboter-Ursprung wieder abgelegt hat, gibt man nun die charakteristischen Daten des Kondensators an:

Name: Kondensator
Grösse: 20
Fachnr: 902

Wieder ist der Greifpunkt anzufahren, 'Greifhöhe definieren' und 'Starte Roboter' zu drücken. Nachdem der Roboter auch hier alle Samples durchgefahren hat, kann man 'OK' drücken.

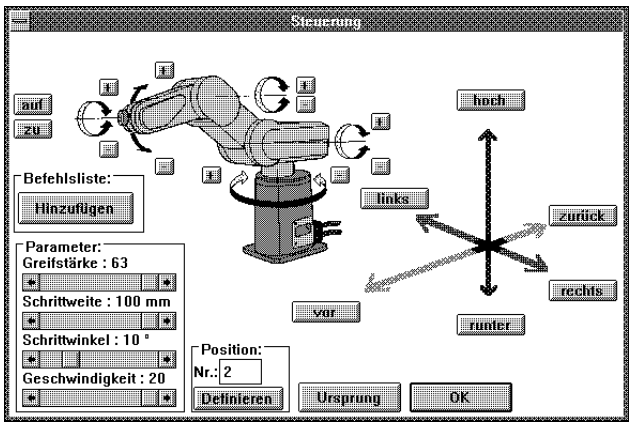

Bild 40

Es erscheint im Fenster von 'Brain' das noch untrainierte neuronale Netz (Bild 41).

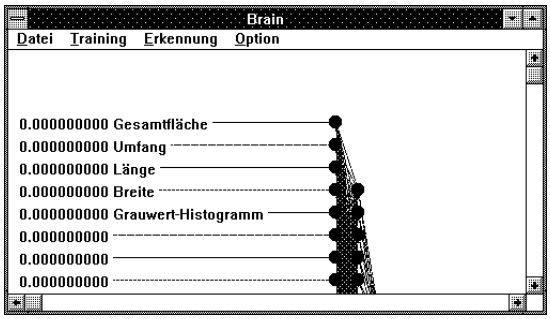

Bild 41

Durch 'OptionAnzeigeKlein' sowie durch Vergrößern des Fensterausschnitts ist das neuronale Netz komplett im Fenster darstellbar (s. Bild 42).

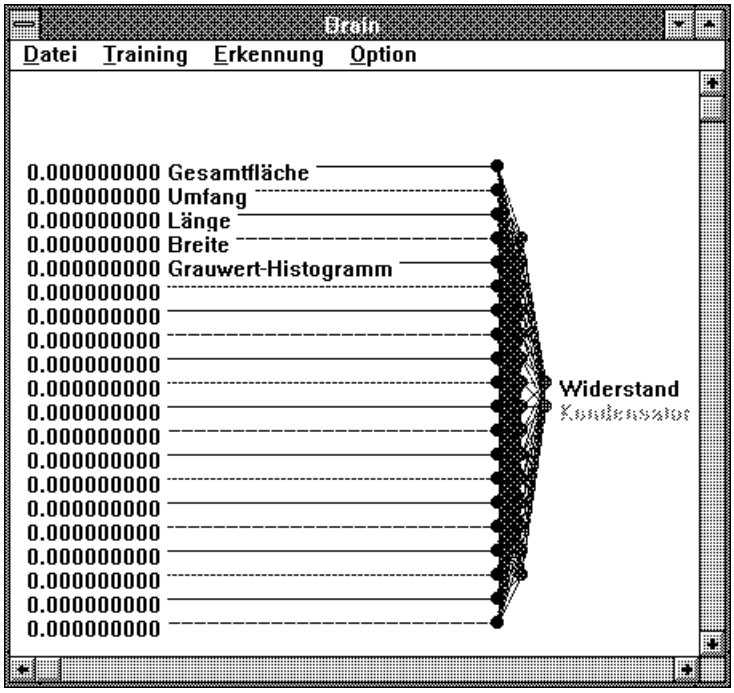

Bild 42

Um das Netz zu trainieren, wählt man 'ErkennungTrainingStart'. Um sich einen Überblick über den Trainingsvorgang zu verschaffen, selektiert man 'TrainingAnzeigen'. Nach erfolgreichem Training (genaueres hierzu unter dem Kapitel 5.3.22 TrainingAnzeigen) schaltet man durch erneuter Wahl von 'ErkennungTrainingStart' diesen wieder aus.

Es ist 'ErkennungRoboterEinsatz' anzuklicken. Man gelangt in den Runtime-Modus durch 'ErkennungStart'. Jetzt ist NeuroRob voll im Einsatz. Legt man nun einen der beiden Gegenstände in den Erfassungsbereich der Kamera, so sollte NeuroRob den Gegenstand erkennen und den Roboter veranlassen, das erkannte Objekt zu greifen und in das dafür vorgesehene Fach abzulegen.

6. Zusammenfassung

6.1. Ergebnis und Beurteilung der Arbeit

Die uns gestellte Aufgabe wurde erfolgreich gelöst. Wir haben mit NeuroRob ein System entwickelt, mit dem es möglich ist, beliebige Gegenstände anhand ihrer Grauwerte zu unterscheiden. Weiterhin kann man Objekte, die binärisierbar sind (also einen genügend großen Kontrast zum Hintergrund bilden), anhand weiterer Merkmale erkennen und diese dann auch mit Hilfe des Roborters in definierte Fächer einsortieren.

Bemerkenswert ist hier, daß wir durch die Funktionsbaugruppe 3, welche eine Grauwertverarbeitung ermöglicht, in der Lage sind, Objekte ohne Leuchttisch zu erkennen.

Die Leistungsfähigkeit der neuronalen Netze hat unsere Erwartungen bei weitem übertroffen. Anfängliche Bedenken bezüglich der Dauer des Lernvorganges sind unbegründet gewesen, da es sich gezeigt hat, daß der Zustand des Netzes im allgemeinen recht schnell zu einem Minimum konvergiert. Trainingszeiten im Minutenbereich (486 PC) reichten fast immer aus, um eine hohe Erkennungsrate zu erreichen. Bei sich stark ähnelnden Gegenständen sollte man unbedingt möglichst viele Trainingsvektoren aufnehmen. Überschneiden sich die Merkmalsvektoren, so ist natürlich auch ein neuronales Netz nicht in der Lage, die Gegenstände zu 100% richtig zu klassifizieren. In diesem Fall versucht das Netz, den Bayes-Klassifikator zu ziehen (vgl. Bild 43 und 44).

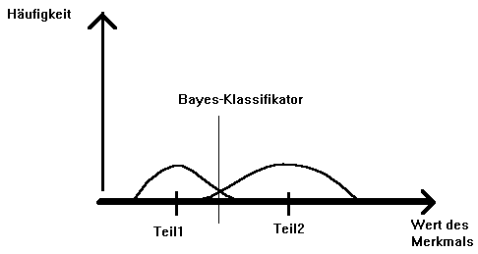

Bild 43
Der Bayes-Klassifikator bei nur einem Merkmal

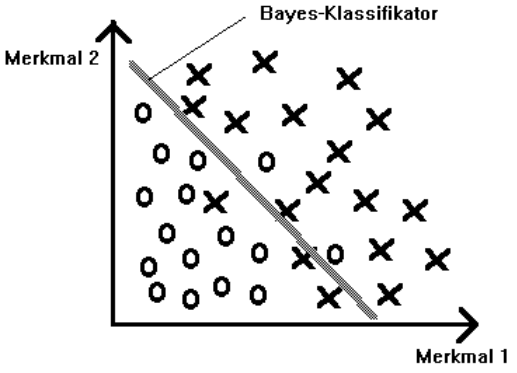

Bild 44
Der Bayes-Klassifikator bei zwei Merkmalen

Ein wesentliches Leistungsmerkmal von NeuroRob ist die Fähigkeit, Gegenstände unabhängig von ihrer Lage (egal, ob sie auf dem 'Bauch', dem 'Rücken', der 'Seite' usw. liegen) zu erkennen. Dies ist in der Praxis von großer Bedeutung, da es nicht gewährleistet werden

kann, daß z.B. vom Fließband fallende Teile immer auf den Rücken fallen. Für die Gewährleistung dieser Fähigkeit ist es natürlich notwendig, daß man alle möglichen Lagen auch als Trainingsvektoren aufnimmt.

Weiterhin besteht die Möglichkeit, Klassen von Gegenständen zu bilden. Hierzu diene folgendes Beispiel: Man habe eine Reihe von Widerständen und Kondensatoren, die untereinander auch noch verschiedene Werte haben und sich somit allesamt voneinander unterscheiden. Nimmt man nun für Teil 1 von jedem Widerstand - und für Teil 2 von jedem Kondensator - mindestens einen Merkmalsvektor auf, so ist NeuroRob nach genügend langer Trainingszeit in der Lage, zwischen der Klasse von Widerständen (Teil1) und der Klasse von Kondensatoren (Teil2) zu unterscheiden.

Bei der Erstellung eines neuronalen Netzes steht man stets vor dem Problem, wieviele Neuronen man im Hiddenlayer einsetzt. Die Anzahl der Eingangs- und Ausgangsneuronen ist gegeben. Für die Größe des Hiddenlayer gibt es aber (bisher) kein allgemeingültiges Gesetz. Eigene Untersuchungen haben ergeben, daß ein Netz selbst mit nur einem Hiddenneuron noch erkennungsfähig bleibt. Die Gewichtsanpassung pro Sample geht hier natürlich sehr schnell, da man kaum Gewichte zum Anpassen hat. Dafür muß man jedoch die Trainingsdurchläufe wesentlich erhöhen, um ein einigermaßen gutes Ergebnis zu bekommen. Das andere Extrema (viele Hiddenneuronen) hat den Vorteil, daß man nur wenig Trainingsdurchläufe benötigt, dafür dauert aber die Gewichtsanpassung pro Sample wesentlich länger, da die Anpassungszeit exponentiell mit der Neuronenzahl wächst. Gute Ergebnisse haben wir eigentlich stets mit dem arithmetischen Mittel von der Anzahl der Eingangs- und Ausgangsneuronen erzielt.

6.2. Anwendungsmöglichkeiten

Anwendung könnte diese Diplomarbeit überall dort finden, wo verschiedene Gegenstände voneinander unterschieden werden müssen. Man stelle sich als Beispiel ein Zulieferband vor, welches verschiedene Objekte in den Bilderfassungsbereich bringt. Hier werden dann die Objekte erkannt und anschließend vom Roboter sortiert.

6.3. Anregungen für weitere Untersuchungen

Nachteilig ist, daß wir beim Einlernen von Gegenständen die Greifhöhe angeben und abspeichern müssen. Bei Einsatz einer zweiten Kamera, mit Koppelung über einen Kameramultiplexer, wäre eine dreidimensionale Bilderfassung und -verarbeitung möglich, so daß hier die Greifhöhe nicht mehr angegeben werden muß.

Man sollte auch in Erwägung ziehen, ob nicht der Einsatz eines anderen Bildverarbeitungssystems sinnvoll ist, besonders unter der Berücksichtigung, daß moderne 'Framegrabber-Karten' heute schon relativ preisgünstig angeboten werden. Dies hätte dann den Vorteil, daß man das Bild im PC-Speicher direkt zur Verfügung hat und hier auch eigene

Algorithmen zur Bildvorverarbeitung schreiben könnte. Hier ist dann auch eine Farberkennung denkbar. Die Geschwindigkeit der Vorverarbeitung würde sich wesentlich erhöhen, was im Hinblick auf eine Echtzeitbildverarbeitung von Bedeutung ist.

7. Programmlisting

7.1. Das Projekt-File NEUROROB.PRJ

Zum Projekt-File gehören folgende Files:

NEUROROB.C
N_ABOUT.C
N_BRAIN.C
N_FILE.C
N_LIST.C
N_MASTER.C
N_POSITI.C
N_ROBINI.C
N_VISUAL.C
N_STEU.C
N_VIDEOI.C
NEUROROB.DEF
NEUROROB.RC

7.2. Die Moduldefinitionsdatei NEUROROB.DEF

```
;/*************************************************************************
; Modul:       NEUROROB.DEF

; Aufgabe:     Enthält grundlegende Informationen über Programmgeneration

;*************************************************************************/

NAME          NeuroRob

DESCRIPTION   'NeuroRob'

EXETYPE       WINDOWS

CODE          PRELOAD FIXED
DATA          PRELOAD FIXED

HEAPSIZE      4096
STACKSIZE     8192

EXPORTS
  MainWndProc       @1;
  About             @2;
  Steuerung         @3;
  Befehlsliste      @4;
  VideoSteuerung    @5;
  Befehlseingabe    @6;
  Brain             @7;
  NeuesNetz         @8 ;
  DateiLaden        @9;
  Positionsana      @10;
  NeuronenDaten     @11;
  TeileDaten        @12;
  Visualisierung    @13 ;
  Face              @14;
  RobInit           @15;
```

7.3. Die Header-Dateien

7.3.1. NEUROROB.H

```
#define NEUROROBPFAD "D:\\DIPLOM\\"
#define VERSION "Neurorob Version 1.0"


//  Funktionsprototypen

int PASCAL      WinMain(HANDLE, HANDLE, LPSTR, int);
BOOL            InitApplication(HANDLE);
BOOL            InitInstance(HANDLE, int);
long FAR PASCAL Brain(HWND, unsigned, WORD, LONG);
long FAR PASCAL MainWndProc(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL About(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL Befehlsliste(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL Steuerung(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL Visualisierung(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL NeuesNetz(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL RobInit(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL VideoSteuerung(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL DateiLaden (HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL Positionsana (HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL NeuronenDaten (HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL TeileDaten (HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL Face (HWND, unsigned, WORD, LONG);
int             Sende(HANDLE,char *);
int             SendeComm1(HANDLE,char *);
int             SendeComm2(HANDLE,char *);
void            Gelenkdemo(void);
void            HoleVideomatZeile(LPSTR);
void            ZeichneBrain(HWND);
void            Anzeige(int);
BOOL            LadeGrundposition(VOID);
VOID            HolePartSchwerpunkt(VOID);
void            AbsolutRelativ(void);


//************************************************************

void            VerknuepfeNetz(void);
double          Sigmoid(double);
double          SigmoidStrich(double);
void            BerechneAusgang(void);
void            Backpropagation(void);
void            SetzeAusgangsVektor(int);
void            SetzeEingangsVektor(int);
void            HoleEingangsVektor(void);
void            SetzeEingangsTrainingsVektor(int);
int             NeuronenMaximum(void);
void            ErstelleMinMaxVektoren(void);
double          MaxOfTraining(int);
double          MinOfTraining(int,double);
void            TransformiereEingangsVektor(void);
BOOL            LadeNeuronalesNetz(void);
BOOL            SpeicherNeuronalesNetz(void);
```

```
//************************************************************

void MO(LPSTR Param);      // Funktions_Prototypen für
void DS(LPSTR Param);
void DW(LPSTR Param);      // MOVEMASTER-BEFEHLE
void GC(void);
void GP(LPSTR Param);

void GO(void);
void HE(LPSTR Param);
void MA(LPSTR Param);
void MC(LPSTR Param);
void MJ(LPSTR Param);

void MP(LPSTR Param);
void MS(LPSTR Param);
void MT(LPSTR Param);
void NT(void);
void NW(void);
void PD(LPSTR Param);
void PR(LPSTR Param);
void RS(LPSTR Param);
void SD(LPSTR Param);
void SP(LPSTR Param);

//*****************************************************************
```

7.3.2. NEUROMEN.H

```
/* C:\DIPLOM\NEUROMEN.H 12/11/1991 9:48*/
#define  IDM_INFO 100
#define  IDM_BEENDEN  105
#define  IDM_ROBINIT  106
#define  IDM_STEUERUNG  107
#define  IDM_BEFEHLSLISTE 108
#define  IDM_SENDEFILE  109
#define  IDM_VIDEOINIT  110
#define  IDM_NEUINIT  112
#define  IDM_GELENKDEMO 117
#define  IDM_HANOI  118
#define  IDM_POSITIONSANA 120
#define  IDM_VISUALISIERUNG 119
#define  IDM_VIDEOSTEU  111
```

7.3.3. N_EXTERN.H

```
extern  HMENU     hBrainMenu;
extern  HMENU     hMainMenu;
extern  BOOL      bMoveMasterInit;
extern  BOOL      bVideomatInit;
extern  BOOL      bBrain;
extern  BOOL      bBefehlsliste;
extern  BOOL      bVideoSteuerung;
extern  BOOL      bVisualisierung;
extern  BOOL      bSteuerung;
extern  BOOL      bRepeatmodus;
extern  BOOL      bEinzelschrittmodus;
```

```
extern  BOOL      bNetzdefine;
extern  BOOL      bSpeichernAbfrage;
extern  BOOL      bSpeichern;
extern  HWND      hWndNeurorob;
extern  HWND      hWndBrain;
extern  HWND      hDlgBefehlsliste;
extern  HWND      hDlgVisualisierung;
extern  HWND      hDlgSteuerung;
extern  HWND      hDlgVideoSteuerung;
extern  FARPROC   lpBefehlsliste;
extern  FARPROC   lpVisualisierung;
extern  FARPROC   lpSteuerung;
extern  FARPROC   lpVideoSteuerung;
extern  FARPROC   lpProcLaden;
extern  DCB       CommDCB;
extern  int       nCid;
extern  int       nCid2;
extern  char      dummy[];
extern  char      dummy1[];
extern  char      dummy2[];
extern  int       ndummy;
extern  HANDLE    hInst;
extern  HBITMAP   hRobBitmap,
                  hOldBitmap,
                  hFaceBitmap,
                  hAllBitmap;
extern  HCURSOR   hHourGlass;
extern  HCURSOR   hSivips;
extern  HCURSOR   hMaster;
extern  int       roboterspeed;
extern  int       roboterstep;
extern  int       robotergrid;
extern  int       roboterdegree;
//      Folgende globale Variable kommen von OpenDlg (SDK-Exsample)

extern  char FileName[];
extern  char PathName[];
extern  char OpenName[];
extern  char DefPath[];
extern  char DefSpec[];
extern  char DefExt[];
extern  char str[];


extern  LPSTR     szFileCaption;

extern  HPEN hRotPen1;
extern  HPEN hBlauPen1;
extern  HPEN hGruenPen1;
extern  HPEN hLilaPen1;
extern  HPEN hSchwarzPen1;
extern  HPEN hGelbPen1;
extern  HPEN hHellblauPen1;
extern  HPEN hGrauPen1;

extern  HBITMAP hGehirnBitmap;

extern  HBRUSH hRotBrush1;
extern  HBRUSH hBlauBrush1;
extern  HBRUSH hGruenBrush1;
extern  HBRUSH hLilaBrush1;
extern  HBRUSH hSchwarzBrush1;
extern  HBRUSH hGelbBrush1;
```

```
extern  HBRUSH hHellblauBrush1;
extern  HBRUSH hGrauBrush1;

extern  double xPart;
extern  double yPart;
extern  double xCent;
extern  double yCent;
extern  double xUrsprung;
extern  double yUrsprung;
```

7.3.4. BRAINMEN.H

```
/* D:\DIPLOM\BRAINMEN.H 12/31/1991 13:59*/
#define  IDM9_NEU 901
#define  IDM9_LADEN 902
#define  IDM9_SPEICHERN 903
#define  IDM9_ALS 904
#define  IDM9_ENDE  905
#define  IDM9_PRIOHIGH  911
#define  IDM9_PRIOMIDDLE  912
#define  IDM9_PRIOLOW 913
#define  IDM9_START 915
#define  IDM9_GROSS 916
#define  IDM9_MITTEL  917
#define  IDM9_KLEIN 918
#define  IDM9_ANZEIGEN  919
#define  IDM9_TEST  906
#define  IDM9_ERKENNUNG 920
#define  IDM9_RESET 907
#define  IDM9_ROBOTEREINSATZ  921
#define  IDM9_EDITTEILEDATEN  922
```

7.3.5. NDL_ABO.H

```
/* D:\DIPLOM\NDL_ABO.H 11/13/1991 9:27*/
#define  ID12_SHOW  1201
```

7.3.6. NDL_EDIT.H

```
/* D:\DIPLOM\NDL_EDIT.H 10/17/1991 17:16*/
#define  ID7_EDIT 701
```

7.3.7. NDL_FILE.H

```
/* D:\DIPLOM\NDL_FILE.H 1/3/1980 9:23*/
#define  ID6_DIRECTORY  602
#define  ID6_FILES  600
#define  ID6_DIRECTORIES  601
#define  ID6_FILE 603
```

7.3.8. NDL_LIST.H

```
/* D:\DIPLOM\NDL_LIST.H 12/28/1991 16:09*/
#define  ID3_BEFEHLSLISTE 300
#define  ID3_ADD  301
#define  ID3_DEL  302
#define  ID3_LOAD 303
#define  ID3_SAVE 304
#define  ID3_DO 306
#define  ID3_SINGLESTEP 308
#define  ID3_REPEAT 309
#define  ID3_SELECTALL  310
#define  ID3_COMMAND  311
#define  ID3_UNSELECTALL  307
```

7.3.9. NDL_NDAT.H

```
/* D:\DIPLOM\NDL_NDAT.H 11/12/1991 10:56*/
#define  ID13_POTENTIAL 1301
#define  ID13_AUSGANG 1302
#define  ID13_WTEXT 1303
#define  ID13_WEDIT 1304
#define  ID13_SCROLLBAR 1305
```

7.3.10. NDL_NNET.H

```
/* D:\DIPLOM\NDL_NNET.H 12/19/1991 9:31*/
#define  ID11_S0  1150
#define  ID11_S1  1151
#define  ID11_S2  1152
#define  ID11_S3  1153
#define  ID11_T0  1160
#define  ID11_T1  1161
#define  ID11_T2  1162
#define  ID11_T3  1163
#define  ID11_TOTALAREA 1130
#define  ID11_TEILE 1101
#define  ID11_LAYER 1102
#define  ID11_TEILETEXT 1104
#define  ID11_LAYERTEXT 1105
#define  ID11_MAJOR 1131
#define  ID11_MINOR 1132
#define  ID11_NHOLES  1133
#define  ID11_PERIMETER 1134
#define  ID11_RMAX  1135
#define  ID11_RMIN  1136
#define  ID11_AVRAD 1137
#define  ID11_LAENGE  1138
#define  ID11_BREITE  1139
#define  ID11_TEILEDATEN  1106
#define  ID11_SAMPLETEXT  1107
#define  ID11_SAMPLE  1108
#define  ID11_GRAUWERTE 1140
#define  ID11_S4  1154
#define  ID11_S5  1155
#define  ID11_T4  1164
#define  ID11_T5  1165
#define  ID11_ROBOTER_KEIN  1110
#define  ID11_ROBOTER_TRAINING  1111
#define  ID11_ROBOTER_KOMPLETT  1112
```

7.3.11. NDL_POS.H

```
/* D:\DIPLOM\NDL_POS.H 10/30/1991 8:53*/
#define  ID8_LOAD 810
#define  ID8_SAVE 811
#define  ID8_NEW  812
#define  ID8_DATEN  813
#define  ID8_MOVE 814
#define  ID8_POSITION 816
#define  ID8_POSITIONSNR  830
#define  ID8_XWERT  831
#define  ID8_YWERT  832
#define  ID8_ZWERT  833
#define  ID8_HANDPITCH  834
#define  ID8_HANDROLL 835
#define  ID8_HANDGRIP 836
#define  ID8_ROBOTER  837
```

7.3.12. NDL_STEU.H

```
/* D:\DIPLOM\NDL_STEU.H 12/2/1991 11:01*/
#define  ID2_HAND_OPEN  200
#define  ID2_HAND_CLOSE 201
#define  ID2_HAND_RIGHT 202
#define  ID2_HAND_LEFT  203
#define  ID2_HAND_UP  204
#define  ID2_HAND_DOWN  205
#define  ID2_ELBOW_P  206
#define  ID2_ELBOW_N  207
#define  ID2_WAIST_P  208
#define  ID2_WAIST_N  209
#define  ID2_SHOULDER_P 210
#define  ID2_SHOULDER_N 211
#define  ID2_HOCH 212
#define  ID2_RUNTER 213
#define  ID2_LINKS  214
#define  ID2_RECHTS 215
#define  ID2_VOR  216
#define  ID2_ZURUECK  217
#define  ID2_GRID 218
#define  ID2_STEP 219
#define  ID2_SPEED  220
#define  ID2_TEXTGRID 221
#define  ID2_TEXTSPEED  226
#define  ID2_MO_999 224
#define  ID2_ADD  225
#define  ID2_POSEDIT  227
#define  ID2_DEFINIERN  228
#define  ID2_TEXTMM 222
#define  ID2_TEXTGRAD 230
#define  ID2_DEGREE 229
```

7.3.13. NDL_TDAT.H

```
/* D:\DIPLOM\NDL_TDAT.H 1/21/1992 17:41*/
#define  ID15_NAME  1503
```

```
#define  ID15_GROESSE 1504
#define  ID15_SAMPLENUMMER  1507
#define  ID15_SCROLLTEILE 1501
#define  ID15_TEILENUMMER 1502
#define  ID15_SAMPLEAUFNEHMEN 1508
#define  ID15_SCROLLSAMPLE  1506
#define  ID15_FACHNUMMER  1509
#define  ID15_STARTEROBOTER 1511
#define  ID15_GREIFHOEHE  1512
```

7.3.14. NDL_ROB.H

```
/* D:\DIPLOM\NDL_ROB.H 11/18/1991 15:00*/
#define  ID19_NT  1900
#define  ID19_TEXT1 1901
#define  ID19_TEXT2 1902
```

7.3.15. NDL_VID.H

```
/* D:\DIPLOM\NDL_VID.H 12/28/1991 15:39*/
#define  ID21_PLUS  2101
#define  ID21_MINUS 2102
#define  ID21_LIVEG 2103
#define  ID21_LIVEB 2104
#define  ID21_GRAU  2105
#define  ID21_BINAER  2106
#define  ID21_HISTOGRAMM  2107
#define  ID21_ORIENTIERUNG  2108
```

7.3.16. NDL_VISU.H

```
/* D:\DIPLOM\NDL_VISU.H 11/7/1991 12:05*/
#define  ID10_AKTUELL 1003
#define  ID10_DATEN 1001
```

