3 Kommunikation der Komponenten

Um den Betrieb einer Gehmaschine zu ermöglichen, ist die reibungslose Kommunikation aller beteiligten Komponenten sicherzustellen. Eine schematische Darstellung dieser Komponenten findet sich in Abbildung 23. Auf der untersten Ebene der Datenübertragung steht dabei das Messen und Versenden elektrischer Größen zwischen der Gehmaschine und den entsprechenden Hardwarekomponenten wie Analog/Digital-Wandler, Digitale Input/Output-Karten und Einrichtungen zur Erzeugung pulsweitenmodulierter Signale zur Ansteuerung der Servomotoren.

Dieser Ebene übergeordnet ist die Kommunikation der jeweiligen Rechner mit den genannten Hardwarekomponenten. Diese Rechner ermitteln auf Nachfrage die entsprechenden Meßwerte oder versenden die gewünschten Stellgrößen.

Darüber ist das Koordinationssystem der Gehmaschine angesiedelt. Üblicherweise befinden sich die Kernelemente der Koordination auf einem Rechner. Einzelne, besonders zeitaufwendige Aufgaben können auf andere Rechner ausgelagert werden. Dies bietet sich beispielsweise für aufwendigere Visualisierungen an, um die Ausführungszeit des koordinierenden Rechners nicht zu stark zu beeinträchtigen.

Das aus der Arbeit von [Ame95] vorhandene System sollte auf einer modernen Betriebssystem-Plattform neu aufgebaut werden. Die Anforderungen an die Neugestaltung waren:

- komplette Trennung der Koordination und der Hardwareansteuerung,
- Aufteilung der Hardwareansteuerung in unabhängige Module,
- Bildung von Layern zur Erfüllung der verschiedenen Aufgaben,
- Abstraktion der Hardware für den Benutzer,
- logische Abbildung der Maschinenelemente in der Software,
- allgemeine und komplette Abbildung der Hardwarefähigkeiten in den Schnittstellen zur Hardwareansteuerung,
- Möglichkeit zur Kommunikation zu diesen Modulen sowohl auf dem lokalen System als auch über ein Netzwerk und
- Verwendung bereits vorhandener Bibliotheken, wie z. B. für Neuronale Netze oder Bahnplanung.