7.4. Der C-Quellcode

7.4.1. NEUROROB.C

```
/*****************************************************************************
 Modul:       NEUROROB.C

 Aufgabe:     Hauptteil des Programmes. Dieses Modul übernimmt die Anmeldung
              von NeuroRob bei WINDOWS. Sowie die Steuerung des Hauptfensters
              und des Hauptmenues.

 Funktionen:  WinMain
              InitApplication
              InitInstance
              MainWndProc
*****************************************************************************/

#include <windows.h>
#include "neurorob.h"
#include "brainmen.h"
#include "neuromen.h"
#include <stdio.h>

DCB       CommDCB;

HANDLE    nCid;
HANDLE    nCid2;
HANDLE    hInst;

char      dummy[100];
char      dummy1[100];
char      dummy2[100];

HMENU     hMainMenu;
HMENU     hBrainMenu;

BOOL      bMoveMasterInit     = FALSE;
BOOL      bVideomatInit       = FALSE;
BOOL      bBrain              = FALSE;
BOOL      bBefehlsliste       = FALSE;
BOOL      bSteuerung          = FALSE;
BOOL      bVideoSteuerung     = FALSE;
BOOL      bRepeatmodus        = FALSE;
BOOL      bEinzelschrittmodus = FALSE;
BOOL      bVisualisierung     = FALSE;
BOOL      bNetzdefine         = FALSE;
BOOL      bSpeichernAbfrage   = TRUE;
BOOL          bSpeichern      = FALSE;  //MenueItem Speichern erlaubt ?

HWND      hWndBrain;
HWND      hWndNeurorob;
HWND      hDlgBefehlsliste;
HWND      hDlgSteuerung;
HWND      hDlgVisualisierung;
HWND      hDlgVideoSteuerung;

FARPROC   lpBefehlsliste;
FARPROC   lpSteuerung;
FARPROC   lpProcLaden;
```

```
FARPROC   lpVisualisierung;
FARPROC   lpVideoSteuerung;
LPSTR     szFileCaption;

HBITMAP   hRobBitmap;
HBITMAP   hOldBitmap;
HBITMAP   hAllBitmap;
HBITMAP   hFaceBitmap;
HBITMAP   hGehirnBitmap;

HPEN      hRotPen1;
HPEN      hBlauPen1;
HPEN      hLilaPen1;
HPEN      hSchwarzPen1;
HPEN      hGelbPen1;
HPEN      hHellblauPen1;
HPEN      hGruenPen1;
HPEN      hGrauPen1;

HBRUSH    hRotBrush1;
HBRUSH    hBlauBrush1;
HBRUSH    hLilaBrush1;
HBRUSH    hGruenBrush1;
HBRUSH    hSchwarzBrush1;
HBRUSH    hGelbBrush1;
HBRUSH    hHellblauBrush1;
HBRUSH    hGrauBrush1;

HCURSOR   hHourGlass;
HCURSOR   hMaster;
HCURSOR   hSivips;


int       roboterspeed = 20;
int       roboterstep  = 20;
int       robotergrid  = 63;
int       roboterdegree= 10;
int       ndummy;

double    xCent=394;
double    yCent=25;
double    xUrsprung;
double    yUrsprung;
double    xPart;
double    yPart;

char      FileName[128];
char      PathName[128];
char      OpenName[128];
char      DefSpec[13]="*.*";
char      DefExt[]=".txt";
char      str[255];

/*****************************************************************************
 Funktion:     WinMain

 Aufgabe:      Hauptfunktion

 Parameter:
  HANDLE hInstance    :Aktuelle Instance
  HANDLE hPrevInstance:Vorherige Instance
  LPSTR  lpCmdLine    :Zeiger auf String der bei Aufruf hinter dem Programm-
                       namen stand
```

```
  int    nCmdShpw     :

 Rückgabewert:
  int          Der wParam der zuletzt empfangenen Meldung
*************************************************************************/

int PASCAL WinMain( HANDLE  hInstance,
                    HANDLE  hPrevInstance,
                    LPSTR   lpCmdLine,
                    int     nCmdShow)
{
MSG     msg;
HCURSOR hSaveCursor;

  if (!hPrevInstance)
  {
    if (!InitApplication(hInstance))
      return (FALSE);
  }
  else
  {
    MessageBox(GetFocus(),(LPSTR)"NeuroRob ist bereits gstartet worden",NULL,
      MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
    return FALSE;
  }
  if (!InitInstance(hInstance, nCmdShow))
    return (FALSE);
  while (GetMessage(&msg,NULL,NULL,NULL))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return (msg.wParam);
}//WinMain

/*************************************************************************
 Funktion:     InitApplication

 Aufgabe:      Grundinitialisierung

 Parameter:    HANDLE hInstance : Handle der zu initialisierenden Instance

 Rückgabewert: BOOL     wenn TRUE dann war Initialisierung erfolgreich
*************************************************************************/

BOOL InitApplication(HANDLE hInstance)
{
WNDCLASS  wc;
WNDCLASS  wc2;

  wc.style          = NULL;
  wc.lpfnWndProc    = MainWndProc;
  wc.cbClsExtra     = 0;
  wc.cbWndExtra     = 0;
  wc.hInstance      = hInstance;
  wc.hIcon          = LoadIcon(hInstance, "ROB");
  wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground  = GetStockObject(WHITE_BRUSH);
  wc.lpszMenuName   = "NeuroRobMenu";
  wc.lpszClassName  = "NeurorobWClass";
  if(RegisterClass(&wc)==FALSE)
    return(FALSE);
  wc2.style = CS_VREDRAW | CS_HREDRAW | CS_CLASSDC | CS_OWNDC | CS_DBLCLKS;
```

```
  wc2.lpfnWndProc   = Brain;
  wc2.cbClsExtra    = 0;
  wc2.cbWndExtra    = 0;
  wc2.hInstance     = hInstance;
  wc2.hIcon         = LoadIcon(hInstance, "GEHIRN");
  wc2.hCursor       = LoadCursor(NULL, IDC_ARROW);
  wc2.hbrBackground = GetStockObject(WHITE_BRUSH);
  wc2.lpszMenuName  =  "BRAINMENU";
  wc2.lpszClassName = "BrainWClass";
  return(RegisterClass(&wc2));
}//InitApplication

/*************************************************************************
 Funktion:      InitInstance

 Aufgabe:       Instancespecifische Initialisierung

 Parameter:
  HANDLE hInstance : Handle der zu initialisierenden Instance
  int    nCmdShow  : Darstellungsart des Fensters

 Rückgabewert: BOOL   TRUE wenn Initialisierung erfolgreich war
*************************************************************************/

BOOL InitInstance(HANDLE  hInstance,
                  int     nCmdShow)
{
HWND hWnd;

  hInst = hInstance;
  hWnd = CreateWindow("NeurorobWClass","NeuroRob",WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,
/* Use the window class menu.         */
    hInstance,NULL);
  hWndNeurorob=hWnd;
  hWndBrain = CreateWindow((LPSTR)"BrainWClass",(LPSTR)"Brain",
    WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,
    CW_USEDEFAULT,NULL,NULL,hInst,NULL);
  if (!hWnd || !hWndBrain)
    return (FALSE);
  hMainMenu = GetMenu(hWnd);
  hBrainMenu= GetMenu(hWndBrain);
  CheckMenuItem(hBrainMenu,IDM9_MITTEL,MF_CHECKED);
  CheckMenuItem(hBrainMenu,IDM9_PRIOMIDDLE,MF_CHECKED);
  EnableMenuItem(hBrainMenu,IDM9_START,MF_GRAYED);
  EnableMenuItem(hBrainMenu,IDM9_SPEICHERN,MF_GRAYED);
  EnableMenuItem(hBrainMenu,IDM9_ALS,MF_GRAYED);
  EnableMenuItem(hBrainMenu,IDM9_ROBOTEREINSATZ,MF_GRAYED);
  EnableMenuItem(hBrainMenu,IDM9_ERKENNUNG,MF_GRAYED);
  EnableMenuItem(hBrainMenu,IDM9_TEST,MF_GRAYED);
  EnableMenuItem(hBrainMenu,IDM9_RESET,MF_GRAYED);
  EnableMenuItem(hMainMenu,IDM_STEUERUNG,MF_GRAYED);
  EnableMenuItem(hMainMenu,IDM_BEFEHLSLISTE,MF_GRAYED);
  EnableMenuItem(hMainMenu,IDM_SENDEFILE,MF_GRAYED);
  EnableMenuItem(hMainMenu,IDM_POSITIONSANA,MF_GRAYED);
  EnableMenuItem(hMainMenu,IDM_GELENKDEMO,MF_GRAYED);
  EnableMenuItem(hMainMenu,IDM_VISUALISIERUNG,MF_GRAYED);
//  EnableMenuItem(hMainMenu,IDM_NEUINIT,MF_GRAYED);
  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);
  return (TRUE);
}//InitInstance
```

```
/*****************************************************************************
 Funktion:      MainWndProc

 Aufgabe:       HauptCallBackFunktion von Windows

 Parameter:
  HWND      hWnd   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenspezifische Informationen
  LONG      lParam :          "                 "

 Rückgabewert:
  long    0L wenn Nachricht bearbeitet sonst Defaultwert von DefWindowProc
*****************************************************************************/

long FAR PASCAL MainWndProc(HWND      hWnd,
                            unsigned  message,
                            WORD      wParam,
                            LONG      lParam)
{
FILE        *FilePointer;
FARPROC     lpProcAbout;
FARPROC     lpProcRobInit;
FARPROC     lpProcVideoinit;
FARPROC     lpProcPosition;
HDC         hDC;
HDC         hMemoryDC;
PAINTSTRUCT ps;
char        string[6];
int         Return;
HCURSOR     hSaveCursor;
char        where[30];
RECT        *lpRect;

  switch (message) {
    case WM_COMMAND:
         switch (wParam) {
           case IDM_INFO:
                lpProcAbout = MakeProcInstance(About, hInst);
                DialogBox(hInst,"AboutBox",hWnd,lpProcAbout);
                FreeProcInstance(lpProcAbout);
                break;
           case IDM_SENDEFILE:
                szFileCaption = (LPSTR)"Textfile laden";
                lpProcLaden = MakeProcInstance(DateiLaden, hInst);
                if(!DialogBox(hInst,"FILEOPERATION",hWnd,lpProcLaden))
                  break;
                FreeProcInstance(lpProcLaden) ;
                FilePointer=fopen(FileName,"rt");
                if(!FilePointer)
                {
                  MessageBox(GetFocus(),"Datei konnte nicht geoeffnet werden",
                    "Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
                  return (FALSE);
                }
                ndummy=0;
                do
                {
                  if (fgets(dummy, 90 , FilePointer) == NULL)
                    break;
                  dummy[strlen(dummy)-1] = '\0'; // Linefeed entfernen
                  ++ndummy;
                  lstrcat((LPSTR)dummy,(LPSTR)"\n");
```

```
        SendeComm1(nCid,dummy);
      } while (TRUE);
      fclose(FilePointer);
      break;
case IDM_STEUERUNG:
      if(!bSteuerung)
      {
        lpSteuerung = MakeProcInstance(Steuerung,hInst);
        hDlgSteuerung = CreateDialog(hInst,"STEUERUNG",
          hWnd,lpSteuerung);
        bSteuerung = TRUE;
        CheckMenuItem(hMainMenu,IDM_STEUERUNG,MF_CHECKED);
      }
      else
      {
        DestroyWindow(hDlgSteuerung);
        FreeProcInstance(lpSteuerung);
        bSteuerung = FALSE;
        hDlgSteuerung = NULL;
        CheckMenuItem(hMainMenu,IDM_STEUERUNG,MF_UNCHECKED);
      }
      break;
case IDM_ROBINIT:
      lpProcRobInit = MakeProcInstance(RobInit, hInst);
      DialogBox(hInst,"ROBINIT",hWnd,lpProcRobInit);
      FreeProcInstance(lpProcRobInit);
      break;
case IDM_BEFEHLSLISTE:
      if(!bBefehlsliste)
      {
        lpBefehlsliste = MakeProcInstance(Befehlsliste,hInst);
        hDlgBefehlsliste = CreateDialog(hInst,"BEFEHLSLISTE",
          hWnd,lpBefehlsliste);
        bBefehlsliste = TRUE;
        CheckMenuItem(hMainMenu,IDM_BEFEHLSLISTE,MF_CHECKED);
      }
      else
      {
        DestroyWindow(hDlgBefehlsliste);
        FreeProcInstance(lpBefehlsliste);
        bBefehlsliste = FALSE;
        hDlgBefehlsliste = NULL;
        CheckMenuItem(hMainMenu,IDM_BEFEHLSLISTE,MF_UNCHECKED);
      }
      break;
case IDM_VISUALISIERUNG:
      if(!bVisualisierung)
      {
        lpVisualisierung = MakeProcInstance(Visualisierung,hInst);
        hDlgVisualisierung = CreateDialog(hInst,"VISUALISIERUNG",
          hWnd,lpVisualisierung);
        bVisualisierung = TRUE;
        CheckMenuItem(hMainMenu,IDM_VISUALISIERUNG,MF_CHECKED);
      }
      else
      {
        DestroyWindow(hDlgVisualisierung);
        FreeProcInstance(lpVisualisierung);
        bVisualisierung = FALSE;
        hDlgVisualisierung = NULL;
        CheckMenuItem(hMainMenu,IDM_VISUALISIERUNG,MF_UNCHECKED);
      }
      break;
```

```
case IDM_VIDEOINIT:
     hSaveCursor = SetCursor(hSivips);
     if(!VideoInit())
     {
       SetCursor(hSaveCursor);
       MessageBox(GetFocus(),
         (LPSTR)"COM 2 konnte nicht initialisiert werden",NULL,
         MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
       break;
     }
     SetCursor(hSaveCursor);
     bVideomatInit=TRUE;
     MessageBox(GetFocus(),(LPSTR)
       "Bitte legen Sie den Abgleichgegenstand\n auf den Ursprung",
       (LPSTR)" ",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
     HolePartSchwerpunkt();
     xUrsprung=xCent+xPart;
     yUrsprung=yCent+yPart;
     SendeComm2(nCid2,"H");
     SendeComm2(nCid2,"B");
     break;
case IDM_VIDEOSTEU:
     if(!bVideoSteuerung)
     {
       lpVideoSteuerung = MakeProcInstance(VideoSteuerung,hInst);
       hDlgVideoSteuerung = CreateDialog(hInst,"VIDEOSTEUERUNG",
         hWnd,lpVideoSteuerung);
       bVideoSteuerung = TRUE;
       CheckMenuItem(hMainMenu,IDM_VIDEOSTEU,MF_CHECKED);
     }
     else
     {
       DestroyWindow(hDlgVideoSteuerung);
       FreeProcInstance(lpVideoSteuerung);
       bVideoSteuerung = FALSE;
       hDlgVideoSteuerung = NULL;
       CheckMenuItem(hMainMenu,IDM_VIDEOSTEU,MF_UNCHECKED);
     }
     break;
case IDM_NEUINIT:
     if(!bBrain)
     {
       ShowWindow(hWndBrain,SW_SHOWNORMAL);
       UpdateWindow(hWndBrain);
       bBrain = TRUE;
       CheckMenuItem(hMainMenu,IDM_NEUINIT,MF_CHECKED);
     }
     else
     {
       ShowWindow(hWndBrain,SW_HIDE);
       bBrain = FALSE;
       CheckMenuItem(hMainMenu,IDM_NEUINIT,MF_UNCHECKED);
     }
     break;
case IDM_GELENKDEMO:
     MessageBox(GetFocus(),(LPSTR)"Wir starten mit der DEMO",
     (LPSTR)"DEMO",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
     Gelenkdemo();
     MessageBox(GetFocus(),(LPSTR)"PUUHH.. Fertig",(LPSTR)"DEMO",
       MB_OK|MB_SYSTEMMODAL);
     break;
case IDM_BEENDEN:
     PostMessage(hWnd, WM_DESTROY,0,0);
```

```
        case IDM_POSITIONSANA:
             lpProcPosition = MakeProcInstance(Positionsana,hInst);
             DialogBox(hInst,"POSITIONSANA",hWnd,lpProcPosition);
             FreeProcInstance(lpProcPosition);
             break;
        default:
             return (DefWindowProc(hWnd, message, wParam, lParam));
      }
case WM_INITMENU:
      EnableMenuItem(hMainMenu,IDM_ROBINIT,        MF_GRAYED);
      EnableMenuItem(hMainMenu,IDM_STEUERUNG,      MF_GRAYED);
      EnableMenuItem(hMainMenu,IDM_BEFEHLSLISTE,   MF_GRAYED);
      EnableMenuItem(hMainMenu,IDM_SENDEFILE,      MF_GRAYED);
      EnableMenuItem(hMainMenu,IDM_POSITIONSANA,   MF_GRAYED);
      EnableMenuItem(hMainMenu,IDM_GELENKDEMO,     MF_GRAYED);
      EnableMenuItem(hMainMenu,IDM_VISUALISIERUNG, MF_GRAYED);
      EnableMenuItem(hMainMenu,IDM_VIDEOSTEU,      MF_GRAYED);
      EnableMenuItem(hMainMenu,IDM_VIDEOINIT,      MF_GRAYED);
      if(bMoveMasterInit)
      {
        EnableMenuItem(hMainMenu,IDM_STEUERUNG,      MF_ENABLED);
        EnableMenuItem(hMainMenu,IDM_BEFEHLSLISTE,   MF_ENABLED);
        EnableMenuItem(hMainMenu,IDM_SENDEFILE,      MF_ENABLED);
        EnableMenuItem(hMainMenu,IDM_POSITIONSANA,   MF_ENABLED);
        EnableMenuItem(hMainMenu,IDM_GELENKDEMO,     MF_ENABLED);
        EnableMenuItem(hMainMenu,IDM_VISUALISIERUNG, MF_ENABLED);
      }
      else
       EnableMenuItem(hMainMenu,IDM_ROBINIT,MF_ENABLED);
      if(bVideomatInit)
        EnableMenuItem(hMainMenu,IDM_VIDEOSTEU,MF_ENABLED);
      else
        EnableMenuItem(hMainMenu,IDM_VIDEOINIT,MF_ENABLED);
      break;
case WM_CREATE:
      hRobBitmap = LoadBitmap(hInst,"rob");
      hAllBitmap = LoadBitmap(hInst,"all");
      hHourGlass = LoadCursor(NULL,IDC_WAIT);
      hMaster    = LoadCursor(hInst,"MASTER");
      hSivips    = LoadCursor(hInst,"SIVIPS");
      hRotPen1 = CreatePen(PS_SOLID,1,RGB(255,0,0));
      hBlauPen1 = CreatePen(PS_SOLID,1,RGB(0,255,0));
      hGruenPen1 = CreatePen(PS_SOLID,1,RGB(0,0,255));
      hLilaPen1 = CreatePen(PS_SOLID,1,RGB(154,38,217));
      hSchwarzPen1 = CreatePen(PS_SOLID,1,RGB(128,128,128));
      hGelbPen1 = CreatePen(PS_SOLID,1,RGB(205,219,36));
      hHellblauPen1 = CreatePen(PS_SOLID,1,RGB(28,218,227));
      hGrauPen1 = CreatePen(PS_SOLID,1,RGB(173,100,82));
      hRotBrush1 = CreateSolidBrush(RGB(255,0,0));
      hBlauBrush1 = CreateSolidBrush(RGB(0,0,255));
      hGruenBrush1 = CreateSolidBrush(RGB(0,255,0));
      hLilaBrush1 = CreateSolidBrush(RGB(154,38,217));
      hSchwarzBrush1 = CreateSolidBrush(RGB(128,128,128));
      hGelbBrush1 = CreateSolidBrush(RGB(205,219,36));
      hHellblauBrush1 = CreateSolidBrush(RGB(28,218,227));
      hGrauBrush1 = CreateSolidBrush(RGB(173,100,82));
      break;
case WM_PAINT:
      hDC= BeginPaint(hWnd,&ps);
      hMemoryDC = CreateCompatibleDC(hDC);
      hOldBitmap = SelectObject(hMemoryDC,hAllBitmap);
      if(hOldBitmap)
      {
```

```
          BitBlt (hDC,0,0,507,308,hMemoryDC,0,0,SRCCOPY);
          SelectObject(hMemoryDC,hOldBitmap);
        }
        DeleteDC(hMemoryDC);
        EndPaint(hWnd,&ps);
        break;
    case WM_DESTROY:
        if(bNetzdefine&&bSpeichernAbfrage)
        {
          if(MessageBox(GetFocus(),
             (LPSTR)"Moechten Sie das neuronale Netz speichern",(LPSTR)"",
               MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL)==IDYES)
          {
            if(bSpeichern)
              SendMessage(hWndBrain,WM_COMMAND,IDM9_SPEICHERN,0L);
            else
              SendMessage(hWndBrain,WM_COMMAND,IDM9_ALS,0L);
          }
        }
        DestroyWindow(hWndBrain);
        DeleteObject(hRobBitmap);
        DeleteObject(hAllBitmap);
        DeleteObject(hRotPen1);
        DeleteObject(hBlauPen1);
        DeleteObject(hLilaPen1);
        DeleteObject(hSchwarzPen1);
        DeleteObject(hGelbPen1);
        DeleteObject(hHellblauPen1);
        DeleteObject(hGruenPen1);
        DeleteObject(hGrauPen1);
        DeleteObject(hRotBrush1);
        DeleteObject(hBlauBrush1);
        DeleteObject(hLilaBrush1);
        DeleteObject(hSchwarzBrush1);
        DeleteObject(hGelbBrush1);
        DeleteObject(hHellblauBrush1);
        DeleteObject(hGruenBrush1);
        DeleteObject(hGrauBrush1);
        if(bMoveMasterInit)
        {
          SendeComm1(nCid,"PD 999,0,400,200,-90,90,O\n");
          SendeComm1(nCid,"PD 1,153.6,-232.9,813.4,-10.2,179.9,O\n");
          SendeComm1(nCid,"SP 18 \n");
          SendeComm1(nCid,"MO 1\n");
        }
        CloseComm(nCid);
        CloseComm(nCid2);
        PostQuitMessage(0);
        break;
    default:
        return (DefWindowProc(hWnd, message, wParam, lParam));
  }
  return (NULL);
}//MainWndProc

void Anzeige(int Wert)
{
char  dooferdummy[20];
  itoa(Wert,dooferdummy,10);
  MessageBox(GetFocus(),(LPSTR)dooferdummy,(LPSTR)"Anzeige",MB_OK);
}//Anzeige
```

7.4.2. N_ABOUT.C

```
/*****************************************************************************
 Modul:         N_ABOUT.C

 Aufgabe:       Darstellung der Indormationsdialogbox

 Funktionen:    About
*****************************************************************************/

#include "windows.h"
#include "neurorob.h"
#include "n_extern.h"
#include "ndl_abo.h"

FARPROC  lpProcFace;


/*****************************************************************************
 Funktion:      About

 Aufgabe:       Informationsdialogbox darstellen

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*****************************************************************************/

BOOL FAR PASCAL About(HWND      hDlg,
                      unsigned  message,
                      WORD      wParam,
                      LONG      lParam)
{
  switch (message){
    case WM_INITDIALOG:
         return (TRUE);
    case WM_COMMAND:
         switch(wParam){
               case IDOK:
               case IDCANCEL:
                    EndDialog(hDlg, TRUE);
                    return(TRUE);
               case ID12_SHOW:
                    lpProcFace = MakeProcInstance(Face, hInst);
                    DialogBox(hInst,"FACEBOX",hDlg,lpProcFace);
                    FreeProcInstance(lpProcFace);
                    break;
         }
  }
  return (FALSE);
}//About


/*****************************************************************************
 Funktion:      Face

 Aufgabe:       Informationsdialogbox darstellen
```

```
 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*************************************************************************/

BOOL FAR PASCAL Face(HWND     hDlg,
                     unsigned message,
                     WORD     wParam,
                     LONG     lParam)


{
HDC         hDC;
HDC         hMemoryDC;
PAINTSTRUCT ps;

  switch (message){
    case WM_INITDIALOG:
         hFaceBitmap = LoadBitmap(hInst,"face");
         return (TRUE);
    case WM_COMMAND:
         switch(wParam){
               case IDOK:
               case IDCANCEL:
                    EndDialog(hDlg, TRUE);
                    return(TRUE);
         }
    case WM_PAINT:
         hDC = BeginPaint(hDlg,&ps);
         hMemoryDC = CreateCompatibleDC(hDC);
         hOldBitmap = SelectObject(hMemoryDC,hFaceBitmap);
         if(hOldBitmap)
         {
           BitBlt (hDC,0,0,600,600,hMemoryDC,0,0,SRCCOPY);
           SelectObject(hMemoryDC,hOldBitmap);
         }
         DeleteDC(hMemoryDC);
         EndPaint(hDlg,&ps);
          break;

  }
  return (FALSE);
}//Face
```

7.4.3. N_BRAIN.C

```
/*****************************************************************************
 Modul:       N_BRAIN.C

 Aufgabe:     Dieses Modul übernimmt die Bilderkennung mittels eines
              neuronalen Netzes

 Funktionen:  VerknuepfeNetz
              SetzeEingangsVektor
              SetzeAusgangsVektor
              Sigmoid
              SigmoidStrich
              BerechneAusgang
              HoleEingangsVektor
              SetzeEingangsTrainingsVektor
              ErstelleMinMaxVektoren
              MaxOfTraining
              MinOfTraining
              TransformiereEingangsVektor
              NeuronenMaximum
              Backpropagation
              Brain
              ZeichneBrain
              LadeNeuronalesNetz
              SpeicherNeuronalesNetz
              NeuesNetz
              NeuronenDaten
              TeileDaten

*****************************************************************************/


#include "stdio.h"
#include "stdlib.h"
#include "windows.h"
#include "neurorob.h"
#include "neuromen.h"
#include "n_extern.h"
#include "brainmen.h"
#include "ndl_nnet.h"
#include "ndl_ndat.h"
#include "ndl_tdat.h"
#include "ndl_test.h"
#include "math.h"


#define ERKENNUNGSSCHWELLE 0.98
#define NETZ_HOEHE 32
#define NETZ_BREITE 4
#define ZUFALL ((double)(rand())/60000.0)
#define ROBOTERRANDOM (((double)rand()/32767.0)*2.0-1.0)
#define ROBOTERRANDOMP ((int)(((double)rand())/((double)RAND_MAX)*355))
#define TETACONST (1.0)
#define ABSTANDGROSS 40
#define ABSTANDMITTEL 21
#define ABSTANDKLEIN 15
#define GROESSEGROSS 20
#define GROESSEMITTEL 12
#define GROESSEKLEIN 8
#define HIGH 60
#define MIDDLE 500
```

```
#define LOW 3000
#define HIGHDURCHLAEUFE 1000
#define MIDDLEDURCHLAEUFE 300
#define LOWDURCHLAEUFE 1
#define XORGOFFSET (-280)
#define YORGOFFSET (-40)
#define HIDDENMAX NETZ_HOEHE
#define HIDDENPAGE 5
#define TESTMAX 1000
#define TESTPAGE 100
#define MERKMALEMAX NETZ_HOEHE
#define TEILEMAX 16
#define LAYERMAX (NETZ_BREITE-2)
#define SAMPLEMAX 8
#define SAMPLEANZAHL (SAMPLEMAX*TEILEMAX) //muss ein Divisor von 65536 sein !
#define TEILE 0
#define LAYER 1
#define SAMPLE 2
#define KEIN 0
#define TRAINING 1
#define KOMPLETT 2


typedef struct eingaenge
        {
          double  huge *x;
          double  w;
        }EINGAENGE;

typedef struct neuron
        {
          EINGAENGE huge xin[NETZ_HOEHE];
          double    teta;
          double    p;
          double    ausgang;
          double    delta;
          int       xpos1;
          int       ypos1;
          int       xpos2;
          int       ypos2;
        }NEURON;

int           Netzform[NETZ_BREITE]={0,3,3,3};

NEURON  huge  Netz[NETZ_HOEHE][NETZ_BREITE];

double  huge  EingangsTrainingsVektor[NETZ_HOEHE][SAMPLEMAX*TEILEMAX];
double  huge  AusgangsTrainingsVektor[NETZ_HOEHE][SAMPLEMAX*TEILEMAX];

double  huge  EingangsVektor[NETZ_HOEHE];
double  huge  AusgangsVektor[NETZ_HOEHE];

double  huge  Maximum[NETZ_HOEHE];
double  huge  Minimum[NETZ_HOEHE];

HPEN          hBluePen;
HPEN          hBlackPen;
HPEN          hGreenPen;

HBRUSH        hOldBrush;
HBRUSH        hRedBrush[256];
HBRUSH        hGreenBrush;
HBRUSH        hBlueBrush;
```

```
RECT          rClientRect;

int           scrollhmax = 700;
int           scrollvmax = 1700;
int           hpage      = 100;
int           vpage      = 100;
int           balkenmerker;
int           nDurchlaeufe      = MIDDLEDURCHLAEUFE;
int           nPrio             = MIDDLE;
int           nHiddenlayer      = 1;
int           nTeile            = 2;
int           nSampleProTeil    = 1;
int           nNeuronenAbstand  = ABSTANDMITTEL;
int           nNeuronenGroesse  = GROESSEMITTEL;
int           nXoffset          = 20;
int           nLayerMax         = 3;
int           nAktuelli;              //
int           nAktuellj;              // Selektiertes Neuron
int           nHorzAuswahl;
int           nHorzmax[3]       ={TEILEMAX,LAYERMAX,SAMPLEMAX};
int           nHorzPage[3]      ={3,1,3};
int           nRoboterEinsatz   = KEIN;
int           nTimerz           = 0;

static int    nSelect;


long          lPropagations     = 0L;


BOOL          bMerkmal[11];
BOOL          bStart            = FALSE;
BOOL          bNurEllipsen      = FALSE;
BOOL          bZeigeTraining    = FALSE;
BOOL          bBrainIcon        = TRUE;
BOOL          bRoboterEinsatz   = FALSE;
BOOL          bErkennung        = FALSE;
BOOL          bTestphase        = FALSE;

FARPROC       lpTeileDaten;

char          TeileGroesse[TEILEMAX+1][12];
char          TeileFach[TEILEMAX+1][12];
char          TeileHoehe[TEILEMAX+1][12];
char          TeileName[TEILEMAX+1][21]={ "Teil 1",
                                          "Teil 2",
                                          "Teil 3",
                                          "Teil 4",
                                          "Teil 5",
                                          "Teil 6",
                                          "Teil 7",
                                          "Teil 8",
                                          "Teil 9",
                                          "Teil 10",
                                          "Teil 11",
                                          "Teil 12",
                                          "Teil 13",
                                          "Teil 14",
                                          "Teil 15",
                                          "Teil 16"};

char        MerkmalName[MERKMALEMAX+1][50];
```

```
char        NetzName[128];

char        MajorAng[20];


/*************************************************************************
 Funktion:      VerknuepfeNetz

 Aufgabe:       Diese Funktion verbindet die einzelnen Neuronen zu einem Netz

 Parameter:     -

 Rückgabewert:  -

*************************************************************************/


void  VerknuepfeNetz(void)
{
int nHoehe;
int nBreite;
int nNeuroneingaenge=0;

  for(nHoehe=0;nHoehe<Netzform[0];++nHoehe)
  {
    (Netz[nHoehe][0].xin[nNeuroneingaenge]).x =
      (double huge *)&EingangsVektor[nHoehe];
    (Netz[nHoehe][0].xin[nNeuroneingaenge]).w = ZUFALL;
    Netz[nHoehe][0].teta = TETACONST;
  }
  for(nBreite=1;nBreite<nHiddenlayer+2;++nBreite)
    for(nHoehe=0;nHoehe<Netzform[nBreite];++nHoehe)
      for(nNeuroneingaenge=0;nNeuroneingaenge<Netzform[nBreite-1];
            ++nNeuroneingaenge)
      {
        (Netz[nHoehe][nBreite].xin[nNeuroneingaenge]).x =
          (double huge *)&(Netz[nNeuroneingaenge][nBreite-1].ausgang);
        (Netz[nHoehe][nBreite].xin[nNeuroneingaenge]).w = ZUFALL;
        Netz[nHoehe][nBreite].teta = TETACONST;
      }
}//VerknuepfeNetz

/*************************************************************************
 Funktion:      SetzeEingangsVektor

 Aufgabe:       Setzt den EingangsVektor einem EingangsTrainingsVektor gleich

 Parameter:     sample: bestimmmt den zu setzenden EingangsTrainingsVektor

 Rückgabewert:  -

*************************************************************************/

void SetzeEingangsVektor(int sample)
{
int z;
  for(z=0;z<Netzform[0];++z)
    EingangsVektor[z]=EingangsTrainingsVektor[z][sample];
  TransformiereEingangsVektor();
}//SetzeEingangsVektor

/*************************************************************************
 Funktion:      SetzeAusgangsVektor
```

```
 Aufgabe:       Setzt den AusgangsVektor einem AusgangsTrainingsVektor gleich

 Parameter:     sample: bestimmmt den zu setzenden AusgangsTrainingsVektor

 Rückgabewert:  -

*****************************************************************************/

void SetzeAusgangsVektor(int sample)
{
int z;
  for(z=0;z<Netzform[nHiddenlayer+1];++z)
    AusgangsVektor[z]=AusgangsTrainingsVektor[z][sample];
}//SetzeAusgangsVektor

/*****************************************************************************
 Funktion:      Sigmoid

 Aufgabe:       Berechnet die Sigmoide-Funktion

 Parameter:     p: Argument für die Funktion

 Rückgabewert:  double: Funktionswert der Sigmoiden

*****************************************************************************/

double  Sigmoid(double p)
{
  return(1.0/(1.0+exp(-p)));
}//Sigmoid

/*****************************************************************************
 Funktion:      SigmoidStrich

 Aufgabe:       Berechnet die Abgleitete Funktion der Sigmoiden

 Parameter:     p: Argument für die Funktion

 Rückgabewert:  double: Funktionswert der Abgeleiteten Funktion

*****************************************************************************/

double  SigmoidStrich(double p)
{
static  double  dDummy;

  dDummy = Sigmoid(p);
  dDummy *= dDummy;
  return(exp(-p)*dDummy);
}//SigmoidStrich

/*****************************************************************************
 Funktion:      BerechneAusgang

 Aufgabe:       Berechnet den Ausgangslayer von dem im Moment angelegten
                EingangsVektors des neuronalen Netzes

 Parameter:     -

 Rückgabewert:  -

*****************************************************************************/
```

```
void BerechneAusgang(void)
{
double  p;
int     nHoehe;
int     nBreite;
int     nNeuroneingaenge;

  for(nHoehe=0;nHoehe<Netzform[0];++nHoehe)
    Netz[nHoehe][0].ausgang = EingangsVektor[nHoehe];
  for(nBreite=1;nBreite<nHiddenlayer+2;++nBreite)
    for(nHoehe=0;nHoehe<Netzform[nBreite];++nHoehe)
    {
      p=0.0;
      for(nNeuroneingaenge=0;nNeuroneingaenge<Netzform[nBreite-1];
          ++nNeuroneingaenge)
        p += *(Netz[nHoehe][nBreite].xin[nNeuroneingaenge].x) *
             Netz[nHoehe][nBreite].xin[nNeuroneingaenge].w;
      p-=Netz[nHoehe][nBreite].teta;
      Netz[nHoehe][nBreite].p = p;
      Netz[nHoehe][nBreite].ausgang = Sigmoid(p);
    }
}//BerechneAusgang

/*****************************************************************************
 Funktion:      HoleEingangsVektor

 Aufgabe:       Holt die Merkmale von der Sivips-Anlage und selektiert
                die für die Erkennung relevanten Merkmale

 Parameter:     -

 Rückgabewert:  -

*****************************************************************************/

void HoleEingangsVektor(void)
{
int     Grauwerte[64];
char    Grauwert[15];
char    Merkmal[20];
char    copy[40];
int     i,j,k,l;
double  x;

  k=0;
  for(l=0;l<10;++l)
  {
    SendeComm2(nCid2,"A");
    HoleVideomatZeile((LPSTR)Merkmal);
    if(bMerkmal[l]==TRUE)
    {
        x=atof(Merkmal);
        EingangsVektor[k]=x;
        ++k;
    }
  }
  x=0;
  for(i=0;i<16;++i)
  {
    x=0;
    for(j=0;j<4;++j)
    {
```

```
      SendeComm2(nCid2,"A");
      HoleVideomatZeile((LPSTR)Grauwert);
      x+=atof(Grauwert);
    }
    EingangsVektor[i+k]=x;
  }
  SendeComm2(nCid2,"A");
  HoleVideomatZeile((LPSTR)copy);
  i=0;
  while(copy[i]==' ')
    ++i;
  lstrcpy((LPSTR)MajorAng,(LPSTR)&(copy[i]));

}//HoleEingangsVektor

/*****************************************************************************
 Funktion:      SetzeEingangsTrainingsVektor

 Aufgabe:       Speichert den aktuellen EingangsVektor im EingangsTrainings-
                Vektor

 Parameter:     sample: bestimmt in welchem EingangsTrainingsVektor der
                        EingangsVektor gespeichert werden soll

 Rückgabewert:  -

*****************************************************************************/

void SetzeEingangsTrainingsVektor(int sample)
{
int z;
  for(z=0;z<Netzform[0];++z)
    EingangsTrainingsVektor[z][sample]=EingangsVektor[z];
}//SetzeEingangsTrainingsVektor

/*****************************************************************************
 Funktion:      ErstelleMinMaxVektoren

 Aufgabe:       Erstellt die zur Transformation notwendigen Umrechnungs-
                vektoren

 Parameter:     -

 Rückgabewert:  -

*****************************************************************************/

void ErstelleMinMaxVektoren(void)
{
int     z;
  for(z=0;z<Netzform[0];++z)
  {
    Maximum[z] = MaxOfTraining(z);
    Minimum[z] = MinOfTraining(z,Maximum[z]);
  }
}//ErstelleMinMaxVektoren

/*****************************************************************************
 Funktion:      MaxOfTraining

 Aufgabe:       Bestimmt den Maximalwert eines Eingangs in den Trainings-
                vektoren
```

```
 Parameter:     nEingang: Bestimmt den zu untersuchenden Eingang

 Rückgabewert:  -

*****************************************************************************/

double MaxOfTraining(int nEingang)
{
double  dMaximum=-1E300;
int     z;
  for(z=0;z<nTeile*nSampleProTeil;++z)
    dMaximum = max(dMaximum,EingangsTrainingsVektor[nEingang][z]);
  return(dMaximum);
}//MaxOfTraining

/*****************************************************************************
 Funktion:      MinOfTraining

 Aufgabe:       Bestimmt den Minimalwert eines Eingangs in den Trainings-
                vektoren

 Parameter:     nEingang: Bestimmt den zu untersuchenden Eingang

 Rückgabewert:  -

*****************************************************************************/

double MinOfTraining(int nEingang,double dMinimum)
{
int     z;
  for(z=0;z<nTeile*nSampleProTeil;++z)
    dMinimum = min(dMinimum,EingangsTrainingsVektor[nEingang][z]);
  return(dMinimum);
}//MinOfTraining

/*****************************************************************************
 Funktion:      TransformiereEingangsVektor

 Aufgabe:       Transformiert den EingangsVektor mit Hilfe der MinMaxVektoren
                auf einen Werteberech zwischen 1/11 und 1

 Parameter:     -

 Rückgabewert:  -

*****************************************************************************/

void TransformiereEingangsVektor(void)
{
int z;
  for(z=0;z<Netzform[0];++z)
  {
    if((Maximum[z]-Minimum[z])>0.0)
    {
      if(EingangsVektor[z]>Maximum[z])
        EingangsVektor[z]=1.0;
      else if (EingangsVektor[z]<Minimum[z])
        EingangsVektor[z]=0.1;
      else
        EingangsVektor[z]=(((EingangsVektor[z]-Minimum[z])/
                           (Maximum[z]-Minimum[z]))+0.1)/1.1;
    }
    else
```

```
      EingangsVektor[z]=0.1;
  }
}//TransformiereEingangsVektor

/*****************************************************************************
 Funktion:      NeuronenMaximum

 Aufgabe:       Bestimmt das Neuron im Ausgangslayer welches das höchste
                Potential hat

 Parameter:     -

 Rückgabewert:  int: Bestimmt die Nummer des Neuron

*****************************************************************************/

int NeuronenMaximum(void)
{
int     i;
int     nAusgangsNeuron;
double  Merker;

  nAusgangsNeuron = -1;
  Merker=0.0;
  for(i=0;i<Netzform[nHiddenlayer+1];++i)
  {
    if(Netz[i][nHiddenlayer+1].ausgang>Merker)
    {
      Merker = Netz[i][nHiddenlayer+1].ausgang;
      nAusgangsNeuron=i;
    }
  }
  return(nAusgangsNeuron);
}//NeuronenMaximum

/*****************************************************************************
 Funktion:      Backpropagation

 Aufgabe:       Berechnet die Gewichtsanpassung der Neuronenverbindungen

 Parameter:     -

 Rückgabewert:  -

*****************************************************************************/

void Backpropagation(void)
{
int     i,j,k;
double  delta;

  for(i=0;i<Netzform[nHiddenlayer+1];++i)
    Netz[i][nHiddenlayer+1].delta =
      (Netz[i][nHiddenlayer+1].ausgang-AusgangsVektor[i])*
      SigmoidStrich(Netz[i][nHiddenlayer+1].p);
  for(i=nHiddenlayer;i>0;--i)
    for(j=0;j<Netzform[i];++j)
    {
      delta = 0.0;
      for(k=0;k<Netzform[i+1];++k)
        delta += Netz[k][i+1].delta * Netz[k][i+1].xin[j].w;
      delta *= SigmoidStrich(Netz[j][i].p);
      Netz[j][i].delta = delta;
```

```
    }
  for(i=1;i<nHiddenlayer+2;++i)
    for(j=0;j<Netzform[i];++j)
      for(k=0;k<Netzform[i-1];++k)
        Netz[j][i].xin[k].w -= Netz[j][i].delta * Netz[k][i-1].ausgang;
}//BackPropagation

/*************************************************************************
 Funktion:      Brain

 Aufgabe:       Callbackfunktion von Brain. Hier wird das Brain-Menu
                abgearbeitet

 Parameter:
  HWND      hWnd   : Fensterhandle
  unsigned  msg    : Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  long    wird von Windows verwaltet
*************************************************************************/


long FAR PASCAL Brain(HWND hWnd,unsigned msg,WORD wParam,LONG lParam)

{
FARPROC     lpProcNeuesNetz;
FARPROC     lpProcTestNetz;
FARPROC     lpNeuronenDaten;
int         i,j,k,x,y,z,xDreh,yDreh,nDrehWinkel;
int         nBreite;
int         nHoehe;
int         nNeuroneneingaenge;
int         time1,time2;
HDC         hDC;
FILE        *FilePointer;
LONG        lTimer;
double      alpha;

  switch(msg){
    case WM_COMMAND:
         switch (wParam){
           case IDM9_NEU:
                lpProcNeuesNetz = MakeProcInstance(NeuesNetz,hInst);
                DialogBox(hInst,"NEUESNETZ",hWnd,lpProcNeuesNetz);
                FreeProcInstance(lpProcNeuesNetz);
                bSpeichernAbfrage=TRUE;
                BringWindowToTop(hWnd);
                break;
           case IDM9_TEST:
                if(bTestphase)
                {
                  CheckMenuItem(hBrainMenu,IDM9_TEST,MF_UNCHECKED);
                  bTestphase = FALSE;
                }
                else
                {
                  CheckMenuItem(hBrainMenu,IDM9_TEST,MF_CHECKED);
                  bTestphase = TRUE;
                }
                break;
           case IDM9_RESET:
```

```
      if(MessageBox(GetFocus(),
        (LPSTR)"Soll das neuronale Netz\n wirklich alles vergessen",
        (LPSTR)"",MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL)==IDYES)
          VerknuepfeNetz();
      InvalidateRect(hWnd,NULL,TRUE);
      break;
case IDM9_ENDE:
      if(bStart)
        SendMessage(hWnd,WM_COMMAND,IDM9_START,0L);
      ShowWindow(hWndBrain,SW_HIDE);
      bBrain = FALSE;
      CheckMenuItem(hMainMenu,IDM_NEUINIT,MF_UNCHECKED);
      break;
case IDM9_START:
      if(bStart)
      {
        CheckMenuItem(hBrainMenu,IDM9_START,MF_UNCHECKED);
        bStart = FALSE;
        KillTimer(hWnd,1);
      }
      else
      {
        CheckMenuItem(hBrainMenu,IDM9_START,MF_CHECKED);
        bStart = TRUE;
        lPropagations=0L;
        SetTimer(hWnd,1,nPrio,NULL);
      }
      bSpeichernAbfrage=TRUE;
      break;
case IDM9_ANZEIGEN:
      if(bZeigeTraining)
      {
        CheckMenuItem(hBrainMenu,IDM9_ANZEIGEN,MF_UNCHECKED);
        bZeigeTraining = FALSE;
      }
      else
      {
        CheckMenuItem(hBrainMenu,IDM9_ANZEIGEN,MF_CHECKED);
        bZeigeTraining = TRUE;
      }
      break;
case IDM9_PRIOHIGH:
      nPrio = HIGH;
      nDurchlaeufe = HIGHDURCHLAEUFE;
      CheckMenuItem(hBrainMenu,IDM9_PRIOHIGH,MF_CHECKED);
      CheckMenuItem(hBrainMenu,IDM9_PRIOMIDDLE,MF_UNCHECKED);
      CheckMenuItem(hBrainMenu,IDM9_PRIOLOW,MF_UNCHECKED);
      break;
case IDM9_PRIOMIDDLE:
      nPrio = MIDDLE;
      nDurchlaeufe = MIDDLEDURCHLAEUFE;
      CheckMenuItem(hBrainMenu,IDM9_PRIOHIGH,MF_UNCHECKED);
      CheckMenuItem(hBrainMenu,IDM9_PRIOMIDDLE,MF_CHECKED);
      CheckMenuItem(hBrainMenu,IDM9_PRIOLOW,MF_UNCHECKED);
      break;
case IDM9_PRIOLOW:
      nPrio = LOW;
      nDurchlaeufe = LOWDURCHLAEUFE;
      CheckMenuItem(hBrainMenu,IDM9_PRIOHIGH,MF_UNCHECKED);
      CheckMenuItem(hBrainMenu,IDM9_PRIOMIDDLE,MF_UNCHECKED);
      CheckMenuItem(hBrainMenu,IDM9_PRIOLOW,MF_CHECKED);
      break;
case IDM9_GROSS:
```

```
      nNeuronenAbstand = ABSTANDGROSS;
      nNeuronenGroesse = GROESSEGROSS;
      CheckMenuItem(hBrainMenu,IDM9_GROSS,MF_CHECKED);
      CheckMenuItem(hBrainMenu,IDM9_MITTEL,MF_UNCHECKED);
      CheckMenuItem(hBrainMenu,IDM9_KLEIN,MF_UNCHECKED);
      InvalidateRect(hWnd,NULL,TRUE);
      break;
case IDM9_MITTEL:
      nNeuronenAbstand = ABSTANDMITTEL;
      nNeuronenGroesse = GROESSEMITTEL;
      CheckMenuItem(hBrainMenu,IDM9_GROSS,MF_UNCHECKED);
      CheckMenuItem(hBrainMenu,IDM9_MITTEL,MF_CHECKED);
      CheckMenuItem(hBrainMenu,IDM9_KLEIN,MF_UNCHECKED);
      InvalidateRect(hWnd,NULL,TRUE);
      break;
case IDM9_KLEIN:
      nNeuronenAbstand = ABSTANDKLEIN;
      nNeuronenGroesse = GROESSEKLEIN;
      CheckMenuItem(hBrainMenu,IDM9_GROSS,MF_UNCHECKED);
      CheckMenuItem(hBrainMenu,IDM9_MITTEL,MF_UNCHECKED);
      CheckMenuItem(hBrainMenu,IDM9_KLEIN,MF_CHECKED);
      InvalidateRect(hWnd,NULL,TRUE);
      break;
case IDM9_ALS:
      szFileCaption = (LPSTR)"Neuronales Netz speichern";
      strcpy(DefExt,".NET");
      lpProcLaden = MakeProcInstance(DateiLaden, hInst);
      if(!DialogBox(hInst,"FILEOPERATION",hWnd,lpProcLaden))
        break;
      FreeProcInstance(lpProcLaden);
      lstrcpy((LPSTR)NetzName,(LPSTR)FileName);
case IDM9_SPEICHERN:
      lstrcpy((LPSTR)FileName,(LPSTR)NetzName);
      if(SpeicherNeuronalesNetz())
      {
        bSpeichern = TRUE;
        bSpeichernAbfrage = FALSE;
      }
      break;
case IDM9_LADEN:
      if(LadeNeuronalesNetz())
      {
        lstrcpy((LPSTR)NetzName,(LPSTR)FileName);
        bNetzdefine = TRUE;
        bSpeichern=TRUE;
        SetzeEingangsVektor(0);
        BerechneAusgang();
        InvalidateRect(hWndBrain,NULL,TRUE);
        bSpeichernAbfrage=FALSE;
      }
      break;
case IDM9_ERKENNUNG:
      if(bErkennung)
      {
        CheckMenuItem(hBrainMenu,IDM9_ERKENNUNG,MF_UNCHECKED);
        bErkennung = FALSE;
        KillTimer(hWnd,1);
      }
      else
      {
        CheckMenuItem(hBrainMenu,IDM9_ERKENNUNG,MF_CHECKED);
        bErkennung = TRUE;
        SetTimer(hWnd,1,1,NULL);
```

```
            }
            break;
        case IDM9_ROBOTEREINSATZ:
            if(bRoboterEinsatz)
            {
              CheckMenuItem(hBrainMenu,IDM9_ROBOTEREINSATZ,MF_UNCHECKED);
              bRoboterEinsatz = FALSE;
            }
            else
            {
              CheckMenuItem(hBrainMenu,IDM9_ROBOTEREINSATZ,MF_CHECKED);
              bRoboterEinsatz = TRUE;
            }
            break;
        case IDM9_EDITTEILEDATEN:
            lpTeileDaten = MakeProcInstance(TeileDaten,hInst);
            DialogBox(hInst,"TEILEDATEN",hWnd,lpTeileDaten);
            FreeProcInstance(lpTeileDaten);
            bSpeichernAbfrage=TRUE;
            break;
      }
      break;
case WM_TIMER:
      KillTimer(hWnd,1);
      if(bStart)
      {
        ++nSelect;
        lTimer=GetTickCount();
        do
        {
          SetzeEingangsVektor(nTimerz);
          BerechneAusgang();
          SetzeAusgangsVektor(nTimerz);
          Backpropagation();
          ++lPropagations;
          ++nTimerz;
          if(nTimerz==nSampleProTeil*nTeile)
            nTimerz=0;
        }while((GetTickCount()-lTimer)<nDurchlaeufe);
        SetzeEingangsVektor(nSelect%(nTeile*nSampleProTeil));
        BerechneAusgang();
        if(bZeigeTraining && !IsIconic(hWnd))
        {
          InvalidateRect(hWnd,NULL,FALSE);
          SendMessage(hWnd,WM_PAINT,0,0L);
        }
        SetTimer(hWnd,1,nPrio,NULL);
        break;
      }
      if(bErkennung)
      {
        if(bRoboterEinsatz && nRoboterEinsatz==KOMPLETT)
        {
          SendeComm2(nCid2,"T");
          HoleVideomatZeile((LPSTR)dummy2);
          if(dummy2[0]=='N')
          {
            SetTimer(hWnd,1,1,NULL);
            break;
          }
        }
        HoleEingangsVektor();
        TransformiereEingangsVektor();
```

```
        BerechneAusgang();
        InvalidateRect(hWndBrain,NULL,TRUE);
        SendMessage(hWndBrain,WM_PAINT,0,0L);
        if(bRoboterEinsatz && nRoboterEinsatz==KOMPLETT)
        {
          HolePartSchwerpunkt();
          alpha=atof(MajorAng);
          nDrehWinkel=(int)((-alpha)*180.0/M_PI+90);
          while(nDrehWinkel>3)
            nDrehWinkel-=180;
          while(nDrehWinkel<-177)
            nDrehWinkel+=180;
          MO("996");
          wsprintf((LPSTR)dummy,(LPSTR)"0,0,0,0,%i",nDrehWinkel);
          MJ((LPSTR)dummy);
          xDreh=(int)(xUrsprung-xPart);
          yDreh=(int)(yUrsprung-yPart);
          wsprintf((LPSTR)dummy,(LPSTR)"%i,%i,%i",xDreh,yDreh,
            atoi(TeileHoehe[NeuronenMaximum()])+30);
          AbsolutRelativ();
          SendeComm1(nCid,dummy);
          wsprintf((LPSTR)dummy,(LPSTR)"%i,%i,%i",xDreh,yDreh,
            atoi(TeileHoehe[NeuronenMaximum()]));
          AbsolutRelativ();
          if(!bTestphase)
            SendeComm1(nCid,dummy);
          GC();
          MO((LPSTR)"999,C");
          wsprintf((LPSTR)dummy,(LPSTR)"%s,C",
            (LPSTR)TeileFach[NeuronenMaximum()]);
          MO((LPSTR)dummy);
          GO();
        }
        SetTimer(hWnd,1,1,NULL);
      }
      break;
case WM_CREATE:
      hGehirnBitmap = LoadBitmap(hInst,"GEHIRNBILD");
      SetScrollRange(hWnd,SB_VERT,0,scrollvmax,0);
      SetScrollRange(hWnd,SB_HORZ,0,scrollhmax,0);
      for(i=0;i<256;++i)
        hRedBrush[i] = CreateSolidBrush(RGB(i,0,0));
      hBluePen    = CreatePen(PS_SOLID,1,RGB(0,0,255));
      hGreenPen   = CreatePen(PS_SOLID,1,RGB(0,255,0));
      hGreenBrush= CreateSolidBrush(RGB(0,255,0));
      hBlueBrush  = CreateSolidBrush(RGB(0,0,255));
      hBlackPen   = CreatePen(PS_SOLID,1,RGB(0,0,0));
      for(ndummy=0;ndummy<TEILEMAX;++ndummy)
        lstrcpy((LPSTR)TeileHoehe[ndummy],(LPSTR)"200");
      break;
case WM_SIZE:
      GetClientRect(hWndBrain,(LPRECT)&rClientRect);
      hpage = rClientRect.right;
      vpage = rClientRect.bottom;
      break;
case WM_PAINT:
      ZeichneBrain(hWnd);
      break;
case WM_LBUTTONDBLCLK:
      if(!bStart)
      {
        x = LOWORD(lParam)+GetScrollPos(hWnd,SB_HORZ)+XORGOFFSET;
        y = HIWORD(lParam)+GetScrollPos(hWnd,SB_VERT)+YORGOFFSET;
```

```
        for(i=0;i<nHiddenlayer+2;++i)
          for(j=0;j<Netzform[i];++j)
          {
            if(Netz[j][i].xpos1<x &&
               Netz[j][i].xpos2>x &&
               Netz[j][i].ypos1<y &&
               Netz[j][i].ypos2>y)
            {
              nAktuelli=i;
              nAktuellj=j;
              lpNeuronenDaten = MakeProcInstance(NeuronenDaten, hInst);
              DialogBox(hInst,"NEURONENDATEN",hWnd,lpNeuronenDaten);
              FreeProcInstance(lpNeuronenDaten);
            }
          }
        }
        else
        {
          MessageBox(GetFocus(),
           (LPSTR)"Waehrend des Trainings keine Neuronendaten",
           NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        }
        break;
case WM_HSCROLL:
     balkenmerker = GetScrollPos(hWnd,SB_HORZ);
     switch(wParam){
       case SB_LINEDOWN:
            if(balkenmerker<scrollhmax)
            ++balkenmerker;
            break;
       case SB_LINEUP:
            if(balkenmerker>0)
              --balkenmerker;
            break;
       case SB_PAGEDOWN:
            if(balkenmerker<scrollhmax-hpage)
              balkenmerker+=hpage;
            else
              balkenmerker=scrollhmax;
            break;
       case SB_PAGEUP:
            if(balkenmerker>hpage)
              balkenmerker-=hpage;
            else
              balkenmerker=0;
            break;
       case SB_BOTTOM:
            balkenmerker=0;
            break;
       case SB_TOP:
            balkenmerker=scrollhmax;
            break;
       case SB_THUMBTRACK:
            balkenmerker=LOWORD(lParam);
            break;
     }
     SetScrollPos(hWnd,SB_HORZ,balkenmerker,TRUE);
     InvalidateRect(hWnd,NULL,TRUE);
     break;
case WM_VSCROLL:
     balkenmerker = GetScrollPos(hWnd,SB_VERT);
     switch(wParam){
       case SB_LINEDOWN:
```

```
            if(balkenmerker<scrollvmax)
            ++balkenmerker;
            break;
        case SB_LINEUP:
            if(balkenmerker>0)
              --balkenmerker;
            break;
        case SB_PAGEDOWN:
            if(balkenmerker<scrollvmax-vpage)
              balkenmerker+=vpage;
            else
              balkenmerker=scrollvmax;
            break;
        case SB_PAGEUP:
            if(balkenmerker>vpage)
              balkenmerker-=vpage;
            else
              balkenmerker=0;
            break;
        case SB_BOTTOM:
            balkenmerker=0;
            break;
        case SB_TOP:
            balkenmerker=scrollvmax;
            break;
        case SB_THUMBTRACK:
            balkenmerker=LOWORD(lParam);
            break;
      }
      SetScrollPos(hWnd,SB_VERT,balkenmerker,TRUE);
      InvalidateRect(hWnd,NULL,TRUE);
      break;
case WM_INITMENU:
      EnableMenuItem(hBrainMenu,IDM9_NEU,             MF_GRAYED);
      EnableMenuItem(hBrainMenu,IDM9_SPEICHERN,       MF_GRAYED);
      EnableMenuItem(hBrainMenu,IDM9_ALS,             MF_GRAYED);
      EnableMenuItem(hBrainMenu,IDM9_LADEN,           MF_GRAYED);
      EnableMenuItem(hBrainMenu,IDM9_START,           MF_GRAYED);
      EnableMenuItem(hBrainMenu,IDM9_TEST,            MF_ENABLED);
      EnableMenuItem(hBrainMenu,IDM9_RESET,           MF_GRAYED);
      EnableMenuItem(hBrainMenu,IDM9_ERKENNUNG,       MF_GRAYED);
      EnableMenuItem(hBrainMenu,IDM9_ROBOTEREINSATZ, MF_GRAYED);
      EnableMenuItem(hBrainMenu,IDM9_EDITTEILEDATEN, MF_GRAYED);
      if(!bStart && !bErkennung)
      {
        EnableMenuItem(hBrainMenu,IDM9_EDITTEILEDATEN,MF_ENABLED);
        EnableMenuItem(hBrainMenu,IDM9_NEU,  MF_ENABLED); //weg damit
        EnableMenuItem(hBrainMenu,IDM9_LADEN,MF_ENABLED);
        if(bNetzdefine)
        {
          EnableMenuItem(hBrainMenu,IDM9_ALS,MF_ENABLED);
          if(bSpeichern)
            EnableMenuItem(hBrainMenu,IDM9_SPEICHERN,MF_ENABLED);
          EnableMenuItem(hBrainMenu,IDM9_START,MF_ENABLED);
          EnableMenuItem(hBrainMenu,IDM9_RESET,MF_ENABLED);
          if(bVideomatInit)
          {
            EnableMenuItem(hBrainMenu,IDM9_NEU,           MF_ENABLED);
            EnableMenuItem(hBrainMenu,IDM9_ERKENNUNG,     MF_ENABLED);
          }
          if(bMoveMasterInit&&nRoboterEinsatz==KOMPLETT)
            EnableMenuItem(hBrainMenu,IDM9_ROBOTEREINSATZ,MF_ENABLED);
        }
```

```
          }
          if(bStart && !bErkennung)
            EnableMenuItem(hBrainMenu,IDM9_START,MF_ENABLED);
          if(!bStart && bErkennung)
          {
            EnableMenuItem(hBrainMenu,IDM9_ERKENNUNG,MF_ENABLED);
            if(bMoveMasterInit && nRoboterEinsatz==KOMPLETT)
              EnableMenuItem(hBrainMenu,IDM9_ROBOTEREINSATZ, MF_ENABLED);
          }
          break;
    case WM_DESTROY:
          for(i=0;i<256;++i)
            DeleteObject(hRedBrush[i]);
          DeleteObject(hGehirnBitmap);
          DeleteObject(hBlueBrush);
          DeleteObject(hGreenBrush);
          DeleteObject(hBluePen);
          DeleteObject(hBlackPen);
          DeleteObject(hGreenPen);
          if(bStart)
            SendMessage(hWndBrain,WM_COMMAND,IDM9_START,0L);
          if(bErkennung)
            SendMessage(hWndBrain,WM_COMMAND,IDM9_ERKENNUNG,0L);
          break;
    case WM_SYSCOMMAND:
          switch(wParam){
            case SC_CLOSE:
                 if(bStart)
                   SendMessage(hWndBrain,WM_COMMAND,IDM9_START,0L);
                 if(bErkennung)
                   SendMessage(hWndBrain,WM_COMMAND,IDM9_ERKENNUNG,0L);
                 PostMessage(hWndNeurorob,WM_COMMAND,IDM_NEUINIT,0L);
                 break;
            default:
                 return (DefWindowProc(hWnd, msg, wParam, lParam));
          }
          break;
    default:
          return (DefWindowProc(hWnd, msg, wParam, lParam));
  }
  return (NULL);
}//Brain

/*****************************************************************************
 Funktion:      ZeichneBrain

 Aufgabe:       Übernimmt die Darstellung des neuronalen Netzes

 Parameter:     hWnd: Handle des Fensters in dem gezeichnet werden soll

 Rückgabewert:  -

*****************************************************************************/


void ZeichneBrain(HWND hWnd)
{
HDC         hDC;
HDC         hMemoryDC;
PAINTSTRUCT ps;
int         i,j,k;

  hDC= BeginPaint(hWnd,&ps);
```

```
if(bNetzdefine)
{
  SetMapMode(hDC,MM_ANISOTROPIC);
  SetWindowExt(hDC,1,1);
  SetViewportExt(hDC,1,1);
  SetWindowOrg(hDC,GetScrollPos(hWnd,SB_HORZ)+XORGOFFSET,
    GetScrollPos(hWnd,SB_VERT)+YORGOFFSET);
  SelectObject(hDC,hBluePen);
  if(bStart)
  {
    wsprintf((LPSTR)dummy,"Trainingsanzahl : %li    ",lPropagations);
    TextOut(hDC,XORGOFFSET+10,-20,(LPSTR)dummy,lstrlen((LPSTR)dummy));

  }
  for(i=0;i<nHiddenlayer+1;++i)
    for(j=0;j<Netzform[i];++j)
    {
      for(k=0;k<Netzform[i+1];++k)
      {
        MoveTo(hDC,i*nNeuronenAbstand+nNeuronenGroesse/2+nXoffset,
          (j+1)*nNeuronenAbstand+nNeuronenGroesse/2+
          (nLayerMax/2-Netzform[i]/2)*nNeuronenAbstand);
        LineTo(hDC,(i+1)*nNeuronenAbstand+nNeuronenGroesse/2+nXoffset,
          (k+1)*nNeuronenAbstand+nNeuronenGroesse/2+
          (nLayerMax/2-Netzform[i+1]/2)*nNeuronenAbstand);
      }
      if(i==0)
      {
        MoveTo(hDC,XORGOFFSET+10,
          (j+1)*nNeuronenAbstand+nNeuronenGroesse/2+
          (nLayerMax/2-Netzform[i]/2)*nNeuronenAbstand);
        LineTo(hDC,i*nNeuronenAbstand+nNeuronenGroesse/2+nXoffset,
          (nLayerMax/2-Netzform[i]/2)*nNeuronenAbstand+
          (j+1)*nNeuronenAbstand+nNeuronenGroesse/2);
      }
    }
  SelectObject(hDC,hBlackPen);
  for(i=0;i<nHiddenlayer+2;++i)
    for(j=1;j<Netzform[i]+1;++j)
    {
      Netz[j-1][i].xpos1=i*nNeuronenAbstand+nXoffset;
      Netz[j-1][i].ypos1=j*nNeuronenAbstand+
        (nLayerMax/2-Netzform[i]/2)*nNeuronenAbstand;
      Netz[j-1][i].xpos2=i*nNeuronenAbstand+nNeuronenGroesse+nXoffset;
      Netz[j-1][i].ypos2=j*nNeuronenAbstand+nNeuronenGroesse+
        (nLayerMax/2-Netzform[i]/2)*nNeuronenAbstand;
      SelectObject(hDC,hRedBrush[min((int)fabs((Netz[j-1][i].ausgang*255)),
        255)]);
      Ellipse(hDC,Netz[j-1][i].xpos1,Netz[j-1][i].ypos1,
        Netz[j-1][i].xpos2,Netz[j-1][i].ypos2);
      if(i==nHiddenlayer+1)
      {
        lstrcpy((LPSTR)dummy,(LPSTR)TeileName[j-1]);
        if((AusgangsTrainingsVektor[j-1][nSelect%(nSampleProTeil*nTeile)]
          ==1)&&bStart)
          lstrcat((LPSTR)dummy,(LPSTR)"<-");
        else
          lstrcat((LPSTR)dummy,(LPSTR)"   ");
        if(NeuronenMaximum()==j-1)
        {
          SetTextColor(hDC,RGB(0,255,0));
          TextOut(hDC,Netz[j-1][i].xpos1+nNeuronenGroesse+5,
            Netz[j-1][i].ypos1,(LPSTR)dummy,lstrlen((LPSTR)dummy));
```

```
            SetTextColor(hDC,RGB(0,0,0));
          }
          else
            TextOut(hDC,Netz[j-1][i].xpos1+nNeuronenGroesse+5,
              Netz[j-1][i].ypos1,(LPSTR)dummy,lstrlen((LPSTR)dummy));
          if((AusgangsTrainingsVektor[j-1][nSelect%(nSampleProTeil*nTeile)]==1)
              &&bStart)
          {
            wsprintf((LPSTR)dummy,
              "Sample Nr.: %i      Soll: %s                    ",
              (nSelect%nSampleProTeil)+1,(LPSTR)TeileName[j-1]);
            TextOut(hDC,0,-20,(LPSTR)dummy,lstrlen((LPSTR)dummy));
          }
        }
        if(i==0)
        {
          sprintf((LPSTR)dummy,(LPSTR)"%10.9lf %s",Netz[j-1][i].ausgang,
            (LPSTR)MerkmalName[j-1]);
          TextOut(hDC,XORGOFFSET+10,
            Netz[j-1][i].ypos1,(LPSTR)dummy,lstrlen((LPSTR)dummy));
        }
      }
  }
  else
  {
    SetWindowOrg(hDC,GetScrollPos(hWnd,SB_HORZ),
      GetScrollPos(hWnd,SB_VERT)+YORGOFFSET);
    hMemoryDC = CreateCompatibleDC(hDC);
    hOldBitmap = SelectObject(hMemoryDC,hGehirnBitmap);
    if(hOldBitmap)
    {
      BitBlt (hDC,20,0,500,408,hMemoryDC,0,0,SRCCOPY);
      SelectObject(hMemoryDC,hOldBitmap);
    }
    DeleteDC(hMemoryDC);
  }
  EndPaint(hWnd,&ps);
}//ZeichneBrain

/*****************************************************************************
 Funktion:      LadeNeuronalesNetz

 Aufgabe:       Läd ein neuronales Netz

 Parameter:     -

 Rückgabewert:  BOOL: TRUE wenn gelungen

*****************************************************************************/


BOOL LadeNeuronalesNetz(void)
{
int     i,j,k;
int     nHoehe,nBreite,nNeuroneneingaenge;
FILE    *FilePointer;
HCURSOR hSaveCursor;

  if(bNetzdefine&&bSpeichernAbfrage)
  {
    if(MessageBox(GetFocus(),
      (LPSTR)"Moechten Sie das Netz speichern",(LPSTR)"",
      MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL)==IDYES)
```

```
  {
    if(bSpeichern)
      SendMessage(hWndBrain,WM_COMMAND,IDM9_SPEICHERN,0L);
    else
      SendMessage(hWndBrain,WM_COMMAND,IDM9_ALS,0L);
  }
}
szFileCaption = (LPSTR)"Neuronales Netz laden";
strcpy(DefExt,".NET");
lpProcLaden = MakeProcInstance(DateiLaden, hInst);
if(!DialogBox(hInst,"FILEOPERATION",hWndBrain,lpProcLaden))
  return FALSE;
FreeProcInstance(lpProcLaden) ;
FilePointer=fopen(FileName,"rt");
if(!FilePointer)
{
  MessageBox(GetFocus(),(LPSTR)"Datei konnte nicht geoeffnet werden",
    (LPSTR)"Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
  return (FALSE);
}
fgets(dummy, 90 , FilePointer);
dummy[strlen(dummy)-1]='\0';
if(lstrcmp((LPSTR)dummy,(LPSTR)VERSION)!=0)
{
  wsprintf((LPSTR)dummy,(LPSTR)"Gewaehlte Datei ist nicht vom Typ\n%s",
    (LPSTR)VERSION);
  MessageBox(GetFocus(),(LPSTR)dummy,(LPSTR)"Fehler",
    MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
  return FALSE;
}
hSaveCursor = SetCursor(hHourGlass);
fgets(dummy, 90 , FilePointer);
fgets(dummy, 90 , FilePointer);
nRoboterEinsatz=atoi(dummy);
fgets(dummy, 90 , FilePointer);
fgets(dummy, 90 , FilePointer);
nHiddenlayer=atoi(dummy);
fgets(dummy, 90 , FilePointer);
for(i=0;i<nHiddenlayer+2;++i)
{
  fgets(dummy, 90 , FilePointer);
  dummy[strlen(dummy)-1]='\0';
  Netzform[i]=atoi(dummy);
}
VerknuepfeNetz();
nLayerMax=0;
for(i=0;i<nHiddenlayer+2;++i)
  nLayerMax = max(nLayerMax,Netzform[i]);
for(i=0;i<Netzform[0];++i)
{
  fgets(dummy, 90 , FilePointer);
  dummy[strlen(dummy)-1]='\0';
  strcpy(MerkmalName[i],dummy);
}
for(i=0;i<Netzform[nHiddenlayer+1];++i)
{
  fgets(dummy, 90 , FilePointer);
  dummy[strlen(dummy)-1]='\0';
  strcpy(TeileName[i],dummy);
  fgets(dummy,90,FilePointer);
  dummy[strlen(dummy)-1]='\0';
  strcpy(TeileGroesse[i],dummy);
  fgets(dummy,90,FilePointer);
```

```
  dummy[strlen(dummy)-1]='\0';
  strcpy(TeileFach[i],dummy);
  fgets(dummy,90,FilePointer);
  dummy[strlen(dummy)-1]='\0';
  strcpy(TeileHoehe[i],dummy);
}
fgets(dummy,90,FilePointer);
for(nHoehe=0;nHoehe<Netzform[0];++nHoehe)
{
  fgets(dummy,90,FilePointer);
  dummy[strlen(dummy)-1]='\0';
  Maximum[nHoehe]=atof(dummy);
}
fgets(dummy,90,FilePointer);
for(nHoehe=0;nHoehe<Netzform[0];++nHoehe)
{
  fgets(dummy,90,FilePointer);
    dummy[strlen(dummy)-1]='\0';
  Minimum[nHoehe]=atof(dummy);
}
for(nBreite=1;nBreite<nHiddenlayer+2;++nBreite)
  for(nHoehe=0;nHoehe<Netzform[nBreite];++nHoehe)
  {
    fgets(dummy,90,FilePointer);
    for(nNeuroneneingaenge=0;nNeuroneneingaenge<
      Netzform[nBreite-1];++nNeuroneneingaenge)
    {
      fgets(dummy,90,FilePointer);
      dummy[strlen(dummy)-1]='\0';
      (Netz[nHoehe][nBreite].xin[nNeuroneneingaenge]).w=atof(dummy);
    }
  }
fgets(dummy,90,FilePointer);
for(i=0;i<11;++i)
{
  fgets(dummy,90,FilePointer);
  bMerkmal[i]=atoi(dummy);
}
fgets(dummy,90,FilePointer);
fgets(dummy,90,FilePointer);
nTeile=atoi(dummy);
fgets(dummy,90,FilePointer);
fgets(dummy,90,FilePointer);
nSampleProTeil=atoi(dummy);
for(i=0;i<nTeile*nSampleProTeil;++i)
{
  fgets(dummy,90,FilePointer);
  fgets(dummy,90,FilePointer);
  for(j=0;j<nLayerMax;++j)
  {
    fgets(dummy,90,FilePointer);
    EingangsTrainingsVektor[j][i]=atof(dummy);
  }
  fgets(dummy,90,FilePointer);
  for(j=0;j<nLayerMax;++j)
  {
    fgets(dummy,90,FilePointer);
    AusgangsTrainingsVektor[j][i]=atof(dummy);
  }
}
fclose(FilePointer);
SetCursor(hSaveCursor);
return TRUE;
```

```
}//LadeNeuronalesNetz

/*****************************************************************************
 Funktion:      SpeicherNeuronalesNetz

 Aufgabe:       speichert ein neuronales Netz

 Parameter:     -

 Rückgabewert:  BOOL: TRUE wenn gelungen

*****************************************************************************/

BOOL SpeicherNeuronalesNetz(void)
{
int     i,j,k;
int     nHoehe,nBreite,nNeuroneneingaenge;
FILE    *FilePointer;
HCURSOR hSaveCursor;


  ErstelleMinMaxVektoren();
  FilePointer=fopen(FileName,"wt");
  if(!FilePointer)
  {
    MessageBox(GetFocus(),"Datei konnte nicht geoeffnet werden",
      "Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
    return (FALSE);
  }
  hSaveCursor = SetCursor(hHourGlass);
  wsprintf((LPSTR)dummy,(LPSTR)"%s\n",(LPSTR)VERSION);
  fputs(dummy,FilePointer);
  fputs("Robotereinsatz:\n",FilePointer);
  wsprintf((LPSTR)dummy,(LPSTR)"%i\n",nRoboterEinsatz);
  fputs(dummy,FilePointer);
  fputs("nHiddenlayer :\n",FilePointer);
  wsprintf((LPSTR)dummy,(LPSTR)"%i\n",nHiddenlayer);
  fputs(dummy,FilePointer);
  fputs("Netzform:\n",FilePointer);
  for(i=0;i<nHiddenlayer+2;++i)
  {
    wsprintf((LPSTR)dummy,(LPSTR)"%i\n",Netzform[i]);
    fputs(dummy,FilePointer);
  }
  for(i=0;i<Netzform[0];++i)
  {
    wsprintf((LPSTR)dummy,(LPSTR)"%s\n",(LPSTR)MerkmalName[i]);
    fputs(dummy,FilePointer);
  }
  for(i=0;i<Netzform[nHiddenlayer+1];++i)
  {
    wsprintf((LPSTR)dummy,(LPSTR)"%s\n",(LPSTR)TeileName[i]);
    fputs(dummy,FilePointer);
    wsprintf((LPSTR)dummy,(LPSTR)"%s\n",(LPSTR)TeileGroesse[i]);
    fputs(dummy,FilePointer);
    wsprintf((LPSTR)dummy,(LPSTR)"%s\n",(LPSTR)TeileFach[i]);
    fputs(dummy,FilePointer);
    wsprintf((LPSTR)dummy,(LPSTR)"%s\n",(LPSTR)TeileHoehe[i]);
    fputs(dummy,FilePointer);
  }
  fputs("MaximumVektor:\n",FilePointer);
  for(nHoehe=0;nHoehe<Netzform[0];++nHoehe)
  {
```

```
    sprintf(dummy,"%26.18lf\n",Maximum[nHoehe]);
    fputs(dummy,FilePointer);
  }
  fputs("MinimumVektor:\n",FilePointer);
  for(nHoehe=0;nHoehe<Netzform[0];++nHoehe)
  {
    sprintf(dummy,"%26.18lf\n",Minimum[nHoehe]);
    fputs(dummy,FilePointer);
  }
  for(nBreite=1;nBreite<nHiddenlayer+2;++nBreite)
    for(nHoehe=0;nHoehe<Netzform[nBreite];++nHoehe)
    {
      wsprintf((LPSTR)dummy,(LPSTR)"Breite:%i Hoehe:%i\n",nBreite,nHoehe);
      fputs(dummy,FilePointer);
      for(nNeuroneneingaenge=0;nNeuroneneingaenge<
        Netzform[nBreite-1];++nNeuroneneingaenge)
      {
        sprintf(dummy,"%26.18lf\n",
        (Netz[nHoehe][nBreite].xin[nNeuroneneingaenge]).w);
        fputs(dummy,FilePointer);
      }
    }
  fputs("Merkmale:\n",FilePointer);
  for(i=0;i<11;++i)
    fprintf(FilePointer,(LPSTR)"%i\n",bMerkmal[i]);
  fputs("nTeile :\n",FilePointer);
  wsprintf((LPSTR)dummy,(LPSTR)"%i\n",nTeile);
  fputs(dummy,FilePointer);
  fputs("nSampleProTeil :\n",FilePointer);
  wsprintf((LPSTR)dummy,(LPSTR)"%i\n",nSampleProTeil);
  fputs(dummy,FilePointer);
  for(i=0;i<nTeile;++i)
    for(k=0;k<nSampleProTeil;++k)
    {
      fprintf(FilePointer,(LPSTR)"Teil %i Sample %i\n",i,k);
      fprintf(FilePointer,(LPSTR)"Eingang:\n");
      for(j=0;j<nLayerMax;++j)
        fprintf(FilePointer,(LPSTR)"%20.16lf\n",
          EingangsTrainingsVektor[j][i*nSampleProTeil+k]);
      fprintf(FilePointer,(LPSTR)"Ausgang:\n");
      for(j=0;j<nLayerMax;++j)
        fprintf(FilePointer,(LPSTR)"%20.16lf\n",
          AusgangsTrainingsVektor[j][i*nSampleProTeil+k]);
    }
  fclose(FilePointer);
  SetCursor(hSaveCursor);
  return TRUE;
}//SpeicherNeuronalesNetz

/*****************************************************************************
 Funktion:      NeuesNetz

 Aufgabe:       Erfragt die Daten des neuen Netzes, das angelegt werden soll

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*****************************************************************************/
```

```
BOOL FAR PASCAL NeuesNetz(HWND      hDlg,
                          unsigned  message,
                          WORD      wParam,
                          LONG      lParam)
{
int   i,j;
BOOL  back;

  switch (message){
    case WM_INITDIALOG:
         if(bNetzdefine&&bSpeichernAbfrage)
         {
           back=MessageBox(GetFocus(),
             (LPSTR)"Moechten Sie das Netz speichern",(LPSTR)"",
             MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL);
           if(back==IDYES)
           {
             if(bSpeichern)
               SendMessage(hWndBrain,WM_COMMAND,IDM9_SPEICHERN,0L);
             else
               SendMessage(hWndBrain,WM_COMMAND,IDM9_ALS,0L);
           }
         }
         SetScrollRange(GetDlgItem(hDlg,ID11_TEILE),SB_CTL,1,
           nHorzmax[TEILE],0);
         SetScrollPos(GetDlgItem(hDlg,ID11_TEILE),SB_CTL,2,1);
         SetScrollRange(GetDlgItem(hDlg,ID11_LAYER),SB_CTL,1,
           nHorzmax[LAYER],0);
         SetScrollPos(GetDlgItem(hDlg,ID11_LAYER),SB_CTL,1,1);
         SetScrollRange(GetDlgItem(hDlg,ID11_SAMPLE),SB_CTL,1,
           nHorzmax[SAMPLE],0);
         SetScrollPos(GetDlgItem(hDlg,ID11_SAMPLE),SB_CTL,1,1);
         for(i=1;i<NETZ_BREITE-1;++i)
         {
          ShowScrollBar(GetDlgItem(hDlg,ID11_S0+i),SB_CTL,FALSE);
          SetWindowText(GetDlgItem(hDlg,ID11_T0+i),(LPSTR)"");
         }
         SetWindowText(GetDlgItem(hDlg,ID11_T0),(LPSTR)"3");
         for(i=0;i<NETZ_BREITE-1;++i)
         {
           SetScrollRange(GetDlgItem(hDlg,ID11_S0+i),SB_CTL,1,HIDDENMAX,0);
           SetScrollPos(GetDlgItem(hDlg,ID11_S0+i),SB_CTL,3,1);
         }
         for(i=0;i<11;++i)
           bMerkmal[i] = FALSE;
         for(i=0;i<MERKMALEMAX;++i)
           MerkmalName[i][0]='\0';
         for(i=1;i<NETZ_BREITE;++i)
           Netzform[i]=3;
         for(i=0;i<TEILEMAX;++i)
         {
          lstrcpy((LPSTR)TeileGroesse[i],(LPSTR)"20");
          lstrcpy((LPSTR)TeileFach[i],(LPSTR)"999");
         }
         nHiddenlayer=1;
         nSampleProTeil = 1;
         Netzform[nHiddenlayer+1]=2;
         nTeile = 2;
         bNetzdefine  = FALSE;
         Netzform[0]  = 0;
         CheckRadioButton(hDlg,ID11_ROBOTER_KEIN,ID11_ROBOTER_KOMPLETT,
```

```
        ID11_ROBOTER_KEIN);
      nRoboterEinsatz = KEIN;
      return (TRUE);
  case WM_COMMAND:
      switch(wParam){
        case IDOK:
             nLayerMax=0;
             for(i=0;i<nHiddenlayer+2;++i)
               nLayerMax = max(nLayerMax,Netzform[i]);
             if(Netzform[0]>0)
             {
               bNetzdefine = TRUE;
               VerknuepfeNetz();
               BerechneAusgang();
               bSpeichern = FALSE;
             }
             InvalidateRect(hWndBrain,NULL,TRUE);
             for(i=0;i<SAMPLEANZAHL;++i)
               for(j=0;j<NETZ_HOEHE;++j)
               {
                 AusgangsTrainingsVektor[j][i]=0;
                 EingangsTrainingsVektor[j][i]=0;
               }
             EndDialog(hDlg,TRUE);
             if(!bSteuerung&&nRoboterEinsatz!=KEIN)
             {
               SendMessage(hWndNeurorob,WM_COMMAND,IDM_STEUERUNG,0L);
               BringWindowToTop(hDlgSteuerung);
             }
             lpTeileDaten = MakeProcInstance(TeileDaten,hInst);
             DialogBox(hInst,"TEILEDATEN",hDlg,lpTeileDaten);
             FreeProcInstance(lpTeileDaten);
             return (TRUE);
        case IDCANCEL:
             bNetzdefine = FALSE;
             bSpeichern = FALSE;
             InvalidateRect(hWndBrain,NULL,TRUE);
             EndDialog(hDlg,FALSE);
             return (TRUE);
             break;
        case ID11_ROBOTER_TRAINING:
        case ID11_ROBOTER_KOMPLETT:
             if(!bMoveMasterInit)
               break;
        case ID11_ROBOTER_KEIN:
             CheckRadioButton(hDlg,ID11_ROBOTER_KEIN,ID11_ROBOTER_KOMPLETT,
               wParam);
             nRoboterEinsatz = wParam-ID11_ROBOTER_KEIN;
             break;
        case ID11_TOTALAREA:
        case ID11_MAJOR:
        case ID11_MINOR:
        case ID11_NHOLES:
        case ID11_PERIMETER:
        case ID11_RMAX:
        case ID11_RMIN:
        case ID11_AVRAD:
        case ID11_LAENGE:
        case ID11_BREITE:
        case ID11_GRAUWERTE:
             if(bMerkmal[wParam-ID11_TOTALAREA] == FALSE)
             {
               CheckDlgButton(hDlg,wParam,(WORD) 1);
```

```
                bMerkmal[wParam-ID11_TOTALAREA] = TRUE;
                ++Netzform[0];
                if(wParam==ID11_GRAUWERTE)
                  Netzform[0]+=15;
              }
              else
              {
                CheckDlgButton(hDlg,wParam,(WORD) 0);
                bMerkmal[wParam-ID11_TOTALAREA] = FALSE;
                --Netzform[0];
                if(wParam==ID11_GRAUWERTE)
                  Netzform[0]-=15;
              }
              j=0;
              for(i=0;i<11;++i)
              {
                if(bMerkmal[i])
                {
                  ndummy=GetDlgItemText(hDlg,ID11_TOTALAREA+i,
                    (LPSTR)dummy,30);
                  dummy[ndummy]='\0';
                  wsprintf((LPSTR)MerkmalName[j],(LPSTR)"%s ",
                    (LPSTR)dummy);
                  ++j;
                }
              }
              return (TRUE);
        case ID11_TEILEDATEN:
              lpTeileDaten = MakeProcInstance(TeileDaten,hInst);
              DialogBox(hInst,"TEILEDATEN",hDlg,lpTeileDaten);
              FreeProcInstance(lpTeileDaten);
              break;
      }
      return TRUE;
case WM_VSCROLL:
      balkenmerker = GetScrollPos(HIWORD(lParam),SB_CTL);
      switch(wParam){
        case SB_LINEDOWN:
              if(balkenmerker<HIDDENMAX)
              ++balkenmerker;
              break;
        case SB_LINEUP:
              if(balkenmerker>1)
                --balkenmerker;
              break;
        case SB_PAGEDOWN:
              if(balkenmerker<HIDDENMAX-HIDDENPAGE)
                balkenmerker+=HIDDENPAGE;
              else
                balkenmerker=HIDDENMAX;
              break;
        case SB_PAGEUP:
              if(balkenmerker>HIDDENPAGE+1)
                balkenmerker-=HIDDENPAGE;
              else
                balkenmerker=1;
              break;
        case SB_BOTTOM:
              balkenmerker=1;
              break;
        case SB_TOP:
              balkenmerker=HIDDENMAX;
              break;
```

```
        case SB_THUMBTRACK:
             balkenmerker=LOWORD(lParam);
             break;
      }
      SetScrollPos(HIWORD(lParam),SB_CTL,balkenmerker,TRUE);
      itoa(balkenmerker,dummy,10);
      SetWindowText(GetDlgItem(hDlg,GetDlgCtrlID(HIWORD(lParam))+10)
       ,(LPSTR)dummy);
      Netzform[GetDlgCtrlID(HIWORD(lParam))-ID11_S0+1] = balkenmerker;
      return TRUE;
case WM_HSCROLL:
      if (HIWORD(lParam) == GetDlgItem(hDlg,ID11_TEILE))
        nHorzAuswahl = TEILE;
      if (HIWORD(lParam) == GetDlgItem(hDlg,ID11_LAYER))
        nHorzAuswahl = LAYER;
      if (HIWORD(lParam) == GetDlgItem(hDlg,ID11_SAMPLE))
        nHorzAuswahl = SAMPLE;
      balkenmerker = GetScrollPos(HIWORD(lParam),SB_CTL);
      switch(wParam){
        case SB_LINEDOWN:
             if(balkenmerker<nHorzmax[nHorzAuswahl])
             ++balkenmerker;
             break;
        case SB_LINEUP:
             if(balkenmerker>1)
               --balkenmerker;
             break;
        case SB_PAGEDOWN:
             if(balkenmerker<nHorzmax[nHorzAuswahl]-nHorzPage[nHorzAuswahl])
               balkenmerker+=nHorzPage[nHorzAuswahl];
             else
               balkenmerker=nHorzmax[nHorzAuswahl];
             break;
        case SB_PAGEUP:
             if(balkenmerker>nHorzPage[nHorzAuswahl])
               balkenmerker-=nHorzPage[nHorzAuswahl];
             else
               balkenmerker=1;
             break;
        case SB_BOTTOM:
             balkenmerker=1;
             break;
        case SB_TOP:
             balkenmerker=nHorzmax[nHorzAuswahl];
             break;
        case SB_THUMBPOSITION:
             balkenmerker=LOWORD(lParam);
             break;
      }
      SetScrollPos(HIWORD(lParam),SB_CTL,balkenmerker,TRUE);
      if (nHorzAuswahl == TEILE)
      {
        wsprintf((LPSTR)dummy,(LPSTR)"Anzahl der Teile : %i",balkenmerker);
        SetWindowText(GetDlgItem(hDlg,ID11_TEILETEXT),(LPSTR)dummy);
        nTeile = balkenmerker;
        Netzform[nHiddenlayer+1] = nTeile;
      }
      if (nHorzAuswahl == LAYER)
      {
        wsprintf((LPSTR)dummy,(LPSTR)"Hiddenlayer: %i",
         balkenmerker);
        nHiddenlayer = balkenmerker;
        SetWindowText(GetDlgItem(hDlg,ID11_LAYERTEXT),(LPSTR)dummy);
```

```
            for(i=0;i<nHiddenlayer;++i)
            {
              ShowScrollBar(GetDlgItem(hDlg,ID11_S0+i),SB_CTL,TRUE);
              Netzform[i+1]=GetScrollPos(GetDlgItem(hDlg,ID11_S0+i),SB_CTL);
              wsprintf((LPSTR)dummy,(LPSTR)"%i",Netzform[i+1]);
              SetWindowText(GetDlgItem(hDlg,ID11_T0+i),(LPSTR)dummy);
            }
            Netzform[nHiddenlayer+1] = nTeile;
            for(i=nHiddenlayer;i<NETZ_BREITE-1;++i)
            {
              ShowScrollBar(GetDlgItem(hDlg,ID11_S0+i),SB_CTL,FALSE);
              SetScrollPos(GetDlgItem(hDlg,ID11_S0+i),SB_CTL,3,0);
              SetWindowText(GetDlgItem(hDlg,ID11_T0+i),(LPSTR)"");
            }
          }
          if (nHorzAuswahl == SAMPLE)
          {
            wsprintf((LPSTR)dummy,(LPSTR)"Sample pro Teil: %i",balkenmerker);
            SetWindowText(GetDlgItem(hDlg,ID11_SAMPLETEXT),(LPSTR)dummy);
            nSampleProTeil = balkenmerker;
          }
          return TRUE;
    }
    return FALSE;
}//NeuesNetz

/*************************************************************************
 Funktion:      NeuronenDaten

 Aufgabe:       Wird bei Doppelklick auf Neuron aufgerufen und gibt Auskunft
                über die internen Neuronendaten

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*************************************************************************/


BOOL FAR PASCAL NeuronenDaten(HWND      hDlg,
                              unsigned  message,
                              WORD      wParam,
                              LONG      lParam)

{
int         k;
double      dDummy;
static  int oldbalkenmerker;

  switch (message){
    case WM_INITDIALOG:
          if(nAktuelli>0)
            SetScrollRange(GetDlgItem(hDlg,ID13_SCROLLBAR),SB_CTL,1,
              Netzform[nAktuelli-1],0);
          else
            SetScrollRange(GetDlgItem(hDlg,ID13_SCROLLBAR),SB_CTL,1,1,0);
          SetScrollPos(GetDlgItem(hDlg,ID13_SCROLLBAR),SB_CTL,1,1);
          SetDlgItemText(hDlg,ID13_WTEXT,(LPSTR)"W1:");
          sprintf(dummy,"%lf",Netz[nAktuellj][nAktuelli].xin[0].w);
```

```
      SetDlgItemText(hDlg,ID13_WEDIT,(LPSTR)dummy);
      sprintf(dummy,"%lf",Netz[nAktuellj][nAktuelli].ausgang);
      SetDlgItemText(hDlg,ID13_AUSGANG,(LPSTR)dummy);
      sprintf(dummy,"%lf",Netz[nAktuellj][nAktuelli].p);
      SetDlgItemText(hDlg,ID13_POTENTIAL,(LPSTR)dummy);
      oldbalkenmerker = 1;
      return (TRUE);
case WM_COMMAND:
      switch(wParam)
      {
        case IDCANCEL:
        case IDOK:
             if(nAktuelli>0)
             {
                 GetDlgItemText(hDlg,ID13_WEDIT,(LPSTR)dummy,25);
                 dDummy=atof((LPSTR)dummy);
                 Netz[nAktuellj][nAktuelli].xin[oldbalkenmerker-1].w=dDummy;
             }
             else
             {
               GetDlgItemText(hDlg,ID13_WEDIT,(LPSTR)dummy,25);
               dDummy=atof((LPSTR)dummy);
               Netz[nAktuellj][nAktuelli].xin[0].w = dDummy;
             }
             EndDialog(hDlg, TRUE);
             return (TRUE);
       }
       break;
case WM_VSCROLL:
      balkenmerker = GetScrollPos(HIWORD(lParam),SB_CTL);
      GetDlgItemText(hDlg,ID13_WEDIT,
       (LPSTR)dummy,20);
      dDummy = atof((LPSTR)dummy);
      Netz[nAktuellj][nAktuelli].xin[oldbalkenmerker-1].w=dDummy;
      switch(wParam){
        case SB_LINEDOWN:
             if(balkenmerker<Netzform[nAktuelli-1])
               ++balkenmerker;
             break;
        case SB_LINEUP:
             if(balkenmerker>1)
               --balkenmerker;
             break;
        case SB_PAGEDOWN:
             if(balkenmerker<Netzform[nAktuelli-1])
               ++balkenmerker;
             break;
        case SB_PAGEUP:
             if(balkenmerker>1)
               --balkenmerker;
             break;
        case SB_BOTTOM:
             balkenmerker=1;
             break;
        case SB_TOP:
             balkenmerker=Netzform[nAktuelli-1];
             break;
        case SB_THUMBTRACK:
             balkenmerker=LOWORD(lParam);
             break;
      }
      SetScrollPos(HIWORD(lParam),SB_CTL,balkenmerker,TRUE);
      wsprintf((LPSTR)dummy,(LPSTR)"W%i:",balkenmerker);
```

```
        SetDlgItemText(hDlg,ID13_WTEXT,(LPSTR)dummy);
        sprintf(dummy,"%lf",Netz[nAktuellj][nAktuelli].xin[balkenmerker-1].w);
        SetDlgItemText(hDlg,ID13_WEDIT,(LPSTR)dummy);
        oldbalkenmerker = balkenmerker;
        break;
  }
  return (FALSE);
}//NeuronenDaten

/*************************************************************************
 Funktion:      TeileDaten

 Aufgabe:       In dieser DialogBox werden die spezifischen Daten der Teile
                erfragt

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*************************************************************************/


BOOL FAR PASCAL TeileDaten(HWND     hDlg,
                           unsigned message,
                           WORD     wParam,
                           LONG     lParam)

{
int   i,j,k;
int   xZufall;
int   yZufall;
int   pZufall;
char  OldPosition[50];

static int oldbalkenmerker;
char Position[60];

  switch (message){
    case WM_INITDIALOG:
         SetScrollRange(GetDlgItem(hDlg,ID15_SCROLLTEILE),SB_CTL,1,nTeile,0);
         SetScrollPos(GetDlgItem(hDlg,ID15_SCROLLTEILE),SB_CTL,1,1);
         SetScrollRange(GetDlgItem(hDlg,ID15_SCROLLSAMPLE),SB_CTL,1,
          nSampleProTeil,0);
         SetScrollPos(GetDlgItem(hDlg,ID15_SCROLLSAMPLE),SB_CTL,1,1);
         SetDlgItemText(hDlg,ID15_TEILENUMMER,(LPSTR)"Teilenummer: 1");
         SetDlgItemText(hDlg,ID15_SAMPLENUMMER,(LPSTR)"Samplenummer: 1");
         SetDlgItemText(hDlg,ID15_NAME,(LPSTR)TeileName[0]);
         SetDlgItemText(hDlg,ID15_GROESSE,(LPSTR)TeileGroesse[0]);
         SetDlgItemText(hDlg,ID15_FACHNUMMER,(LPSTR)TeileFach[0]);
         oldbalkenmerker=1;
         return (TRUE);
    case WM_COMMAND:
         switch(wParam)
         {
           case IDCANCEL:
           case IDOK:
                GetDlgItemText(hDlg,ID15_NAME,
                  (LPSTR)TeileName[oldbalkenmerker-1],20);
                GetDlgItemText(hDlg,ID15_GROESSE,
```

```
        (LPSTR)TeileGroesse[oldbalkenmerker-1],9);
      GetDlgItemText(hDlg,ID15_FACHNUMMER,
        (LPSTR)TeileFach[oldbalkenmerker-1],9);
      ndummy=atoi(TeileFach[oldbalkenmerker-1]);
      if(ndummy<0 || ndummy>999)
        lstrcpy((LPSTR)TeileFach[oldbalkenmerker-1],(LPSTR)"999");
      InvalidateRect(hWndBrain,NULL,TRUE);
      PostMessage(hWndBrain,WM_PAINT,0,0L);
      bSpeichernAbfrage=TRUE;
      ErstelleMinMaxVektoren();
      EndDialog(hDlg, TRUE);
      return (TRUE);
    case ID15_SAMPLEAUFNEHMEN:
      if(bVideomatInit)
      {
        i=GetScrollPos(GetDlgItem(hDlg,ID15_SCROLLTEILE),
          SB_CTL)-1;
        j=GetScrollPos(GetDlgItem(hDlg,ID15_SCROLLSAMPLE),
          SB_CTL)-1;
        HoleEingangsVektor();
        SetzeEingangsTrainingsVektor(i*nSampleProTeil+j);
        AusgangsTrainingsVektor[i][i*nSampleProTeil+j]=1;
        ErstelleMinMaxVektoren();
        TransformiereEingangsVektor();
        SendMessage(hDlg,WM_HSCROLL,SB_LINEDOWN,
          MAKELONG(0,GetDlgItem(hDlg,ID15_SCROLLSAMPLE)));
      }
      else
        MessageBox(GetFocus(),
          (LPSTR)"Videomat ist noch nicht initialisiert",
          NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);

      break;
    case ID15_GREIFHOEHE:
      HoleRoboterPosition((LPSTR)Position,(LPSTR)dummy);
      ndummy=0;
      while(Position[ndummy]!=',')
        ++ndummy;
        ++ndummy;
      while(Position[ndummy]!=',')
        ++ndummy;
      ++ndummy;
      strcpy(dummy,&(Position[ndummy]));
      ndummy=0;
      while(dummy[ndummy]!=',')
        ++ndummy;
      dummy[ndummy]='\0';
      i=GetScrollPos(GetDlgItem(hDlg,ID15_SCROLLTEILE),
        SB_CTL)-1;
      lstrcpy((LPSTR)TeileHoehe[i],(LPSTR)dummy);
      break;
    case ID15_STARTEROBOTER:
      if(bVideomatInit)
      {
        if(nRoboterEinsatz==KEIN)
          MessageBox(hDlg,
            (LPSTR)"Robotereinsatz wurde nicht angewählt",
            NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        else
        {
          if(bMoveMasterInit)
          {
            i=GetScrollPos(GetDlgItem(hDlg,ID15_SCROLLTEILE),
```

```
      SB_CTL)-1;
    MO("999");
    wsprintf((LPSTR)dummy,(LPSTR)"0,400,%i",
      atoi(TeileHoehe[i]));
    AbsolutRelativ();
    SendeComm1(nCid,dummy);
    GC();
    MO("998");
    MO("997");
    for(j=GetScrollPos(GetDlgItem(hDlg,ID15_SCROLLSAMPLE),
          SB_CTL)-1;j<nSampleProTeil;++j)
    {
      xZufall=ROBOTERRANDOM*
        (200-2*atoi(TeileGroesse[i]))/2+xCent;
      yZufall=ROBOTERRANDOM*
        (260-2*atoi(TeileGroesse[i]))/2+yCent;
      pZufall=ROBOTERRANDOMP;
      if(j==0)
      {
        xZufall=xCent;
        yZufall=yCent;
        pZufall=0;
      }
      wsprintf((LPSTR)dummy,(LPSTR)"0,0,0,0,%i",-pZufall);
      MJ((LPSTR)dummy);
      wsprintf((LPSTR)dummy,(LPSTR)"%i,%i,%i",
        xZufall,yZufall,atoi(TeileHoehe[i]));
      AbsolutRelativ();
      SendeComm1(nCid,dummy);
      HE("995");
      GO();
      wsprintf((LPSTR)dummy,(LPSTR)"%i,%i,%i",
        xZufall,yZufall,atoi(TeileHoehe[i])+40);
      AbsolutRelativ();
      SendeComm1(nCid,dummy);
      HE("994");
      MO("993");
      HoleEingangsVektor();
      SetzeEingangsTrainingsVektor(i*nSampleProTeil+j);
      AusgangsTrainingsVektor[i][i*nSampleProTeil+j]=1;
      ErstelleMinMaxVektoren();
      TransformiereEingangsVektor();
      HolePartSchwerpunkt();
      wsprintf((LPSTR)dummy,(LPSTR)"Samplenummer: %i",j+1);
      MO("994,O");
      MO("995,O");
      GC();
      MO("997");
      SendMessage(hDlg,WM_HSCROLL,SB_LINEDOWN,
        MAKELONG(0,GetDlgItem(hDlg,ID15_SCROLLSAMPLE)));
    }
    SendMessage(hDlg,WM_HSCROLL,SB_BOTTOM,
      MAKELONG(0,GetDlgItem(hDlg,ID15_SCROLLSAMPLE)));
    MO("998");
    wsprintf((LPSTR)dummy,(LPSTR)"0,0,%i",
      -200+atoi(TeileHoehe[i]));
    DS((LPSTR)dummy);
    GO();
    MO("999");
    SendMessage(hDlg,WM_HSCROLL,SB_LINEDOWN,
      MAKELONG(0,GetDlgItem(hDlg,ID15_SCROLLTEILE)));
  }
  else
```

```
                    MessageBox(GetFocus(),
                       (LPSTR)"Movemaster ist noch nicht initialisiert",
                       NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
                  }
                }
                else
                  MessageBox(GetFocus(),
                     (LPSTR)"Videomat ist noch nicht initialisiert",
                     NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);

                break;
        }
        break;
  case WM_HSCROLL:
        balkenmerker = GetScrollPos(HIWORD(lParam),SB_CTL);
        GetDlgItemText(hDlg,ID15_NAME,
         (LPSTR)TeileName[oldbalkenmerker-1],20);
        GetDlgItemText(hDlg,ID15_GROESSE,
         (LPSTR)TeileGroesse[oldbalkenmerker-1],9);
        GetDlgItemText(hDlg,ID15_FACHNUMMER,
          (LPSTR)TeileFach[oldbalkenmerker-1],9);
        ndummy=atoi(TeileFach[oldbalkenmerker-1]);
        if(ndummy<0 || ndummy>999)
          lstrcpy((LPSTR)TeileFach[oldbalkenmerker-1],(LPSTR)"999");
        if(HIWORD(lParam)==GetDlgItem(hDlg,ID15_SCROLLTEILE))
        {
          ndummy=nTeile;
        }
        else
        {
          ndummy=nSampleProTeil;
        }
        switch(wParam){
          case SB_LINEDOWN:
                if(balkenmerker<ndummy)
                  ++balkenmerker;
                break;
          case SB_LINEUP:
                if(balkenmerker>1)
                  --balkenmerker;
                break;
          case SB_PAGEDOWN:
                if(balkenmerker<ndummy)
                  ++balkenmerker;
                break;
          case SB_PAGEUP:
                if(balkenmerker>1)
                  --balkenmerker;
                break;
          case SB_BOTTOM:
                balkenmerker=1;
                break;
          case SB_TOP:
                balkenmerker=ndummy;
                break;
          case SB_THUMBTRACK:
                balkenmerker=LOWORD(lParam);
                break;
        }
        if(HIWORD(lParam)==GetDlgItem(hDlg,ID15_SCROLLTEILE))
        {
          SetScrollPos(HIWORD(lParam),SB_CTL,balkenmerker,TRUE);
          wsprintf((LPSTR)dummy,(LPSTR)"Teilenummer: %i",balkenmerker);
```

```
            SetDlgItemText(hDlg,ID15_TEILENUMMER,(LPSTR)dummy);
            SetDlgItemText(hDlg,ID15_NAME,(LPSTR)TeileName[balkenmerker-1]);
            SetDlgItemText(hDlg,ID15_GROESSE,
              (LPSTR)TeileGroesse[balkenmerker-1]);
            SetDlgItemText(hDlg,ID15_FACHNUMMER,
              (LPSTR)TeileFach[balkenmerker-1]);
            oldbalkenmerker = balkenmerker;
          }
          else
          {
            SetScrollPos(HIWORD(lParam),SB_CTL,balkenmerker,TRUE);
            wsprintf((LPSTR)dummy,(LPSTR)"Samplenummer: %i",balkenmerker);
            SetDlgItemText(hDlg,ID15_SAMPLENUMMER,(LPSTR)dummy);
          }
          break;
  }
  return (FALSE);
}//TeileDaten

/*****************************************************************************
 Funktion:        AbsolutRelativ

 Aufgabe:         berechnet den String dummy von absoluten Roboterkoordinaten
                  in relative um

 Parameter:

 Rückgabewert:

*****************************************************************************/

void AbsolutRelativ(void)
{
char hilf[50];
char hilf2[50];
int  i,j;
int  xa,ya,za;
int  xh,yh,zh;

  i=0;
  j=0;
  lstrcpy((LPSTR)hilf,(LPSTR)dummy);
  while(hilf[i]!=',')
  {
    hilf2[j]=hilf[i];
    ++i;
    ++j;
  }
  hilf2[j]='\0';
  xa=atoi(hilf2);
  ++i;
  j=0;
  while(hilf[i]!=',')
  {
    hilf2[j]=hilf[i];
    ++i;
    ++j;
  }
  hilf2[j]='\0';
  ya=atoi(hilf2);
  ++i;
  j=0;
  while(hilf[i]!='\0')
```

```
  {
    hilf2[j]=hilf[i];
    ++i;
    ++j;
  }
  hilf2[j]='\0';
  za=atoi(hilf2);
  HoleRoboterPosition((LPSTR)hilf,(LPSTR)hilf2);
  i=0;
  j=0;
  while(hilf[i]!=',')
  {
    hilf2[j]=hilf[i];
    ++i;
    ++j;
  }
  hilf2[j]='\0';
  xh=atoi(hilf2);
  ++i;
  j=0;
  while(hilf[i]!=',')
  {
    hilf2[j]=hilf[i];
    ++i;
    ++j;
  }
  hilf2[j]='\0';
  yh=atoi(hilf2);
  ++i;
  j=0;
  while(hilf[i]!='\0')
  {
    hilf2[j]=hilf[i];
    ++i;
    ++j;
  }
  hilf2[j]='\0';
  zh=atoi(hilf2);
  wsprintf((LPSTR)dummy,(LPSTR)"DS %i,%i,%i\n",xa-xh,ya-yh,za-zh);
}//AbsolutRelativ
```

7.4.4. N_FILE.C

```
/*****************************************************************************
 Modul:       N_FILE.C

 Aufgabe:     Stellt eine Dateiauswahlbox zur Verfügung

 Funktionen:  DateiLaden
*****************************************************************************/

#include "windows.h"
#include "neurorob.h"
#include "n_extern.h"
#include "n_file.h"

/*****************************************************************************
 Funktion:      DateiLaden

 Aufgabe:       Dateiauswahldialogbox darstellen

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*****************************************************************************/

BOOL FAR PASCAL DateiLaden(HWND     hDlg,
                           unsigned message,
                           WORD     wParam,
                           LONG     lParam)
{
  switch (message){
    case WM_INITDIALOG:
         SendMessage(hDlg,WM_SETTEXT,NULL,(LPSTR)szFileCaption);
         wsprintf((LPSTR)dummy,(LPSTR)"*%s",(LPSTR)DefExt);
         SetDlgItemText(hDlg,ID6_FILE,(LPSTR)dummy);
         DlgDirList(hDlg,(LPSTR)dummy,ID6_FILES,0,0x0000);
         DlgDirList(hDlg,(LPSTR)"D:\*.*",
          ID6_DIRECTORIES,ID6_DIRECTORY,0xC010);
         SendDlgItemMessage(hDlg,ID6_FILE,EM_SETSEL,NULL,MAKELONG(0,0x7fff));
         SetFocus(GetDlgItem(hDlg,ID6_FILE));
         return (FALSE);
    case WM_COMMAND:
         switch(wParam){
           case IDOK:
                SendDlgItemMessage(hDlg,ID6_FILE,WM_GETTEXT,13,
                  (LPSTR)FileName);
                if((strchr(FileName,'*')==NULL)&&(strchr(FileName,'?')==NULL))
                {
                  EndDialog(hDlg,TRUE);
                  return(TRUE);
                }
                else
                {
                  DlgDirList(hDlg,(LPSTR)FileName,ID6_FILES,0,0x0000);
                }
                break;
           case IDCANCEL:
```

```
                EndDialog(hDlg, FALSE);
                return (TRUE);
           case ID6_DIRECTORIES:
                switch(HIWORD(lParam)){
                   case LBN_DBLCLK:
                        DlgDirSelect(hDlg,str,ID6_DIRECTORIES);
                        strcat(str,"*");
                        strcat(str,DefExt);
                        DlgDirList(hDlg,(LPSTR)str,ID6_FILES,0,0x0000);
                        DlgDirList(hDlg,(LPSTR)str,ID6_DIRECTORIES,
                          ID6_DIRECTORY,0xC010);
                        PostMessage(hDlg,WM_COMMAND,IDOK,0L);
                        break;
                }
                break;
           case ID6_FILES:
                switch(HIWORD(lParam)){
                   case LBN_SELCHANGE:
                        DlgDirSelect(hDlg,str,ID6_FILES);
                        SetDlgItemText(hDlg,ID6_FILE,(LPSTR)str);
                        strcpy(FileName,str);
                        break;
                   case LBN_DBLCLK:
                        DlgDirSelect(hDlg,str,ID6_FILES);
                        SetDlgItemText(hDlg,ID6_FILE,(LPSTR)str);
                        strcpy(FileName,str);
                        PostMessage(hDlg,WM_COMMAND,IDOK,NULL);
                        break;
                }
                break;
          }
          break;
  }
  return (FALSE);
}//DateiLaden
```

7.4.5. N_LIST.C

```
/*****************************************************************************
 Modul:       N_LIST.C

 Aufgabe:     Bearbeitet den Hauptmenuepunkt "Befehlsliste"

 Funktionen:  Befehlsliste
              Befehlseingabe
*****************************************************************************/

#include "windows.h"
#include "neurorob.h"
#include "n_extern.h"
#include "ndl_list.h"
#include "ndl_visu.h"
#include "neuromen.h"
#include "ndl_edit.h"
#include <stdio.h>

#define  ID7_ADD 799            //Sendet die Befehlseingabe

BOOL FAR PASCAL Befehlseingabe( HWND,unsigned,WORD,LONG);

BOOL    bBefehlseingabe = FALSE;
BOOL    bListeSpeichern = FALSE;
FARPROC lpBefehlseingabe;
HWND    hDlgBefehlseingabe;


/*****************************************************************************
 Funktion:      Befehlsliste

 Aufgabe:       Darstellen und bearbeiten der Befehlsliste-Dialogbox

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  msg    : Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*****************************************************************************/

BOOL FAR PASCAL Befehlsliste(HWND hDlg,unsigned msg,WORD wParam,LONG lParam)
{
FILE    *FilePointer;
FARPROC lpProcBefehlseingabe;
char    robpos[40];
char    befehl[40];
char    grid[6];
char    Listenbefehl[60];
int     buffer[1001];
int     nAbbruch;
DWORD   select;
DWORD   ldummy;

  switch(msg){
    case WM_INITDIALOG:
         if(bRepeatmodus)
           CheckDlgButton(hDlg,ID3_REPEAT,(WORD) 1);
```

```
        if(bEinzelschrittmodus)
          CheckDlgButton(hDlg,ID3_SINGLESTEP,(WORD) 1);
        bBefehlseingabe = FALSE;
        bListeSpeichern = FALSE;
        return(TRUE);
    case WM_COMMAND:
        switch (wParam){
          case IDOK:
               PostMessage(hDlg,WM_SYSCOMMAND,SC_CLOSE,0L);
               break;
          case ID3_ADD:
               HoleRoboterPosition((LPSTR)robpos,(LPSTR)grid);
               befehl[0]='\0';
               lstrcat((LPSTR)befehl,(LPSTR)"MP ");
               lstrcat((LPSTR)befehl,(LPSTR)robpos);
               ndummy=SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_ADDSTRING,
                 0,(DWORD)(LPSTR)befehl);
               if((ndummy==LB_ERR)||(ndummy==LB_ERRSPACE))
                 MessageBox(GetFocus(),(LPSTR)"Fehler beim Hinzufuegen",
                    (LPSTR)"Liste",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
               ndummy=SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_ADDSTRING,
                 0,(DWORD)(LPSTR)grid);
               if((ndummy==LB_ERR)||(ndummy==LB_ERRSPACE))
                 MessageBox(GetFocus(),(LPSTR)"Fehler beim Hinzufuegen",
                    (LPSTR)"Liste",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
               bListeSpeichern = TRUE;
               break;
          case ID7_ADD:    //Kommt vom editfenster
               ndummy=SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_ADDSTRING,
                 0,(DWORD)(LPSTR)lParam);
               bListeSpeichern = TRUE;
               break;
          case ID3_COMMAND:
               if(!bBefehlseingabe)
               {
                 lpBefehlseingabe = MakeProcInstance(Befehlseingabe,hInst);
                 hDlgBefehlseingabe = CreateDialog(hInst,"BEFEHLSEINGABE",
                   hDlg,lpBefehlseingabe);
                 CheckDlgButton(hDlg,ID3_COMMAND,(WORD) 1);
                 bBefehlseingabe = TRUE;
               }
               else
               {
                 DestroyWindow(hDlgBefehlseingabe);
                 FreeProcInstance(lpBefehlseingabe);
                 bBefehlseingabe = FALSE;
                 CheckDlgButton(hDlg,ID3_COMMAND,(WORD) 0);
                 bRepeatmodus = TRUE;
                 hDlgBefehlseingabe = NULL;
               }
               break;
          case ID3_DEL:
               if(MessageBox(GetFocus(),(LPSTR)"Wirklich loeschen",
                 (LPSTR)"",MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL)==IDYES)
               {
                 while((select = SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,
                           LB_GETSELITEMS,100,(DWORD)buffer))!=0)
                 {
                     if(SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,
                       LB_DELETESTRING,(WORD)buffer[0],0L)==LB_ERR)
                           MessageBox(GetFocus(),(LPSTR)"Fehler beim Löschen",
                             (LPSTR)"Liste",
                             MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
```

```
        }
      }
      bListeSpeichern = TRUE;
      break;
case ID3_SELECTALL:
      SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_SETSEL,1,-1L);
      break;
case ID3_UNSELECTALL:
      SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_SETSEL,0,-1L);
      break;
case ID3_LOAD:
      if(SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_GETCOUNT,0,0L)
        &&bListeSpeichern)
        SendMessage(hDlg,WM_COMMAND,ID3_SAVE,0L);
      strcpy(DefExt,".TXT");
      szFileCaption = (LPSTR)"Befehlsliste laden";
      lpProcLaden = MakeProcInstance(DateiLaden, hInst);
      if(!DialogBox(hInst,"FILEOPERATION",hDlg,lpProcLaden))
        break;
      FreeProcInstance(lpProcLaden) ;
      FilePointer=fopen(FileName,"rt");
      if(!FilePointer)
      {
        MessageBox(GetFocus(),"Datei konnte nicht geoeffnet werden",
          "Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        return (FALSE);
      }
      ndummy=0;
      do
      {
        if (fgets(dummy, 90 , FilePointer) == NULL)
          break;
        dummy[strlen(dummy)-1] = '\0';
        ++ndummy;
        ndummy=SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,
          LB_ADDSTRING,0,(DWORD)(LPSTR)dummy);
        if((ndummy==LB_ERR)||(ndummy==LB_ERRSPACE))
          MessageBox(GetFocus(),(LPSTR)
            "Fehler beim Laden einer Liste",
            (LPSTR)"Liste",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
      } while (TRUE);
      fclose(FilePointer);
      bListeSpeichern = FALSE;
      break;
case ID3_SAVE:
      SendMessage(hDlg,WM_COMMAND,ID3_SELECTALL,0L);
      if(SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_GETCOUNT,0,0L)
        ==0)
      {
        MessageBox(GetFocus(),
          (LPSTR)"Befehlsliste leer",
          (LPSTR)"Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        break;
      }
      strcpy(DefExt,".TXT");
      szFileCaption = (LPSTR)"Befehlsliste speichern";
      lpProcLaden = MakeProcInstance(DateiLaden, hInst);
      if(!DialogBox(hInst,"FILEOPERATION",hDlg,lpProcLaden))
        break;
      FreeProcInstance(lpProcLaden) ;
      FilePointer=fopen(FileName,"wt");
      if(!FilePointer)
      {
```

```
        MessageBox(GetFocus(),"Datei konnte nicht geoeffnet werden",
          "Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        return (FALSE);
      }
      select = SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,
                LB_GETSELITEMS,1000,(DWORD)buffer);
      if(select>0L)
      {

        for(ldummy=0;ldummy<select;++ldummy)
        {
          if(SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_GETTEXT,
            (WORD)buffer[(int)ldummy],(DWORD)dummy)==LB_ERR)
              MessageBox(GetFocus(),(LPSTR)"Fehler beim Textholen",
                (LPSTR)"Liste",
                MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
          lstrcat((LPSTR)dummy,(LPSTR)"\n");
          fputs(dummy,FilePointer);
        }
      }
      fclose(FilePointer);
      bListeSpeichern = FALSE;
      break;
    case ID3_DO:
        select = SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,
                  LB_GETSELITEMS,1001,(DWORD)buffer);
        if(select == 1001)
        {
          MessageBox(hDlg,(LPSTR)"Max. 1000 Befehle",NULL,
            MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
          break;
        }
        if(select ==  0)
          MessageBox(GetFocus(),
            (LPSTR)"Es wurden noch keine Befehle markiert",
            (LPSTR)"Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        if(select>0L)
        {

          for(ldummy=0;ldummy<select;++ldummy)
          {
            if(SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_GETTEXT,
              (WORD)buffer[(int)ldummy],(DWORD)Listenbefehl)==LB_ERR)
              MessageBox(GetFocus(),(LPSTR)"Fehler beim Textholen",
              (LPSTR)"Liste",
                MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
            if(bEinzelschrittmodus)
            {
              nAbbruch=MessageBox(GetFocus(),(LPSTR)Listenbefehl,
                (LPSTR)"Sende Befehl",MB_YESNOCANCEL|
                MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
              if(nAbbruch!=IDYES)
                break;
            }
            lstrcat((LPSTR)Listenbefehl,(LPSTR)"\n");
            SendeComm1(nCid,Listenbefehl);
            if(GetAsyncKeyState(VK_RBUTTON)==0x8000)
              break;
            SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_SETTOPINDEX,
              (WORD)buffer[(int)ldummy],0L);
            if(bVisualisierung)
              SendMessage(hDlgVisualisierung,WM_COMMAND,
                ID10_AKTUELL,0L);
```

```
                    }
                  }
                if((nAbbruch!=IDCANCEL)&&bRepeatmodus&&
                  (GetAsyncKeyState(VK_RBUTTON)!=0x8000))
                    PostMessage(hDlg,WM_COMMAND,ID3_DO,0L);
                break;
          case ID3_REPEAT:
                if(bRepeatmodus == FALSE)
                {
                  CheckDlgButton(hDlg,ID3_REPEAT,(WORD) 1);
                  bRepeatmodus = TRUE;
                }
                else
                {
                  CheckDlgButton(hDlg,ID3_REPEAT,(WORD) 0);
                  bRepeatmodus = FALSE;
                }
                break;
          case ID3_SINGLESTEP:
                if(bEinzelschrittmodus == FALSE)
                {
                  CheckDlgButton(hDlg,ID3_SINGLESTEP,(WORD) 1);
                  bEinzelschrittmodus = TRUE;
                }
                else
                {
                  CheckDlgButton(hDlg,ID3_SINGLESTEP,(WORD) 0);
                  bEinzelschrittmodus = FALSE;
                }
                break;
          default:
                return FALSE;
        }
        break;
    case WM_SYSCOMMAND:
        switch(wParam){
          case SC_CLOSE:
                if(SendDlgItemMessage(hDlg,ID3_BEFEHLSLISTE,LB_GETCOUNT,0,0L)
                  &&bListeSpeichern)
                  SendMessage(hDlg,WM_COMMAND,ID3_SAVE,0L);
                DestroyWindow(hDlgBefehlsliste);
                FreeProcInstance(lpBefehlsliste);
                bBefehlsliste = FALSE;
                hDlgBefehlsliste = NULL;
                CheckMenuItem(hMainMenu,IDM_BEFEHLSLISTE,MF_UNCHECKED);
                return TRUE;
          default:
                return FALSE;
        }
        break;
    default:
        return FALSE;
  }
  return FALSE;
}//BefehlsListe

/***************************************************************************
 Funktion:      Befehlseingabe

 Aufgabe:       Befehlseingabedialogbox darstellen

 Parameter:
  HWND      hDlg   : Fensterhandle
```

```
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*************************************************************************/

BOOL FAR PASCAL Befehlseingabe( HWND      hDlg,
                                unsigned  message,
                                WORD      wParam,
                                LONG      lParam)
{
  switch (message){
    case WM_INITDIALOG:
         return (TRUE);
    case WM_COMMAND:
         switch(wParam){
           case IDOK:
                SendDlgItemMessage(hDlg,ID7_EDIT,WM_GETTEXT,40,(DWORD)dummy);
                SendMessage(hDlgBefehlsliste,WM_COMMAND,ID7_ADD,
                  (DWORD)(LPSTR)dummy);
                SendDlgItemMessage(hDlg,ID7_EDIT,WM_SETTEXT,0,(LPSTR)"");
                SetFocus(GetDlgItem(hDlg,ID7_EDIT));
                return(TRUE);
           case IDCANCEL:
                SendMessage(hDlgBefehlseingabe,WM_SYSCOMMAND,SC_CLOSE,0L);
                return (TRUE);
         }
         break;
    case WM_SYSCOMMAND:
         switch(wParam){
           case SC_CLOSE:
                DestroyWindow(hDlgBefehlseingabe);
                FreeProcInstance(lpBefehlseingabe);
                bBefehlseingabe = FALSE;
                hDlgBefehlseingabe = NULL;
                CheckDlgButton(hDlgBefehlsliste,ID3_COMMAND,(WORD) 0);
                return TRUE;
           default:
                return FALSE;
         }
         break;
    default:
         return FALSE;
  }
  return FALSE;
}//BefehlsEingabe
```

7.4.6. N_MASTER.C

```
/*****************************************************************************
 Modul:       N_MASTER.C

 Aufgabe:     Macht einige Roboterbefehle für 'C' verfügbar
              Genaue Erläuterung der Aufgaben im Handbuch des Roboters

 Funktionen:  MO
              DS
              DW
              GC
              GP
              GO
              HE
              MA
              MC
              MJ
              MP
              MS
              MP
              NT
              NW
              PD
              RS
              SD
              SP

*****************************************************************************/


#include "windows.h"
#include "neurorob.h"
#include "n_extern.h"


void MO (LPSTR Param)    // Position wird angefahren
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"MO %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//MO

void DS (LPSTR Param)    // Position wird angefahren
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"DS %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//DS

void DW (LPSTR Param)    // Position wird angefahren
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"DW %s\n",Param);
  SendeComm1(nCid,Befehl);
}//DW

void GC (void)    // Schließt die Hand
{
```

```
  SendeComm1(nCid,"GC\n");
}//GC

void GP (LPSTR Param)    // Kraft der Hand
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"GP %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//GP

void GO (void)    // Öffnet die Hand
{
  SendeComm1(nCid,"GO\n");
}//GO

void HE (LPSTR Param)    // Position wird gespeichert
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"HE %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//HE

void MA (LPSTR Param)    // Position wird angefahren
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"MA %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//MA

void MC (LPSTR Param)    // Position wird angefahren
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"MC %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//MC

void MJ (LPSTR Param)    // Winkeländerung
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"MJ %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//MJ

void MP (LPSTR Param)    // Position wird angefahren
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"MP %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//MP

void MS (LPSTR Param)    // Position wird angefahren
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"MS %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//MS
```

```
void MT (LPSTR Param)    // Position wird angefahren
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"MT %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//MT

void NT (void)    // Nestposition
{
  SendeComm1(nCid,"NT\n");
}//NT

void NW (void)    // NEW
{
  SendeComm1(nCid,"NW\n");
}//NW

void PD (LPSTR Param)    // Position wird gespeichert
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"PD %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//PD

void RS (LPSTR Param)    // RESET
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"RS %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//RS

void SD (LPSTR Param)    // Speed TOOL_CENTER POINT
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"SD %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//SD

void SP (LPSTR Param)    // SPEED Roboter
{
char Befehl[50];

  wsprintf((LPSTR)Befehl,(LPSTR)"SP %s\n",(LPSTR)Param);
  SendeComm1(nCid,Befehl);
}//SP
```

7.4.7. N_POSITI.C

```
/*****************************************************************************
 Modul:       N_POSITI

 Aufgabe:     Stellt Funktionen zur Positionsanalyse zur Verfügung

 Funktionen:  Positionsana
*****************************************************************************/

#include <windows.h>
#include "n_extern.h"
#include "ndl_pos.h"
#include "neurorob.h"
#include "string.h"
#include <stdio.h>


char  Befehl[50];
char  Befehl1[50];
char  where[50];
char  feld1[10];
char  feld2[10];
char  feld3[10];
char  feld4[10];
char  feld5[10];
char  feld6[10];
char  feld7[10];
int   i,j,c,anzahl;
int   lenwhere=0;
DWORD index;

/*****************************************************************************
 Funktion:      Positionsana

 Aufgabe:       Darstellung der Dialogbox für die Positionsanalyse

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*****************************************************************************/

BOOL FAR PASCAL Positionsana (HWND      hDlg,
                              unsigned  message,
                              WORD      wParam,
                              LONG      lParam)
{
COMSTAT   CommStatus;
int       nFehler;
FILE      *FilePointer;

  switch(message){
    case WM_INITDIALOG:
         return(TRUE);
    case WM_COMMAND:
         switch(wParam){
           case IDOK:
```

```
case IDCANCEL:
     EndDialog(hDlg, FALSE);
     return(TRUE);
case ID8_ROBOTER:
     SendDlgItemMessage(hDlg,ID8_POSITION,LB_RESETCONTENT,0,0);
     SetCursor(LoadCursor(NULL,IDC_WAIT));
     for(i=1;i<=999;i++)
     {
       itoa(i,dummy,10);
       wsprintf((LPSTR)Befehl,(LPSTR)"PR %s\n",(LPSTR)dummy);
       FlushComm(nCid,1);
       lenwhere=0;
       Sende(nCid,Befehl);
       where[0]='\0';
       do
       {
         ndummy=ReadComm(nCid,(LPSTR)dummy,50);
         dummy[ndummy]='\0';
         lstrcat((LPSTR)where,(LPSTR)dummy);
         lenwhere+=ndummy;
         nFehler = GetCommError(nCid,&CommStatus);
         if (nFehler>0)
           MessageBox(GetFocus(),(LPSTR)"Comm",(LPSTR)"FEHLER",
             MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
       }while(dummy[ndummy-1]!='\n');
       where[lenwhere-3] = '\0';
       ndummy = lstrlen((LPSTR)where);
       if(ndummy>11)
       {
         itoa(i,dummy,10);
         ndummy = SendDlgItemMessage(hDlg,ID8_POSITION,
           LB_ADDSTRING,0,(LPSTR)dummy);
       }
     }
     SetCursor(LoadCursor(NULL,IDC_ARROW));
     return(TRUE);
case ID8_POSITION:
     if(HIWORD(lParam)!=LBN_SELCHANGE)
       break;
     index=SendDlgItemMessage(hDlg,ID8_POSITION,LB_GETCURSEL,0,0);
     if (index == LB_ERR)
       MessageBox(GetFocus(),(LPSTR)"Kein Eintrag",(LPSTR)"DATEN",
         MB_OK|MB_SYSTEMMODAL);
     else
     {
       SendDlgItemMessage(hDlg,ID8_POSITION,LB_GETTEXT,index,
         (LPSTR)dummy);
       SetDlgItemText(hDlg,ID8_POSITIONSNR,(LPSTR)dummy);
       wsprintf((LPSTR)Befehl,(LPSTR)"PR %s\n",(LPSTR)dummy);
       FlushComm(nCid,1);
       Sende(nCid,(LPSTR)Befehl);
       where[0]='\0';
       do
       {
         ndummy=ReadComm(nCid,(LPSTR)dummy,50);
         dummy[ndummy]='\0';
         lstrcat((LPSTR)where,(LPSTR)dummy);
         lenwhere+=ndummy;
         nFehler = GetCommError(nCid,&CommStatus);
         if (nFehler>0)
           MessageBox(GetFocus(),(LPSTR)"Comm",NULL,
             MB_OK|MB_SYSTEMMODAL);
       }while(dummy[ndummy-1]!='\n');
```

```
  where[lenwhere-3] = '\0';
  i=0;
  c=0;
  while(where[i]!=',')
  {
    feld1[c] = where[i];
    i++;
    c++;
  }
  feld1[c]='\0';
  c=0;
  i++;
  while(where[i]!=',')
  {
    feld2[c] = where[i];
    i++;
    c++;
  }
  feld2[c]='\0';
  c=0;
  i++;
  while(where[i]!=',')
  {
    feld3[c] = where[i];
    i++;
    c++;
  }
  feld3[c]='\0';
    c=0;
    i++;
  while(where[i]!=',')
  {
    feld4[c] = where[i];
    i++;
    c++;
  }
  feld4[c]='\0';
  c=0;
  i++;
  while(where[i]!=',')
  {
    feld5[c] = where[i];
    i++;
    c++;
  }
  feld5[c]='\0';
  c=0;
  i++;
  while(where[i]!='\0')
  {
    feld6[c] = where[i];
    i++;
    c++;
  }
  feld6[c]='\0';
  SetDlgItemText(hDlg,ID8_XWERT,(LPSTR)feld1);
  SetDlgItemText(hDlg,ID8_YWERT,(LPSTR)feld2);
  SetDlgItemText(hDlg,ID8_ZWERT,(LPSTR)feld3);
  SetDlgItemText(hDlg,ID8_HANDPITCH,(LPSTR)feld4);
  SetDlgItemText(hDlg,ID8_HANDROLL,(LPSTR)feld5);
  SetDlgItemText(hDlg,ID8_HANDGRIP,(LPSTR)feld6);
 }
break;
```

```
case ID8_MOVE:
     SendMessage(hDlg,WM_COMMAND,ID8_DATEN,0L);
     index=SendDlgItemMessage(hDlg,ID8_POSITION,LB_GETCURSEL,0,0);
     if(index == LB_ERR)
     {
       MessageBox(GetFocus(),(LPSTR)"Kein Eintrag",(LPSTR)"DATEN",
         MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
     }
     else
     {
       SendDlgItemMessage(hDlg,ID8_POSITION,LB_GETTEXT,index,
         (LPSTR)dummy);
       SetDlgItemText(hDlg,ID8_POSITIONSNR,(LPSTR)dummy);
       wsprintf((LPSTR)Befehl,(LPSTR)"MO %s,%s\n",(LPSTR)dummy,
         (LPSTR)feld6);
       FlushComm(nCid,1);
       SendeComm1(nCid,Befehl);
     }
     break;
case ID8_NEW:
     index=SendDlgItemMessage(hDlg,ID8_POSITION,LB_GETCURSEL,0,0);
     if(index==LB_ERR)
     {
       MessageBox(GetFocus(),(LPSTR)"Kein Eintrag",(LPSTR)"DATEN",
         MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
     }
     else
     {
       SendDlgItemMessage(hDlg,ID8_POSITION,LB_GETTEXT,index,
         (LPSTR)dummy);
       SendDlgItemMessage(hDlg,ID8_POSITION,LB_DELETESTRING,index,
         0);
       SetDlgItemText(hDlg,ID8_POSITIONSNR,(LPSTR)dummy);
       wsprintf((LPSTR)Befehl,(LPSTR)"PC %s\n",(LPSTR)dummy);
       FlushComm(nCid,1);
       SendeComm1(nCid,Befehl);
     }
     break;
case ID8_SAVE:
     lstrcpy((LPSTR)DefExt,(LPSTR)".POS");
     szFileCaption = (LPSTR)"Positionsdaten speichern";
     lpProcLaden = MakeProcInstance(DateiLaden, hInst);
     if(!DialogBox(hInst,"FILEOPERATION",hDlg,lpProcLaden))
       break;
     FreeProcInstance(lpProcLaden) ;
     FilePointer=fopen(FileName,"wt");
     if(!FilePointer)
     {
       MessageBox(GetFocus(),
         (LPSTR)"Datei konnte nicht geoeffnet werden",
         (LPSTR)"Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
       return (FALSE);
     }
     anzahl=SendDlgItemMessage(hDlg,ID8_POSITION,LB_GETCOUNT,0,0);
     for(index=0;index<anzahl;index++)
     {
       SendDlgItemMessage(hDlg,ID8_POSITION,LB_GETTEXT,index,
         (LPSTR)dummy);
       wsprintf((LPSTR)dummy2,(LPSTR)"%s\n",(LPSTR)dummy);
       fputs(dummy2,FilePointer);
       wsprintf((LPSTR)Befehl,(LPSTR)"PR %s",(LPSTR)dummy2);
       lenwhere=0;
       FlushComm(nCid,1);
```

```
            Sende(nCid,Befehl);
            where[0]='\0';
            do
            {
              ndummy=ReadComm(nCid,(LPSTR)dummy,50);
              dummy[ndummy]='\0';
              lstrcat((LPSTR)where,(LPSTR)dummy);
              lenwhere+=ndummy;
              nFehler = GetCommError(nCid,&CommStatus);
              if (nFehler>0)
                MessageBox(GetFocus(),(LPSTR)"Comm",(LPSTR)"FEHLER",
                  MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
            }while(dummy[ndummy-1]!='\n');
            where[lenwhere-2] = '\n';
            where[lenwhere-2] = '\0';
            lstrcat((LPSTR)where,(LPSTR)"\n");
            fputs(where,FilePointer);
          }
          fclose(FilePointer);
          break;
      case ID8_LOAD:
          SendDlgItemMessage(hDlg,ID8_POSITION,LB_RESETCONTENT,0,0L);
          SendeComm1(nCid,"NW\n");
          strcpy(DefExt,".POS");
          szFileCaption = (LPSTR)"Positionen laden";
          lstrcpy((LPSTR)DefExt,(LPSTR)".POS");
          lpProcLaden = MakeProcInstance(DateiLaden, hInst);
          if(!DialogBox(hInst,"FILEOPERATION",hDlg,lpProcLaden))
            break;
          FreeProcInstance(lpProcLaden) ;
          FilePointer=fopen(FileName,"rt");
          if(!FilePointer)
          {
            MessageBox(GetFocus(),
            (LPSTR)"Datei konnte nicht geoeffnet werden",
            (LPSTR)"Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
            return (FALSE);
          }
          ndummy=0;
          do
          {
            if(fgets(dummy,90,FilePointer)==NULL)
              break;
            dummy[strlen(dummy)-1]='\0';
            ndummy=SendDlgItemMessage(hDlg,ID8_POSITION,
              LB_ADDSTRING,0,(LPSTR)dummy);
            if((ndummy==LB_ERR)||(ndummy==LB_ERRSPACE))
              MessageBox(GetFocus(),
                 (LPSTR)"Fehler beim Laden einer Liste",
                 (LPSTR)"Daten lesen",
                MB_OK||MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
            if(fgets(dummy2,90,FilePointer)==NULL)
                break;
            dummy2[strlen(dummy2)-1]='\0';
            wsprintf((LPSTR)Befehl1,(LPSTR)"%s,%s",(LPSTR)dummy,
              (LPSTR)dummy2);
            wsprintf((LPSTR)Befehl,(LPSTR)"PD %s",(LPSTR)Befehl1);
            lstrcat((LPSTR)Befehl,(LPSTR)"\n");
            SendeComm1(nCid,Befehl);
          }while(TRUE);
          fclose(FilePointer);
          break;
    }
```

```
        break;
  }
  return(FALSE);
}//Positionsana
```

7.4.8. N_ROBINI.C

```
/*****************************************************************************
 Modul:       N_ROBINI.C

 Aufgabe:     Initialisiert den Roboter und enthält Kommunikationsfunktionen

 Funktionen:  RobInit
              SendeComm1
              Sende
              HoleRoboterPosition
              LadeGrundposition
              Gelenkdemo
*****************************************************************************/

#include "windows.h"
#include "neurorob.h"
#include "n_extern.h"
#include "ndl_rob.h"
#include "stdio.h"
#include "math.h"

/*****************************************************************************
 Funktion:      RobInit

 Aufgabe:       Initialisiert den Roboter

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*****************************************************************************/


BOOL FAR PASCAL RobInit(HWND      hDlg,
                        unsigned  message,
                        WORD      wParam,
                        LONG      lParam)
{
char              szSetting[] = "COM1: 9600,E,7,2";
COMSTAT           CommStatus;
static  BOOL      bNTSenden;

switch (message){
    case WM_INITDIALOG:
         bNTSenden=FALSE;
         return (TRUE);
    case WM_COMMAND:
         switch(wParam){
           case IDCANCEL:
                bMoveMasterInit=FALSE;
                EndDialog(hDlg, TRUE);
                return(TRUE);
           case ID19_NT:
                if(bNTSenden)
                {
                  CheckDlgButton(hDlg,wParam,0);
                  bNTSenden=FALSE;
```

```
      }
      else
      {
        CheckDlgButton(hDlg,wParam,1);
        bNTSenden=TRUE;
      }
      break;
case IDOK:
      SetDlgItemText(hDlg,ID19_TEXT1,
        (LPSTR)"Bitte warten Sie einen Moment!");
      if (BuildCommDCB(szSetting,&CommDCB)!=0)
      {
        MessageBox(GetFocus(),
          (LPSTR)"Schnittstellenfehler aufgetreten",
          NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        EndDialog(hDlg,TRUE);
      }
      CommDCB.fOutxDsrFlow=1;
      CommDCB.fOutxCtsFlow=1;
      CloseComm(nCid);
      if ((nCid = OpenComm((LPSTR)"COM1",128,128))<0)
      {
        MessageBox(GetFocus(),
          (LPSTR)"Schnittstellenfehler aufgetreten",
          NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        EndDialog(hDlg,TRUE);
        return(TRUE);
      }
      if (SetCommState(&CommDCB) != 0)
      {
        MessageBox(GetFocus(),
          (LPSTR)"Schnittstellenfehler aufgetreten",
          NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        EndDialog(hDlg,TRUE);
        return(TRUE);
      }
      FlushComm(nCid,0);
      FlushComm(nCid,1);
      if(bNTSenden)
        SendeComm1(nCid,"NT\n");
      wsprintf((LPSTR)dummy,(LPSTR)"%i",roboterspeed);
      SP((LPSTR)dummy);
      SendeComm1(nCid,"TL 132\n");
      SendeComm1(nCid,"NW\n");
      if(!LadeGrundposition())
      {
        MessageBox(GetFocus(),
          (LPSTR)"Schnittstellenfehler aufgetreten",
          NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        EndDialog(hDlg,TRUE);
        return(TRUE);
      }
      SendeComm1(nCid,"MO 999,O\n");
      if(GetCommError(nCid,&CommStatus)>0)
      {
        MessageBox(GetFocus(),
          (LPSTR)"Schnittstellenfehler aufgetreten",
          NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
        EndDialog(hDlg,TRUE);
        return(TRUE);
      }
      bMoveMasterInit=TRUE;
      EndDialog(hDlg, TRUE);
```

```
                return(TRUE);
            }
  }
  return (FALSE);
}//RobInit

/*****************************************************************************
 Funktion:      SendeComm1

 Aufgabe:       Sendet ein Befehl zum Roboter mit nachfolgender Fehlerprüfung

 Parameter:     nCid            : Handle der Schnittstelle
                *befehlsstring  : Zeiger auf Zeichenkette die den Befehl
                                  enthält

 Rückgabewert:  int TRUE wenn erfolgreich

*****************************************************************************/


int SendeComm1(HANDLE nCid,char *befehlsstring)
{
char    error[5];
int     lenerror;
COMSTAT CommStatus;
HCURSOR hSaveCursor;
char    Sender[50];
int     nSender;

  hSaveCursor = SetCursor(hMaster);
  Sende(nCid,befehlsstring);
  lenerror = 0;
  FlushComm(nCid,1);
  Sende(nCid,"ER\n");
  error[0]='\0';
  do
  {
    nSender = ReadComm(nCid,(LPSTR)Sender,5);
    Sender[nSender]='\0';
    lstrcat((LPSTR)error,(LPSTR)Sender);
    lenerror+=nSender;
  }while(Sender[nSender-1]!='\n');
  error[lenerror-1]='\0';
  if(error[0]=='1')
  {
    Sende(nCid,"RS\n");
    MessageBox(GetFocus(),(LPSTR)"Starten Sie das System komplett neu",
      (LPSTR)"Hardware Error",MB_OK|MB_ICONHAND|MB_SYSTEMMODAL);
  }
  if(error[0]=='2')
  {
    Sende(nCid,"RS\n");
    MessageBox(GetFocus(),(LPSTR)befehlsstring,
      (LPSTR)"Software Error",MB_OK|MB_ICONHAND|MB_SYSTEMMODAL);
  }
  SetCursor(hSaveCursor);
  return(TRUE);
}//SendComm1

/*****************************************************************************
 Funktion:       Sende

 Aufgabe:        Übernimmt Datenübertragung zum Movemaster
```

```
 Parameter: nCid            : Handle von COM1
            *befehlsstring  : Zeiger auf String der den Befehl enthält

 Rückgabewert: int   negativ wenn Fehler aufgetreten
*****************************************************************************/


int Sende(HANDLE  nCid,char *befehlsstring)
{
COMSTAT CommStatus;
LONG    i;
LONG    time1;
LONG    time2;
LONG    x;
  if(GetCommError(nCid,&CommStatus)!=0)
    return(FALSE);
  do
  {
    GetCommError(nCid,&CommStatus);
  }while(CommStatus.cbOutQue!=0);
  x = 1;
  time1 = GetTickCount();
  do
  {
    time2 = GetTickCount();
  }while((time2-time1)<x);
  if(WriteComm(nCid,befehlsstring,min(lstrlen(befehlsstring),
      128-CommStatus.cbOutQue))<0)
  {
    MessageBox(GetFocus(),(LPSTR)"Fehler beim Schreiben auf COM1",
      (LPSTR)"WriteComm",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
    return(FALSE);
  }
  return(TRUE);
}//Sende


/*****************************************************************************
 Funktion:      HoleRoboterPosition

 Aufgabe:       Holt die aktuellen Roboterkoordinaten und gibt diese in where
                und grid zurück

 Parameter:     where : Zeiger auf String der genug Platz zur Verfügung
                        stellt
                grid  : dito

 Rückgabewert:  -
*****************************************************************************/

void HoleRoboterPosition(LPSTR where,LPSTR grid)
{
int   lenwhere=0;
int   gelesen=0;
char  string[50];

  Sende(nCid,"WH\n");
  where[0]='\0';
  do
  {
    gelesen = ReadComm(nCid,(LPSTR)string,50);
    string[gelesen]='\0';
```

```
    lstrcat((LPSTR)where,(LPSTR)string);
    lenwhere+=gelesen;
  }while(string[gelesen-1]!='\n');
  where[lenwhere-4]='\0';
  where[lenwhere-2]='\0';
  grid[0]='G';
  lstrcpy((LPSTR)&grid[1],(LPSTR)&where[lenwhere-3]);
}//HoleRoboterPosition

/*****************************************************************************
 Funktion:      LadeGrundposition

 Aufgabe:       Läd die Systemposition die in NEUROROB.POS stehen und sendet
                sie zum Movemaster

 Parameter:     -

 Rückgabewert:  BOOL          TRUE wenn erfolgreich
*****************************************************************************/

BOOL LadeGrundposition()
{
FILE *FilePointer;
char Befehl[50];
char Befehl1[50];
char FileName[100]="";

  FilePointer=fopen("NEUROROB.POS","rt");
  if(!FilePointer)
  {
    MessageBox(GetFocus(),"Datei konnte nicht geoeffnet werden",
      "Fehler",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
    return (FALSE);
  }
  ndummy=0;
  do
  {
    if(fgets(dummy,90,FilePointer)==NULL)
      break;
    dummy[strlen(dummy)-1]='\0';
    if(fgets(dummy2,90,FilePointer)==NULL)
      break;
    dummy2[strlen(dummy2)-1]='\0';
    wsprintf((LPSTR)Befehl1,(LPSTR)"%s,%s",(LPSTR)dummy,(LPSTR)dummy2);
    wsprintf((LPSTR)Befehl,(LPSTR)"PD %s",(LPSTR)Befehl1);
    lstrcat((LPSTR)Befehl,(LPSTR)"\n");
    SendeComm1(nCid,Befehl);
  }while(TRUE);
  fclose(FilePointer);
  return TRUE;
}//LadeGrundPosition

/*****************************************************************************
 Funktion:      Gelenkdemo

 Aufgabe:       Führt eine Gelenkarmdemo für den Roboter durch

 Parameter:

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*****************************************************************************/
```

```
void Gelenkdemo(void)
{
float rad,droll,x1,y1;
int  dx,dy,a,i,k,r,z,x,y,we;
char Demo[50];

  SP ("10");
  PD ("600,0,400,200,-90,90,C");
  PD ("601,399.5,-654.7,393.4,-0.6,90,C");
  PD ("602,456.4,315.8,458.9,35.6,90,C");
  MO ("600,C");
  SP ("20");
  DS ("0,0,-100");
  GO ;
  GC ;
  MJ ("0,0,0,90,0");
  DS ("0,0,-100");
  HE ("699");
  MT ("699,-50,C");
  HE ("699");
  MT ("699,100,C");
  MO ("602,C");
  MT ("602,200,C");
  HE ("699");
  MT ("699,-200,C");
  MJ ("-100,0,0,0,0");
  HE ("699");
  MT ("699,200,C");
  HE ("699");
  MT ("699,-200,C");
  MO ("601,C");
  MJ ("-280,0,0,0,0");
  MJ ("0,-25,0,0,0");
  MJ ("280,60,0,0,0");
  MJ ("0,-35,0,0,0");
  MJ ("0,100,0,100,0");
  MJ ("-280,0,0,0,0");
  MO ("601,C");
  MJ ("-150,90,0,-90,0");
  MJ ("0,0,-90,180,90");
  MJ ("0,0,90,-180,-90");
  MJ ("0,-90,-90,0,0");
  MJ ("150,0,90,90,0");
  MO ("600,C");

  MessageBox(GetFocus(),(LPSTR)"Kreis",(LPSTR)"DEMO",MB_OK|MB_SYSTEMMODAL);
  PD ("10,-7,450,35,-90,90,C");
  MO ("10,C");
  a=10;
  dx=-7;
  dy=340;
  r=300-dy;
  for(i=1;i<=30;i++)
  {
    a++;
    rad=3*i*3.14159/45;
    y1=(cos(rad)*r+dy);
    x1=(sin(rad)*r+dx);
    droll=180*atan(x/y)/3.14159;
    we=(int)(90+droll);
    y=(int)y1;
    x=(int)x1;
```

```
    wsprintf(Demo,"%d, %d, %d,35,-90,90,C",a,x,y);
    PD (Demo);
  }
  MC ("10,40,C");
}//GelenkDemo
```

7.4.9. N_STEU.C

```
/*****************************************************************************
 Modul:        N_STEU.C

 Aufgabe:      Übernimmt die Verwaltung der Steuerungdialogbox

 Funktionen:   Steuerung
*****************************************************************************/


#include "windows.h"
#include "neurorob.h"
#include "n_extern.h"
#include "ndl_steu.h"
#include "ndl_visu.h"
#include "neuromen.h"
#include "ndl_list.h"

/*****************************************************************************
 Funktion:      Steuerung

 Aufgabe:       Darstellen und Bearbeiten der Steuerungdialogbox

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*****************************************************************************/

BOOL FAR PASCAL Steuerung(HWND      hDlg,
                          unsigned  message,
                          WORD      wParam,
                          LONG      lParam)

{
char        texti[11];
HDC         hDC;
PAINTSTRUCT ps;
HDC         hMemoryDC;
int         balkenmerker;
BOOL        speed;
int         scrollmax[4]={20,100,63,45};
int         page[4]={4,10,5,5};
char        befehl[20];

  switch (message){
    case WM_INITDIALOG:
         SetScrollRange(GetDlgItem(hDlg,ID2_SPEED),SB_CTL,0,scrollmax[0],0);
         SetScrollPos(GetDlgItem(hDlg,ID2_SPEED),SB_CTL,roboterspeed,1);
         SetScrollRange(GetDlgItem(hDlg,ID2_STEP),SB_CTL,0,scrollmax[1],0);
         SetScrollPos(GetDlgItem(hDlg,ID2_STEP),SB_CTL,roboterstep,1);
         SetScrollRange(GetDlgItem(hDlg,ID2_GRID),SB_CTL,0,scrollmax[2],0);
         SetScrollPos(GetDlgItem(hDlg,ID2_GRID),SB_CTL,robotergrid,1);
         SetScrollRange(GetDlgItem(hDlg,ID2_DEGREE),SB_CTL,0,scrollmax[3],0);
         SetScrollPos(GetDlgItem(hDlg,ID2_DEGREE),SB_CTL,roboterdegree,1);
         wsprintf((LPSTR)dummy,(LPSTR)"Geschwindigkeit : %i",roboterspeed);
         SetWindowText(GetDlgItem(hDlg,ID2_TEXTSPEED),(LPSTR)dummy);
```

```
        wsprintf(befehl,"SP %i\n",roboterspeed);
        SendeComm1(nCid,befehl);
        wsprintf((LPSTR)dummy,(LPSTR)"Schrittweite : %i mm",roboterstep);
        SetWindowText(GetDlgItem(hDlg,ID2_TEXTMM),(LPSTR)dummy);
        wsprintf((LPSTR)dummy,(LPSTR)"Greifstõrke : %i",robotergrid);
        SetWindowText(GetDlgItem(hDlg,ID2_TEXTGRID),(LPSTR)dummy);
        wsprintf(befehl,"GP %i,%i,3\n",robotergrid,robotergrid);
        SendeComm1(nCid,befehl);
        wsprintf((LPSTR)dummy,(LPSTR)"Schrittwinkel : %i _",roboterdegree);
        SetWindowText(GetDlgItem(hDlg,ID2_TEXTGRAD),(LPSTR)dummy);
        return (TRUE);
    case WM_COMMAND:
        switch(wParam){
          case IDOK:
          case IDCANCEL:
               SendMessage(hDlg,WM_SYSCOMMAND,SC_CLOSE,0L);
               return (TRUE);
          case ID2_HAND_OPEN:
               SendeComm1(nCid,(LPSTR)"GO\n");
               break;
          case ID2_HAND_CLOSE:
               SendeComm1(nCid,(LPSTR)"GC\n");
               break;
          case ID2_SHOULDER_P:
               wsprintf(befehl,"MJ 0,%i,0,0,0\n",roboterdegree);
               SendeComm1(nCid,befehl);
               break;
          case ID2_SHOULDER_N:
               wsprintf(befehl,"MJ 0,%i,0,0,0\n",-roboterdegree);
               SendeComm1(nCid,befehl);
               break;
          case ID2_WAIST_P:
               wsprintf(befehl,"MJ %i,0,0,0,0\n",roboterdegree);
               SendeComm1(nCid,befehl);
               break;
          case ID2_WAIST_N:
               wsprintf(befehl,"MJ %i,0,0,0,0\n",-roboterdegree);
               SendeComm1(nCid,befehl);
               break;
          case ID2_ELBOW_P:
               wsprintf(befehl,"MJ 0,0,%i,0,0\n",roboterdegree);
               SendeComm1(nCid,befehl);
               break;
          case ID2_ELBOW_N:
               wsprintf(befehl,"MJ 0,0,%i,0,0\n",-roboterdegree);
               SendeComm1(nCid,befehl);
               break;
          case ID2_HAND_UP:
               wsprintf(befehl,"MJ 0,0,0,%i,0\n",roboterdegree);
               SendeComm1(nCid,befehl);
               break;
          case ID2_HAND_DOWN:
               wsprintf(befehl,"MJ 0,0,0,%i,0\n",-roboterdegree);
               SendeComm1(nCid,befehl);
               break;
          case ID2_HAND_RIGHT:
               wsprintf(befehl,"MJ 0,0,0,0,%i\n",-roboterdegree);
               SendeComm1(nCid,befehl);
               break;
          case ID2_HAND_LEFT:
               wsprintf(befehl,"MJ 0,0,0,0,%i\n",roboterdegree);
               SendeComm1(nCid,befehl);
               break;
```

```
        case ID2_MO_999:
             SendeComm1(nCid,"MO 999\n");
             break;
        case ID2_LINKS:
             wsprintf(befehl,"DS %i,0,0\n",+roboterstep);
             SendeComm1(nCid,befehl);
             break;
        case ID2_RECHTS:
             wsprintf(befehl,"DS %i,0,0\n",-roboterstep);
             SendeComm1(nCid,befehl);
             break;
        case ID2_ZURUECK:
             wsprintf(befehl,"DS 0,%i,0\n",-roboterstep);
             SendeComm1(nCid,befehl);
             break;
        case ID2_VOR:
             wsprintf(befehl,"DS 0,%i,0\n",roboterstep);
             SendeComm1(nCid,befehl);
             break;
        case ID2_RUNTER:
             wsprintf(befehl,"DS 0,0,%i\n",-roboterstep);
             SendeComm1(nCid,befehl);
             break;
        case ID2_HOCH:
             wsprintf(befehl,"DS 0,0,%i\n",roboterstep);
             SendeComm1(nCid,befehl);
             break;
        case ID2_ADD:
             if(bBefehlsliste==FALSE)
               SendMessage(GetParent(hDlg),WM_COMMAND,IDM_BEFEHLSLISTE,0L);
             SendMessage(hDlgBefehlsliste,WM_COMMAND,ID3_ADD,0L);
             SetFocus(hDlg);
             break;
        case ID2_DEFINIERN:
             GetDlgItemText(hDlg,ID2_POSEDIT,(LPSTR)dummy,5);
             ndummy=atoi(dummy);
             if(ndummy<2 || ndummy>990)
             {
               MessageBox(GetFocus(),(LPSTR)"Unzulaessige Positionsnummer",
                 NULL,MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
               break;
             }
             HE(dummy);
             if (ndummy<998)
             {
               wsprintf((LPSTR)dummy,(LPSTR)"%i",ndummy+1);
               SetDlgItemText(hDlg,ID2_POSEDIT,(LPSTR)dummy);
             }
             break;
      }
      if(bVisualisierung)
        SendMessage(hDlgVisualisierung,WM_COMMAND,ID10_AKTUELL,0L);
      break;
case WM_SYSCOMMAND:
      switch(wParam){
        case SC_CLOSE:
             DestroyWindow(hDlgSteuerung);
             FreeProcInstance(lpSteuerung);
             bSteuerung = FALSE;
             hDlgSteuerung = NULL;
             CheckMenuItem(hMainMenu,IDM_STEUERUNG,MF_UNCHECKED);
             return TRUE;
        default:
```

```
            return FALSE;
      }
      break;
case WM_PAINT:
      hDC = BeginPaint(hDlg,&ps);
      hMemoryDC  = CreateCompatibleDC(hDC);
      hOldBitmap = SelectObject(hMemoryDC,hRobBitmap);
      if(hOldBitmap)
      {
         BitBlt (hDC,60,40,600,550,hMemoryDC,0,0,SRCCOPY);
         SelectObject(hMemoryDC,hOldBitmap);
      }
      DeleteDC(hMemoryDC);
      EndPaint(hDlg,&ps);
      break;
case WM_HSCROLL:
      hDC=GetDC(hDlg);
      balkenmerker = GetScrollPos(HIWORD(lParam),SB_CTL);
      if (HIWORD(lParam) == GetDlgItem(hDlg,ID2_SPEED))
         speed=0;
      if (HIWORD(lParam) == GetDlgItem(hDlg,ID2_STEP))
         speed=1;
      if (HIWORD(lParam) == GetDlgItem(hDlg,ID2_GRID))
         speed=2;
      if (HIWORD(lParam) == GetDlgItem(hDlg,ID2_DEGREE))
         speed=3;
      switch(wParam){
        case SB_LINEDOWN:
             if(balkenmerker<scrollmax[speed])
             ++balkenmerker;
             break;
        case SB_LINEUP:
             if(balkenmerker>0)
               --balkenmerker;
             break;
        case SB_PAGEDOWN:
             if(balkenmerker<scrollmax[speed]-page[speed])
               balkenmerker+=page[speed];
             else
               balkenmerker=scrollmax[speed];
             break;
        case SB_PAGEUP:
             if(balkenmerker>page[speed])
               balkenmerker-=page[speed];
             else
               balkenmerker=0;
             break;
        case SB_BOTTOM:
             balkenmerker=0;
             break;
        case SB_TOP:
             balkenmerker=scrollmax[speed];
             break;
        case SB_THUMBTRACK:
             balkenmerker=LOWORD(lParam);
             break;
      }
      SetScrollPos(HIWORD(lParam),SB_CTL,balkenmerker,1);
      if (speed == 0)
      {
         roboterspeed = balkenmerker;
         wsprintf((LPSTR)dummy,(LPSTR)"Geschwindigkeit : %i",roboterspeed);
         SetWindowText(GetDlgItem(hDlg,ID2_TEXTSPEED),(LPSTR)dummy);
```

```
            wsprintf(befehl,"SP %i\n",roboterspeed);
            SendeComm1(nCid,befehl);
         }
         if (speed == 1)
         {
            roboterstep = balkenmerker;
            wsprintf((LPSTR)dummy,(LPSTR)"Schrittweite : %i mm",roboterstep);
            SetWindowText(GetDlgItem(hDlg,ID2_TEXTMM),(LPSTR)dummy);
         }
         if (speed == 2)
         {
            robotergrid = balkenmerker;
            wsprintf((LPSTR)dummy,(LPSTR)"Greifstõrke : %i",robotergrid);
            SetWindowText(GetDlgItem(hDlg,ID2_TEXTGRID),(LPSTR)dummy);
            wsprintf(befehl,"GP %i,%i,3\n",robotergrid,robotergrid);
            SendeComm1(nCid,befehl);
         }
         if (speed == 3)
         {
            roboterdegree = balkenmerker;
            wsprintf((LPSTR)dummy,(LPSTR)"Schrittwinkel : %i _",
              roboterdegree);
            SetWindowText(GetDlgItem(hDlg,ID2_TEXTGRAD),(LPSTR)dummy);
         }
         ReleaseDC(hDlg,hDC);
         break;
   }
   return(FALSE);
}//Steuerung
```

7.4.10. N_VIDEOI.C

```
/*****************************************************************************
 Modul:       N_VIDEO.C

 Aufgabe:     Enthält Funktion, die die Kommunikation mit der Sivips-Anlage
              übernehmen

 Funktionen:  VideoInit
              HolePartSchwerpunkt
              VideoSteuerung
              SendeComm2
              HoleVideomatZeile
*****************************************************************************/

#include "windows.h"
#include "neurorob.h"
#include "n_extern.h"
#include "neuromen.h"
#include "ndl_vid.h"
#include "math.h"

/*****************************************************************************
 Funktion:      VideoInit

 Aufgabe:       Initialisiert die serielle Schnittstelle

 Parameter:

 Rückgabewert:  BOOL  TRUE wenn erfolgreich
*****************************************************************************/


BOOL VideoInit (void)
{
char    szSetting[] = "COM2: 9600,E,7,2";
int     nFehler;
COMSTAT Commstatus;
DCB     CommDCB;

  if(nCid!=0)
    CloseComm(nCid2);
  if (BuildCommDCB(szSetting,&CommDCB)!=0)
    return(FALSE);
  CommDCB.fOutxDsrFlow=1;
  if((nCid2 = OpenComm((LPSTR)"COM2",128,128))<0)
    return(FALSE);
  if (SetCommState(&CommDCB)!=0)
    return(FALSE);
  FlushComm(nCid2,0);
  FlushComm(nCid2,1);
  return(TRUE);
}//VideoInit

/*****************************************************************************
 Funktion:      HolePartSchwerpunkt

 Aufgabe:       Holt vom Videomaten den Schwerpunkt eines Parts

 Parameter:     -

 Rückgabewert:  -
```

```
*************************************************************************/

void HolePartSchwerpunkt(void)
{
char Koordinaten[50];

  Koordinaten[0]='\0';
  SendeComm2(nCid2,"X");
  HoleVideomatZeile((LPSTR)Koordinaten);
  ndummy=0;
  while(Koordinaten[ndummy]==' ')
    ++ndummy;
  strcpy(dummy,&(Koordinaten[ndummy]));
  yPart=atof(dummy);
  Koordinaten[0]='\0';
  SendeComm2(nCid2,"A");
  HoleVideomatZeile((LPSTR)Koordinaten);
  ndummy=0;
  while(Koordinaten[ndummy]==' ')
    ++ndummy;
  strcpy(dummy,&(Koordinaten[ndummy]));
  xPart=atof(dummy);
}//HolePartSchwerpunkt

/*************************************************************************
 Funktion:      Videosteuerung

 Aufgabe:       Darstellen und bearbeiten der VideoSteuerung-Dialogbox

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*************************************************************************/


BOOL FAR PASCAL VideoSteuerung( HWND      hDlg,
                                unsigned  message,
                                WORD      wParam,
                                LONG      lParam)
{
  switch (message){
    case WM_INITDIALOG:
         return (TRUE);
    case WM_COMMAND:
         switch(wParam){
           case IDOK:
           case IDCANCEL:
                SendMessage(hDlg,WM_SYSCOMMAND,SC_CLOSE,0L);
                return (TRUE);
           case ID21_LIVEG:
                SendeComm2(nCid2,"B");
                break;
           case ID21_LIVEB:
                SendeComm2(nCid2,"D");
                break;
           case ID21_GRAU:
                SendeComm2(nCid2,"C");
                break;
```

```
            case ID21_BINAER:
                 SendeComm2(nCid2,"E");
                 break;
            case ID21_PLUS:
                 SendeComm2(nCid2,"P");
                 break;
            case ID21_MINUS:
                 SendeComm2(nCid2,"M");
                 break;
            case ID21_HISTOGRAMM:
                 SendeComm2(nCid2,"H");
                 break;
            case ID21_ORIENTIERUNG:
                 SendeComm2(nCid2,"O");
                 break;
          }
          break;
    case WM_SYSCOMMAND:
          switch(wParam){
            case SC_CLOSE:
                 DestroyWindow(hDlgVideoSteuerung);
                 FreeProcInstance(lpVideoSteuerung);
                 bVideoSteuerung = FALSE;
                 hDlgVideoSteuerung = NULL;
                 CheckMenuItem(hMainMenu,IDM_VIDEOSTEU,MF_UNCHECKED);
                 return TRUE;
            default:
                 return FALSE;
          }
          break;
  }
  return(FALSE);
}//VideoSteuerung

/*****************************************************************************
 Funktion:      SendeComm2

 Aufgabe:       Sendet Zeichenketten zur Sivips-Anlage

 Parameter:     nCid          : Handle der Schnittstelle
                *befehlsstring: Zeiger auf die zu sendende Zeichenkette

 Rückgabewert:  int   negativ falls Fehler aufgetreten
*****************************************************************************/


int SendeComm2(HANDLE nCid2,char *befehlsstring)
{
COMSTAT CommStatus;
int     nFehler;

  if((nFehler=GetCommError(nCid2,&CommStatus))!=0)
  {
    itoa(nFehler,dummy,10);
    MessageBox(GetFocus(),(LPSTR)dummy,(LPSTR)"GetCommError",
      MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
    return(-20);
  }
  do
  {
    GetCommError(nCid2,&CommStatus);
  }while(CommStatus.cbOutQue!=0);
```

```
  if(WriteComm(nCid2,befehlsstring,min(lstrlen(befehlsstring),
      128-CommStatus.cbOutQue))<0)
  {
    MessageBox(GetFocus(),(LPSTR)"Fehler beim Schreiben auf COM2",
      (LPSTR)"WriteComm",MB_OK|MB_ICONEXCLAMATION|MB_SYSTEMMODAL);
    return(-20);
  }
  return(0);
}//SendeComm2

/***************************************************************************
 Funktion:        HoleViedomatZeile

 Aufgabe:         Empfängt eine Zeichenkette vom Videomatsystem

 Parameter:       video :Zeiger auf Zeichenkette die genügend Speicherplatz
                  zur Verfügung stellt

 Rückgabewert:    -
***************************************************************************/


void HoleVideomatZeile(LPSTR video)
{
  video[0]='\0';
  do
  {
    ndummy = ReadComm(nCid2,(LPSTR)dummy,50);
    dummy[ndummy]='\0';
    lstrcat((LPSTR)video,(LPSTR)dummy);
  }while((ndummy == 0) || (dummy[ndummy-1]!='\n'));
  dummy[0] = '\0';
}//HoleVideomatZeile
```

7.4.11. N_VISUAL.C

```
/*****************************************************************************
 Modul:       N_VISUAL.C

 Aufgabe:     Enthält Funktionen zur Visualisierung

 Funktionen:  Visualisierung
*****************************************************************************/


#include "windows.h"
#include "neurorob.h"
#include "n_extern.h"
#include "ndl_visu.h"
#include "neuromen.h"
#include "math.h"

/*****************************************************************************
 Funktion:      Visualisierung

 Aufgabe:       Visualisierungsdialogbox darstellen

 Parameter:
  HWND      hDlg   : Fensterhandle
  unsigned  message: Typ der Nachricht
  WORD      wParam : Nachrichtenabhängiger Wert
  LONG      lParam :        "               "

 Rückgabewert:
  BOOL          TRUE wenn message abgearbeitet
*****************************************************************************/

BOOL FAR PASCAL Visualisierung( HWND      hDlg,
                                unsigned  message,
                                WORD      wParam,
                                LONG      lParam)
{
HDC           hDC;
PAINTSTRUCT   ps;
int           i,c;

static  int   x=0;
static  int   y=0;
static  int   z=0;
static  int   nOffX1;
static  int   nOffX2;
static  int   nOffY1;
static  int   nOffY2;
static  int   nOffZ1;
static  int   nOffZ2;

static  char  Daten[50];
static  char  Grip[60];
static  char  Koordinatenx[10]="0";
static  char  Koordinateny[10]="0";
static  char  Koordinatenz[10]="0";


  switch (message){
    case WM_INITDIALOG:
         Koordinatenx[0]='\0';
```

```
      Koordinateny[0]='\0';
      Koordinatenz[0]='\0';
      return (TRUE);
  case WM_COMMAND:
      switch(wParam){
        case IDOK:
        case IDCANCEL:
             PostMessage(hDlg,WM_SYSCOMMAND,SC_CLOSE,0L);
             return (TRUE);
        case ID10_AKTUELL:
             Koordinatenx[0]='\0';
             Koordinateny[0]='\0';
             Koordinatenz[0]='\0';
             HoleRoboterPosition((LPSTR)Daten,(LPSTR)Grip);
             i=0;
             c=0;
             while(Daten[i]!=',')
             {
               Koordinatenx[c] = Daten[i];
               i++;
               c++;
             }
             Koordinatenx[c] = '\0';
             c=0;
             i++;
             while(Daten[i]!=',')
             {
               Koordinateny[c] = Daten[i];
               i++;
               c++;
             }
             Koordinateny[c] = '\0';
             c=0;
             i++;
             while(Daten[i]!=',')
             {
               Koordinatenz[c] = Daten[i];
               i++;
               c++;
             }
             Koordinatenz[c] = '\0';
             wsprintf((LPSTR)dummy,(LPSTR)"%s,%s,%s",(LPSTR)Koordinatenx,
               (LPSTR)Koordinateny,(LPSTR)Koordinatenz);
             SetDlgItemText(hDC,ID10_DATEN,(LPSTR)dummy);
             InvalidateRect(hDlg,NULL,TRUE);
             SendMessage(hDlg,WM_PAINT,0,0L);
             break;
      }
      break;
  case WM_PAINT:
             hDC = BeginPaint(hDlg, &ps);
             SetMapMode(hDC,MM_ANISOTROPIC);
             SetWindowExt(hDC,1,1);
             SetViewportExt(hDC,1,1);
             SetWindowOrg(hDC,1,1);
             SetViewportOrg(hDC,0,0);
             SelectObject(hDC, hRotPen1);
             MoveTo(hDC,5,120);
             LineTo(hDC,130,120);
             MoveTo(hDC,70,60);
             LineTo(hDC,70,180);

             MoveTo(hDC,145,120);
```

```
LineTo(hDC,270,120);
MoveTo(hDC,210,60);                 // z-Achse
LineTo(hDC,210,180);
                                    // K-System y-z
MoveTo(hDC,285,120);                // y-Achse
LineTo(hDC,405,120);
MoveTo(hDC,350,60);                 // z-Achse
LineTo(hDC,350,180);

MoveTo(hDC,125,115);
LineTo(hDC,130,120);                // Pfeil x
MoveTo(hDC,125,125);
LineTo(hDC,130,120);

MoveTo(hDC,65,65);                  // Pfeil y
LineTo(hDC,70,60);
MoveTo(hDC,75,65);
LineTo(hDC,70,60);

MoveTo(hDC,265,115);
LineTo(hDC,270,120);                // Pfeil x
MoveTo(hDC,265,125);
LineTo(hDC,270,120);

MoveTo(hDC,205,65);
LineTo(hDC,210,60);                 // Pfeil z
MoveTo(hDC,215,65);
LineTo(hDC,210,60);

MoveTo(hDC,400,115);
LineTo(hDC,405,120);
MoveTo(hDC,400,125);                // Pfeil y
LineTo(hDC,405,120);

MoveTo(hDC,345,65);
LineTo(hDC,350,60);                 // Pfeil z
MoveTo(hDC,355,65);
LineTo(hDC,350,60);
                                    // Koord._Text
SelectObject(hDC,hSchwarzPen1);
TextOut  (hDC,125,125,(LPSTR)"x",1);
TextOut  (hDC,75,65,(LPSTR)"y",1);
TextOut  (hDC,266,126,(LPSTR)"x",1);
TextOut  (hDC,215,65,(LPSTR)"z",1);
TextOut  (hDC,401,126,(LPSTR)"y",1);
TextOut  (hDC,355,65,(LPSTR)"z",1);
x=atoi(Koordinatenx);
y=atoi(Koordinateny);
z=atoi(Koordinatenz);
nOffX1=70;                          // OFFSET fuer ....
nOffY1=120;                         // Koordinatensystem
nOffX2=210;
nOffZ1=120;
nOffY2=350;
nOffZ2=120;
x=x/15;
y=y/15;
z=z/15;
y=y*(-1);
z=z*(-1);
SelectObject(hDC,hSchwarzPen1);
SelectObject(hDC,hSchwarzBrush1);
Ellipse(hDC,x+nOffX1-2,y+nOffY1-2,x+nOffX1+2,y+nOffY1+2);
```

```
                Ellipse(hDC,x+nOffX2-2,z+nOffZ1-2,x+nOffX2+2,z+nOffZ1+2);
                y=y*(-1);
                Ellipse(hDC,y+nOffY2-2,z+nOffZ2-2,y+nOffY2+2,z+nOffZ2+2);
                EndPaint(hDlg,&ps);
          break;
     case WM_SYSCOMMAND:
          switch(wParam){
            case SC_CLOSE:
                DestroyWindow(hDlgVisualisierung);
                FreeProcInstance(lpVisualisierung);
                bVisualisierung = FALSE;
                hDlgVisualisierung = NULL;
                CheckMenuItem(hMainMenu,IDM_VISUALISIERUNG,MF_UNCHECKED);
                return TRUE;
            default:
                return FALSE;
          }
          break;
   }
   return(FALSE);
}//Visualisierung
```

7.5. Der Resourcequellcode NEUROROB.RC

```
/*****************************************************************************
 Modul:       NEUROROB.RC

 Aufgabe:     Enthält die alle Resourcen für Neurorob
*****************************************************************************/

#include "windows.h"
#include "neurorob.h"
#include "ndl_steu.h"
#include "ndl_list.h"
#include "neuromen.h"
#include "ndl_vid.h"
#include "ndl_edit.h"
#include "brainmen.h"
#include "ndl_nnet.h"
#include "ndl_file.h"
#include "ndl_pos.h"
#include "ndl_ndat.h"
#include "ndl_tdat.h"
#include "ndl_visu.h"
#include "ndl_abo.h"
#include "ndl_test.h"
#include "ndl_rob.h"

NEUROROBMENU  MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
  MenuItem  "&Info", IDM_INFO
  POPUP "&Roboter"
  BEGIN
    MenuItem  "&Init ...", IDM_ROBINIT
    MenuItem  SEPARATOR
    MenuItem  "&Steuerung ...", IDM_STEUERUNG
    MenuItem  "&Befehlsliste ...", IDM_BEFEHLSLISTE
    MenuItem  "Sende &File ...", IDM_SENDEFILE
    MenuItem  "&Positionsanalyse ...", IDM_POSITIONSANA
    MenuItem  "&Visualisierung ...", IDM_VISUALISIERUNG
    MenuItem  SEPARATOR
    MenuItem  "&Beenden", IDM_BEENDEN
  END
  POPUP "&Videomat"
  BEGIN
    MenuItem  "&Init / Abgleich", IDM_VIDEOINIT
    MenuItem  SEPARATOR
    MenuItem  "&Steuerung ...", IDM_VIDEOSTEU
  END
  POPUP "&Brain"
  BEGIN
    MenuItem  "&Init ...", IDM_NEUINIT
  END
  POPUP "D&emo"
  BEGIN
    MenuItem  "&Gelenkdemo", IDM_GELENKDEMO
  END
END

VIDEOSTEUERUNG DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 88, 26, 104, 182
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Videomat"
BEGIN
```

```
  CONTROL "+" ID21_PLUS, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 34, 17, 31,
13
  CONTROL "-" ID21_MINUS, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 34, 36, 31,
13
  CONTROL "Binõrisierungsschwelle" 0, "BUTTON", WS_CHILD | WS_VISIBLE | 0x7L, 8, 3,
87, 50
  CONTROL "Live-Graubild" ID21_LIVEG, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP,
7, 60, 90, 13
  CONTROL "Live-Binõrbild" ID21_LIVEB, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP,
7, 76, 90, 13
  CONTROL "Graubild" ID21_GRAU, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 7, 92,
90, 13
  CONTROL "Binõrbild" ID21_BINAER, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 7,
108, 90, 13
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 24, 161, 57, 12
  CONTROL "Histogramm" ID21_HISTOGRAMM, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 7, 124, 90, 13
  CONTROL "Orientierung/Schwerpunkt" ID21_ORIENTIERUNG, "BUTTON", WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 7, 140, 90, 13
END

ABOUTBOX DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 56, 9, 174, 208
STYLE WS_TILED | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "_ber NeuroRob"
BEGIN
  CONTROL "Labor f³r Regelungstechnik " -1, "STATIC", WS_CHILD | WS_VISIBLE |
WS_GROUP | 0x1L, 22, 146, 126, 8
  CONTROL "und Proze_lenkung" -1, "STATIC", WS_CHILD | WS_VISIBLE | WS_GROUP | 0x1L,
22, 154, 126, 8
  CONTROL "(c) FH M³nster 26.02.1992" -1, "STATIC", WS_CHILD | WS_VISIBLE | WS_GROUP
| 0x1L, 22, 181, 126, 8
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP | 0x1L,
47, 193, 80, 14
  CONTROL "Fachhochschule M³nster Abteilung Steinfurt" -1, "STATIC", WS_CHILD |
WS_VISIBLE | 0x1L, 1, 118, 171, 11
  CONTROL "Diplomarbeit " -1, "STATIC", WS_CHILD | WS_VISIBLE | 0x1L, 0, 5, 171, 12
  CONTROL "Autoren:" -1, "STATIC", WS_CHILD | WS_VISIBLE, 3, 25, 32, 9
  CONTROL "Ralf Kronemeyer" -1, "STATIC", WS_CHILD | WS_VISIBLE | 0x1L, 45, 20, 79,
9
  CONTROL "Ralf Gronemann" -1, "STATIC", WS_CHILD | WS_VISIBLE | 0x1L, 45, 30, 79, 9
  CONTROL "Referent:     Professor Dipl.-Ing. Egon Weiner" -1, "STATIC", WS_CHILD |
WS_VISIBLE, 3, 46, 165, 8
  CONTROL "Korreferent: Professor Dr. rer. nat. Axel Bleckmann" -1, "STATIC",
WS_CHILD | WS_VISIBLE, 3, 54, 168, 8
  CONTROL "ROB" -1, "STATIC", WS_CHILD | WS_VISIBLE | 0x3L, 77, 162, 17, 17
  CONTROL "Show ..." 1201, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 127, 20,
40, 18
  CONTROL "FH" -1, "STATIC", WS_CHILD | WS_VISIBLE | 0x3L, 77, 129, 17, 17
  CONTROL "Erstellung einer Windows-Applikation zur Steuerung eines Gelenkarmroboter
mit Hilfe einer Bilder- fassungsanlage unter Anwendung neuronaler Netze" -1,
"STATIC", WS_CHILD | WS_VISIBLE, 3, 79, 170, 29
  CONTROL "Thema:" -1, "STATIC", WS_CHILD | WS_VISIBLE, 3, 69, 32, 9
END


STEUERUNG DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 15, 26, 287, 180
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Steuerung"
BEGIN
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP | 0x1L,
187, 164, 48, 14
  CONTROL "+" ID2_ELBOW_P, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 134, 21, 9,
8
```

```
  CONTROL "-" ID2_ELBOW_N, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 134, 30, 9,
8
  CONTROL "+" ID2_SHOULDER_P, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 169, 38,
9, 8
  CONTROL "-" ID2_SHOULDER_N, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 169, 59,
9, 8
  CONTROL "-" ID2_WAIST_N, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 150, 63, 9,
8
  CONTROL "auf" ID2_HAND_OPEN, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 2, 29,
18, 10
  CONTROL "zu" ID2_HAND_CLOSE, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 2, 41,
18, 9
  CONTROL "+" ID2_WAIST_P, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 90, 63, 9,
8
  CONTROL "+" ID2_HAND_UP, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 68, 14, 9,
8
  CONTROL "-" ID2_HAND_LEFT, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 42, 51,
9, 8
  CONTROL "-" ID2_HAND_DOWN, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 68, 59,
9, 8
  CONTROL "+" ID2_HAND_RIGHT, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 42, 22,
9, 8
  CONTROL "Geschwindigkeit" ID2_SPEED, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x2L, 6,
167, 72, 9
  CONTROL "Schrittweite" ID2_DEGREE, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x2L, 6,
149, 72, 9
  CONTROL "Schrittwinkel : 10_" ID2_TEXTGRAD, "STATIC", WS_CHILD | WS_VISIBLE, 6,
140, 72, 9
  CONTROL "Geschwindigkeit : 16" ID2_TEXTSPEED, "STATIC", WS_CHILD | WS_VISIBLE, 6,
158, 72, 9
  CONTROL "Parameter:" -1, "BUTTON", WS_CHILD | WS_VISIBLE | 0x7L, 3, 95, 79, 83
  CONTROL "hoch" ID2_HOCH, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 209, 27,
32, 10
  CONTROL "runter" ID2_RUNTER, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 209,
132, 32, 10
  CONTROL "links" ID2_LINKS, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 164, 73,
32, 10
  CONTROL "vor" ID2_VOR, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 132, 126, 32,
10
  CONTROL "rechts" ID2_RECHTS, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 252,
116, 32, 10
  CONTROL "zur³ck" ID2_ZURUECK, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 252,
81, 32, 10
  CONTROL "Ursprung" ID2_MO_999, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 135,
164, 48, 14
  CONTROL "Befehlsliste:" -1, "BUTTON", WS_CHILD | WS_VISIBLE | 0x7L, 2, 63, 61, 31
  CONTROL "Hinzuf³gen" ID2_ADD, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 4, 75,
54, 15
  CONTROL "Schrittweite" ID2_GRID, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x2L, 6,
113, 72, 9
  CONTROL "Greifstõrke : 63" ID2_TEXTGRID, "STATIC", WS_CHILD | WS_VISIBLE, 6, 104,
72, 9
  CONTROL "Position:" 0, "BUTTON", WS_CHILD | WS_VISIBLE | 0x7L, 85, 143, 46, 35
  CONTROL "2" ID2_POSEDIT, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,
98, 153, 20, 12
  CONTROL "Nr.:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 86, 155, 11, 9
  CONTROL "Definieren" ID2_DEFINIERN, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP,
86, 166, 42, 11
  CONTROL "Schrittweite" ID2_STEP, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x2L, 6,
131, 72, 9
  CONTROL "Schrittweite : 10 mm" ID2_TEXTMM, "STATIC", WS_CHILD | WS_VISIBLE, 6,
122, 72, 9
END
```

```
BEFEHLSLISTE DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 11, 35, 204, 127
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Befehlsliste"
BEGIN
  CONTROL "BEFEHLE :" ID3_BEFEHLSLISTE, "LISTBOX", WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_VSCROLL | 0x49L, 5, 4, 133, 81
  CONTROL "Hinzuf³gen" ID3_ADD, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 146,
5, 53, 13
  CONTROL "L÷schen" ID3_DEL, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 146, 20,
53, 13
  CONTROL "Laden" ID3_LOAD, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 146, 39,
53, 13
  CONTROL "Speichern" ID3_SAVE, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 146,
54, 53, 13
  CONTROL "Ausf³hren" ID3_DO, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 63, 108,
57, 14
  CONTROL "Demakiere alles" ID3_UNSELECTALL, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 4, 108, 57, 14
  CONTROL "Einzelschrittmodus" ID3_SINGLESTEP, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP | 0x2L, 124, 108, 72, 9
  CONTROL "Wiederholmodus" ID3_REPEAT, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP
| 0x2L, 124, 96, 72, 9
  CONTROL "Modi" -1, "BUTTON", WS_CHILD | WS_VISIBLE | 0x7L, 121, 85, 78, 37
  CONTROL "Makiere alles" ID3_SELECTALL, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 4, 89, 57, 14
  CONTROL "Befehl ..." ID3_COMMAND, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP |
0x2L, 69, 91, 45, 12
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 146, 71, 53, 13
END


BEFEHLSEINGABE DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 80, 64, 191, 43
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Befehlseingabe"
BEGIN
  CONTROL "" 701, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP | 0x88L, 9,
21, 95, 12
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP | 0x1L, 111, 21, 33,
12
  CONTROL "Abbruch" 2, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 152, 21, 33, 12
  CONTROL "Befehl:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 9, 7, 93, 10
END

SIVIPS CURSOR SIVIPS.CUR
MASTER CURSOR MASTER.CUR
ROB ICON ROB.ICO
GEHIRN ICON BRAIN.ICO
FH ICON FH.ICO
rob BITMAP ROB.BMP
all BITMAP ALL.BMP
face BITMAP FACE.BMP
GEHIRNBILD BITMAP GEHIRN.BMP

BRAINMENU  MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
  POPUP "&Datei"
  BEGIN
    MenuItem  "&Neu...", IDM9_NEU
    MenuItem  "&Laden...", IDM9_LADEN
    MenuItem  "&Speichern", IDM9_SPEICHERN
    MenuItem  "Speichern &als ...", IDM9_ALS
    MenuItem  SEPARATOR
```

```
    MenuItem  "&Brain beenden", IDM9_ENDE
  END
  POPUP "&Training"
  BEGIN
    MenuItem  "&Hohe Prioritõt", IDM9_PRIOHIGH
    MenuItem  "&Mittlere Prioritõt", IDM9_PRIOMIDDLE
    MenuItem  "&Niedrige Prioritõt", IDM9_PRIOLOW
    MenuItem  SEPARATOR
    MenuItem  "&Start", IDM9_START
    MenuItem  "An&zeigen", IDM9_ANZEIGEN
    MenuItem  SEPARATOR
    MenuItem  "&Reset", IDM9_RESET
  END
  POPUP "&Erkennung"
  BEGIN
    MenuItem  "&Start", IDM9_ERKENNUNG
    MenuItem  "&Testphase", IDM9_TEST
    MenuItem  SEPARATOR
    MenuItem  "&Robotereinsatz", IDM9_ROBOTEREINSATZ
  END
  POPUP "&Option"
  BEGIN
    MenuItem  "&Edit Teiledaten ...", IDM9_EDITTEILEDATEN
    MenuItem  SEPARATOR
    POPUP "&Anzeige"
    BEGIN
      MenuItem  "&Gro_", IDM9_GROSS
      MenuItem  "&Mittel", IDM9_MITTEL
      MenuItem  "&Klein", IDM9_KLEIN
    END
  END
END



NEUESNETZ DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 36, 23, 178, 187
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Neues Netz anlegen"
BEGIN
  CONTROL "Erkennungsmerkmale:" 0, "BUTTON", WS_CHILD | WS_VISIBLE | 0x7L, 1, 0,
104, 165
  CONTROL "Gesamtflõche" ID11_TOTALAREA, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP | 0x2L, 4, 11, 96, 10
  CONTROL "max. Trõgheitsmoment" ID11_MAJOR, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP | 0x2L, 4, 24, 96, 10
  CONTROL "min.Trõgheitsmoment" ID11_MINOR, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP | 0x2L, 4, 37, 96, 10
  CONTROL "L÷cheranzahl" ID11_NHOLES, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP |
0x2L, 4, 50, 96, 10
  CONTROL "Umfang" ID11_PERIMETER, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP |
0x2L, 4, 63, 96, 10
  CONTROL "max. Schwerpunktabstand" ID11_RMAX, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP | 0x2L, 4, 76, 96, 10
  CONTROL "" ID11_S0, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x3L, 121, 75, 9, 46
  CONTROL "" ID11_S1, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x3L, 149, 75, 9, 46
  CONTROL "Neuronenanzahl:" 0, "BUTTON", WS_CHILD | WS_VISIBLE | 0x7L, 107, 64, 69,
71
  CONTROL "9" ID11_T0, "STATIC", WS_CHILD | WS_VISIBLE, 121, 124, 10, 10
  CONTROL "9" ID11_T1, "STATIC", WS_CHILD | WS_VISIBLE, 149, 124, 10, 10
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 6, 167, 42, 17
  CONTROL "Anzahl der Layer : 1" ID11_LAYER, "SCROLLBAR", WS_CHILD | WS_VISIBLE |
0x2L, 107, 52, 69, 9
  CONTROL "" ID11_TEILE, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x2L, 107, 11, 69, 9
```

```
  CONTROL "Anzahl der Teile : 2" ID11_TEILETEXT, "STATIC", WS_CHILD | WS_VISIBLE,
107, 1, 69, 9
  CONTROL "Hiddenlayer: 1" ID11_LAYERTEXT, "STATIC", WS_CHILD | WS_VISIBLE, 107, 42,
69, 9
  CONTROL "Abbruch" 2, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 58, 167, 42, 17
  CONTROL "min. Schwerpunktabstand" ID11_RMIN, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP | 0x2L, 4, 89, 96, 10
  CONTROL "durchschnittlicher Abstand" ID11_AVRAD, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP | 0x2L, 4, 102, 96, 10
  CONTROL "Lõnge" ID11_LAENGE, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP | 0x2L,
4, 115, 96, 10
  CONTROL "Breite" ID11_BREITE, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP | 0x2L,
4, 128, 96, 10
  CONTROL "" ID11_SAMPLE, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x2L, 107, 31, 69, 9
  CONTROL "Sample pro Teil: 1" ID11_SAMPLETEXT, "STATIC", WS_CHILD | WS_VISIBLE,
107, 21, 69, 9
  CONTROL "Grauwert-Histogramm" ID11_GRAUWERTE, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP | 0x2L, 4, 144, 91, 14
  CONTROL "" 0, "BUTTON", WS_CHILD | WS_VISIBLE | 0x7L, 2, 138, 98, 22
  CONTROL "Robotereinsatz" 0, "BUTTON", WS_CHILD | WS_VISIBLE | 0x7L, 107, 137, 69,
47
  CONTROL "kein" ID11_ROBOTER_KEIN, "BUTTON", WS_CHILD | WS_VISIBLE | 0x4L, 111,
146, 53, 13
  CONTROL "nur Training" ID11_ROBOTER_TRAINING, "BUTTON", WS_CHILD | WS_VISIBLE |
0x4L, 111, 158, 53, 13
  CONTROL "komplett" ID11_ROBOTER_KOMPLETT, "BUTTON", WS_CHILD | WS_VISIBLE | 0x4L,
111, 170, 53, 13
END


NEURONENDATEN DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 34, 35, 180, 62
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Neurondaten"
BEGIN
  CONTROL "" ID13_POTENTIAL, "EDIT", WS_CHILD | WS_VISIBLE | WS_DISABLED | WS_BORDER
| WS_TABSTOP, 82, 3, 89, 11
  CONTROL "" ID13_AUSGANG, "EDIT", WS_CHILD | WS_VISIBLE | WS_DISABLED | WS_BORDER |
WS_TABSTOP, 82, 15, 89, 11
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 122, 37, 49, 12
  CONTROL "Inneres Potential :" 0, "STATIC", WS_CHILD | WS_VISIBLE, 8, 5, 70, 12
  CONTROL "Ausgangswert :" 0, "STATIC", WS_CHILD | WS_VISIBLE, 8, 17, 70, 12
  CONTROL "" ID13_SCROLLBAR, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x3L, 8, 28, 9, 31
  CONTROL "" ID13_WEDIT, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 47,
37, 65, 12
  CONTROL "W1:" ID13_WTEXT, "STATIC", WS_CHILD | WS_VISIBLE, 21, 39, 23, 10
END


TEILEDATEN DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 63, 9, 103, 169
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Festlegung der Teiledaten"
BEGIN
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 26, 149, 46, 13
  CONTROL "" ID15_SCROLLTEILE, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x2L, 4, 60, 94,
10
  CONTROL "" ID15_NAME, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 32,
3, 66, 12
  CONTROL "Name:" ID15_NAME, "STATIC", WS_CHILD | WS_VISIBLE, 2, 5, 28, 9
  CONTROL "" ID15_GROESSE, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,
32, 18, 66, 12
  CONTROL "Gr÷sse:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 2, 20, 28, 9
```

```
  CONTROL "Teilnummer: 10" ID15_TEILENUMMER, "STATIC", WS_CHILD | WS_VISIBLE, 4, 49,
64, 10
  CONTROL "" ID15_SCROLLSAMPLE, "SCROLLBAR", WS_CHILD | WS_VISIBLE | 0x2L, 4, 82,
94, 10
  CONTROL "Samplenummer: 5" ID15_SAMPLENUMMER, "STATIC", WS_CHILD | WS_VISIBLE, 4,
71, 64, 10
  CONTROL "Sample aufnehmen" ID15_SAMPLEAUFNEHMEN, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 5, 97, 93, 12
  CONTROL "" ID15_FACHNUMMER, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER |
WS_TABSTOP, 32, 33, 66, 12
  CONTROL "Fachnr.:" -1, "STATIC", WS_CHILD | WS_VISIBLE, 2, 35, 28, 9
  CONTROL "Starte Roboter" ID15_STARTEROBOTER, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 5, 113, 93, 12
  CONTROL "Greifh÷he definieren" ID15_GREIFHOEHE, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 5, 129, 93, 12
END


FILEOPERATION DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 48, 29, 184, 170
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | 0x80L
BEGIN
  CONTROL "" ID6_FILES, "LISTBOX", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_VSCROLL |
0x3L, 6, 45, 60, 122
  CONTROL "" ID6_DIRECTORIES, "LISTBOX", WS_CHILD | WS_VISIBLE | WS_BORDER |
WS_VSCROLL | 0x3L, 75, 45, 60, 122
  CONTROL "Abbruch" 2, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 140, 153, 40,
12
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 140, 131, 40, 12
  CONTROL "" ID6_FILE, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 48,
2, 86, 13
  CONTROL "Filename:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 6, 4, 37, 10
  CONTROL "Files:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 6, 33, 48, 9
  CONTROL "Directories:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 75, 33, 48, 9
  CONTROL "Directory:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 6, 18, 37, 10
  CONTROL "D:\\" ID6_DIRECTORY, "STATIC", WS_CHILD | WS_VISIBLE, 48, 18, 134, 11
END

POSITIONSANA DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 26, 47, 143, 121
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Positionsanalyse"
BEGIN
  CONTROL "Move" ID8_MOVE, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 92, 90, 36,
12
  CONTROL "Neu" ID8_NEW, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 92, 106, 36,
12
  CONTROL "Transfer" ID8_ROBOTER, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 92,
74, 36, 12
  CONTROL "" ID8_POSITION, "LISTBOX", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_VSCROLL
| 0x1L, 3, 1, 39, 118
  CONTROL "Speichern" ID8_SAVE, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 51,
90, 36, 12
  CONTROL "Laden" ID8_LOAD, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 51, 74,
36, 12
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 51, 106, 36, 12
  CONTROL "Positionsnr.:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 51, 3, 45, 9
  CONTROL "X-Wert:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 51, 13, 45, 9
  CONTROL "Y-Wert:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 51, 23, 45, 9
  CONTROL "Z_Wert:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 51, 33, 45, 9
  CONTROL "Hand-Pitch:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 51, 43, 45, 9
  CONTROL "Hand-Roll:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 51, 53, 45, 9
  CONTROL "Hand-Grip:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 51, 63, 45, 9
  CONTROL "" ID8_POSITIONSNR, "STATIC", WS_CHILD | WS_VISIBLE, 93, 3, 46, 9
  CONTROL "" ID8_XWERT, "STATIC", WS_CHILD | WS_VISIBLE, 93, 13, 46, 9
```

```
  CONTROL "" ID8_YWERT, "STATIC", WS_CHILD | WS_VISIBLE, 93, 23, 46, 9
  CONTROL "" ID8_ZWERT, "STATIC", WS_CHILD | WS_VISIBLE, 93, 33, 46, 9
  CONTROL "" ID8_HANDPITCH, "STATIC", WS_CHILD | WS_VISIBLE, 93, 43, 46, 9
  CONTROL "" ID8_HANDROLL, "STATIC", WS_CHILD | WS_VISIBLE, 93, 53, 46, 9
  CONTROL "" ID8_HANDGRIP, "STATIC", WS_CHILD | WS_VISIBLE, 93, 63, 46, 9
END



VISUALISIERUNG DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 21, 45, 204, 105
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Visualisierung"
BEGIN
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 50, 92, 35, 11
  CONTROL "Aktuell" ID10_AKTUELL, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 121,
92, 35, 11
  CONTROL "0,0,0" ID10_DATEN, "STATIC", WS_CHILD | WS_VISIBLE, 44, 5, 121, 9
  CONTROL "X,Y,Z:" 0, "STATIC", WS_CHILD | WS_VISIBLE, 20, 5, 23, 7
END

FACEBOX DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 2, 35, 159, 122
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Kronemeyer / Gronemann          "
BEGIN
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 65, 108, 32, 12
END

ROBINIT DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 55, 53, 206, 75
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | 0x80L
CAPTION "Roboterinitialisierung"
BEGIN
  CONTROL "Bitte schalten Sie den Movemaster ein und die Toolbox aus." 0, "STATIC",
WS_CHILD | WS_VISIBLE, 2, 2, 204, 8
  CONTROL "Sollte der Roboter schon einmal grundinitialisiert worden sein," 0,
"STATIC", WS_CHILD | WS_VISIBLE, 2, 12, 204, 8
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 35, 58, 39, 12
  CONTROL "Abbruch" 2, "BUTTON", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 126, 58, 39, 12
  CONTROL "Nest - Kommando senden" 1900, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP | 0x2L, 50, 44, 99, 12
  CONTROL "so er³brigt sich die Sendung des Nest - Kommandos." 0, "STATIC", WS_CHILD
| WS_VISIBLE, 2, 22, 204, 8
  CONTROL " " 1901, "STATIC", WS_CHILD | WS_VISIBLE | 0x1L, 0, 33, 206, 10
  CONTROL " " 1902, "STATIC", WS_CHILD | WS_VISIBLE | 0x1L, 0, 33, 206, 10
END
```

7.6. Das Sivips-Programm NEUROROB.PRS

```
Main Bilderfassung;

INT i, Schwelle:=58,BildFlag;
BYTE RP_Parts, RP_Blobs, Pruef[4], Element;
WORD RP_Laenge:=128, RP_Datenlaenge:=130, Histogramm[64], Grau,RP_P:=3;
CHAR Los_gehts,Warte;
REAL XCent,YCent,XCent_Hilf,YCent_Hilf;

STRUCT Datensatz BEGIN
                    BYTE RAWBLOBI;
                    BYTE NACHF;
                    BYTE PARENT;
                    BYTE COLOR;
                    BYTE NHOLES;
                    BYTE MODEL;
                    BYTE EXIMIN;
                    BYTE EXIMAX;
                    BYTE EXJMIN;
                    BYTE EXJMAX;
                    BYTE PARITY;
                    BYTE DUMMY;
                    WORD BOUNDARYSTART;
                    WORD CENTROID;
                    WORD MAJORCT1;
                    WORD MOJORCT2;
                    WORD MINORCT1;
                    WORD MINORCT2;
                    WORD RMINLOG;
                    WORD RMAXLOG;
                    WORD XPERIM;
                    WORD YPERIM;
                    WORD DPERIM;
                    WORD NCELLS;
                    WORD TOTALCELLS;
                    DINT SUM[5];
                    REAL AREA;
                    REAL XCENT;
                    REAL YCENT;
                    REAL MAJOR;
                    REAL MINOR;
                    REAL MAJORANG;
                    REAL PERIMETER;
                    REAL TOTALAREA;
                    REAL RMIN;
                    REAL RMAX;
                    REAL RMINANG;
                    REAL RMAXANG;
                    REAL AVRAD;
                    REAL LAENGE;
                    REAL BREITE;
                    REAL BOX_XCENT;
                    REAL BOX_YCENT;
                    REAL NNDIST;
                 END;                              Struktur fuer RLDAT
STRUCT Datensatz Merkm_Liste;
CONST C:="C     $";
ERR_Switch:=2;
DEVEA(2,0);                                       Bildschirmparametrierung
DEVEA(8,4,8);                                     Schnittstellenkonfiguration
```

```
BINC(58);
RP_Blobs:=0;
RP_Parts:=0;
BildFlag:=1;
DELWND("B4.11 $");
DISP(,"B4.11 $");
FOR i:= 0 TO 125 BY 5 DO                          Praesentation
    BCIRC("B4.11 $",128,128,i);
ENDFOR;
WHILE (1=1) DO
    DELWND("W2    $");                            Endlosschleife
    LOC(1,1);
    WRITELN(2,"RARA-Bildverarbeitung");
    DELAY();
ENDWHILE;
END;

DEF DELAY();                                      Warten auf Auftrag vom Client
    Los_gehts:=0;
    WHILE(Los_gehts=0) DO
         READ(81,Los_gehts);
    ENDWHILE;
    STOR(C,"G1    $");                            Bilder speichern
    STOR(C,"G2.1  $");
    STOR(C,"R     $");
    STOR(C,"B4.12 $");
    STOR(C,"B3.1  $");
    SEGBLOB(10,RP_Parts,RP_Blobs,1.1683,0.9926);    Binaerisierung der Szene
    IF(RP_Parts>0)THEN
        PARTDATEN();
    ENDIF;
    IF(Los_gehts="A") THEN
        SENDEMERK();                              Merkmale zum Client
    ENDIF;
    IF(Los_gehts="P") THEN
        SCHWELLERAUF();                           Binaerisierungschwelle rauf
    ENDIF;
    IF(Los_gehts="M") THEN
        SCHWELLERUNTER();                         Binaerisierungschwelle runter
    ENDIF;
    IF(Los_gehts="C") THEN
        DISP("G1    $",);                         Graubild
        BildFlag:=1;
    ENDIF;
    IF(Los_gehts="E") THEN
        DISP(,"B3.1  $");                         Binaerbild
        BildFlag:=2;
    ENDIF;
    IF(Los_gehts="H") THEN
        CHIST("G2.1  $",1,);
        DHIST("B4.11 $",1);                       Histogramm
        DISP(,"B4.11 $");
        BildFlag:=5;
    ENDIF;
    IF(Los_gehts="O") THEN
        IF(RP_Parts>0) THEN
            CENTER("B4.12 $",Element);            Orientierung/Schwerpunkt
            BNDBOX("B4.12 $",Element);
            DISP(,"B4.12 $");
        ENDIF;
        BildFlag:=6;
    ENDIF;
    IF(Los_gehts="D") THEN
```

```
        DISP(,C);                                   Live-Binaerbild
        BildFlag:=3;
    ENDIF;
    IF(Los_gehts="B") THEN
        DISP(C,);                                   Live-Graubild
        BildFlag:=4;
    ENDIF;
    IF(Los_gehts="T") THEN
        RUNTIME();                                  Runtimemodus(Robotereinsatz)
    ENDIF;
    IF(Los_gehts="X") THEN
        WRITELN(81,Merkm_Liste.XCENT);              Schwerpunkt
        WAIT();
        WRITELN(81,Merkm_Liste.YCENT);
    ENDIF;
END;

DEF WAIT();                                         Warte auf Client
    READ(81,Warte);
    WHILE(Warte<>"A") DO
        READ(81,Warte);
    ENDWHILE;
END;

DEF SENDEMERK();                                    Merkmale senden
    CHIST("G2.1  $",1,);
    IF(BildFlag=1) THEN
        DISP("G1    $",);
    ENDIF;
    IF(BildFlag=2) THEN
        DISP(,"B3.1  $");
    ENDIF;
    IF(BildFlag=3) THEN
        DISP(,C);
    ENDIF;
    IF(BildFlag=4) THEN
        DISP(C,);
    ENDIF;
    IF(BildFlag=5) THEN
        DHIST("B4.11 $",1);
        DISP(,"B4.11 $");
    ENDIF;
    GETLIS(HIST,1,RP_Laenge,Histogramm);
    IF(BildFlag=6) THEN
        CENTER("B4.12 $",Element);
        BNDBOX("B4.12 $",Element);
        DISP(,"B4.12 $");
    ENDIF;
    IF(RP_Parts>0) THEN
        WRITELN(81,Merkm_Liste.TOTALAREA);
        WAIT();
        WRITELN(81,Merkm_Liste.MAJOR);
        WAIT();
        WRITELN(81,Merkm_Liste.MINOR);
        WAIT();
        WRITELN(81,Merkm_Liste.NHOLES);
        WAIT();
        WRITELN(81,Merkm_Liste.PERIMETER);
        WAIT();
        WRITELN(81,Merkm_Liste.RMAX);
        WAIT();
        WRITELN(81,Merkm_Liste.RMIN);
        WAIT();
```

```
        WRITELN(81,Merkm_Liste.AVRAD);
        WAIT();
        WRITELN(81,Merkm_Liste.LAENGE);
        WAIT();
        WRITELN(81,Merkm_Liste.BREITE);
        WAIT();
    ELSE
        FOR i:=0 TO 9 DO
            WRITELN(81,0);
            WAIT();
        ENDFOR;
    ENDIF;
    FOR i:=0 TO 63 DO
        Grau:=Histogramm[i];
        WRITELN(81,Grau);
        WAIT();
    ENDFOR;
    WRITELN(81,Merkm_Liste.MAJORANG);
END;

DEF SCHWELLERUNTER();
    Schwelle:=Schwelle-1;
    BINC(Schwelle);
    DISP(,C);
END;

DEF SCHWELLERAUF();
    Schwelle:=Schwelle+1;
    BINC(Schwelle);
    DISP(,C);
END;

DEF RUNTIME();
    LOC(25,1);
    IF(RP_Parts<>1) THEN
        WRITELN(81,"N");                          Pruefen auf ein Teil
        WRITELN(2,"Teileanzahl !");
        GOTO RUNMARKE;
    ENDIF;                                        aktuelle Position merken
    XCent_Hilf:=Merkm_Liste.XCent;
    YCent_Hilf:=Merkm_Liste.YCent;
    STOR(C,"R     $");
    SEGBLOB(10,RP_Parts,RP_Blobs,1.1683,0.9926);
    IF(RP_Parts<>1) THEN
        WRITELN(81,"N");                          Pruefen auf ein Teil
        WRITELN(2,"Teileanzahl !");
        GOTO RUNMARKE;
    ENDIF;
    PARTDATEN();
    XCent:=Merkm_Liste.XCent;                     neue Position merken
    YCent:=Merkm_Liste.YCent;
    IF((ABS(XCent-XCent_Hilf)>0.7) or (ABS(YCent-YCent_Hilf)>0.7)) THEN
        WRITELN(2,"Bewegung !");
        WRITELN(81,"N");               Pruefen ob Bewegung im Toleranzbereich
    ELSE
        WRITELN(81,"J");
        WRITELN(2,"Bild OK");
    ENDIF;
RUNMARKE: LOC(1,1);
END;

DEF PARTDATEN();                                  Pruefen ob Teil Part ist
    Pruef[1]:=0;
```

```
    Pruef[2]:=255;
    WHILE(Pruef[2]<>0) DO
        Element:=Pruef[1];
        GETLIS(RLDAT,Element,RP_P,Pruef);
    ENDWHILE;
    MOMBLOB(Element);
    BNDBLOB(Element);
    GETLIS(RLDAT,Element,RP_Datenlaenge,Merkm_Liste);
END;
```

8. Literaturverzeichnis

[1]	Microsoft Corporation	Software Development Kit - Guide to Programming - Reference Volume 1/2
[2]	Borland GmbH	Borland 'C++'-Compiler - Einführung - Programmierhandbuch - Referenzhandbuch - Benutzerhandbuch - Whitewater Ressourcen Editor
[3]	Dipl.-Ing. Heinz Ennen Dipl.-Ing. Matthias Rhein	Diplomarbeit 06. Juni 1991 Steuerung eines Gelenkarmroboters mit Hilfe eines Bilderfassungssystems
[4]		Mitsubishi Industrial Micro-Robot System Model RV-M2 Instruction Manual 1990
[5]		Siemens Videomat PS Betriebsanleitung Programmiersystem SIVIPS, Benutzerhandbuch
[6]	Honekamp & Wilken	Windows Intern Data Becker GmbH
[7]	Willi Bruns	Künstliche Intelligenz in der Technik Carl Hanser Verlag, München Wien 1990
[8]	Eberhard Schöneburg Nikolaus Hansen Andreas Gawelczyk	Neuronale Netzewerke Markt & Technik Verlag AG 1990
[9]	Bauder	Windows 3 Programmentwicklung Data Becker GmbH
[10]	J. Stanley / E. Bak	Neuronale Netze Systhema Verlag GmbH
[11]	Klaus Peter Kratzer	Neuronale Netze Carl Hanser Verlag München Wien

[12] Rudi Kost
Word für Windows 1.1
Das Kompendium
Markt & Technik Verlag AG

[13] Brian W. Kerninghan
Denis M. Ritchie
Programmieren in C
Carl Hanser Verlag München Wien

[14] P. Ebel / H. Retzlaff
Word für Windows
Alles auf einen Blick
Data Becker GmbH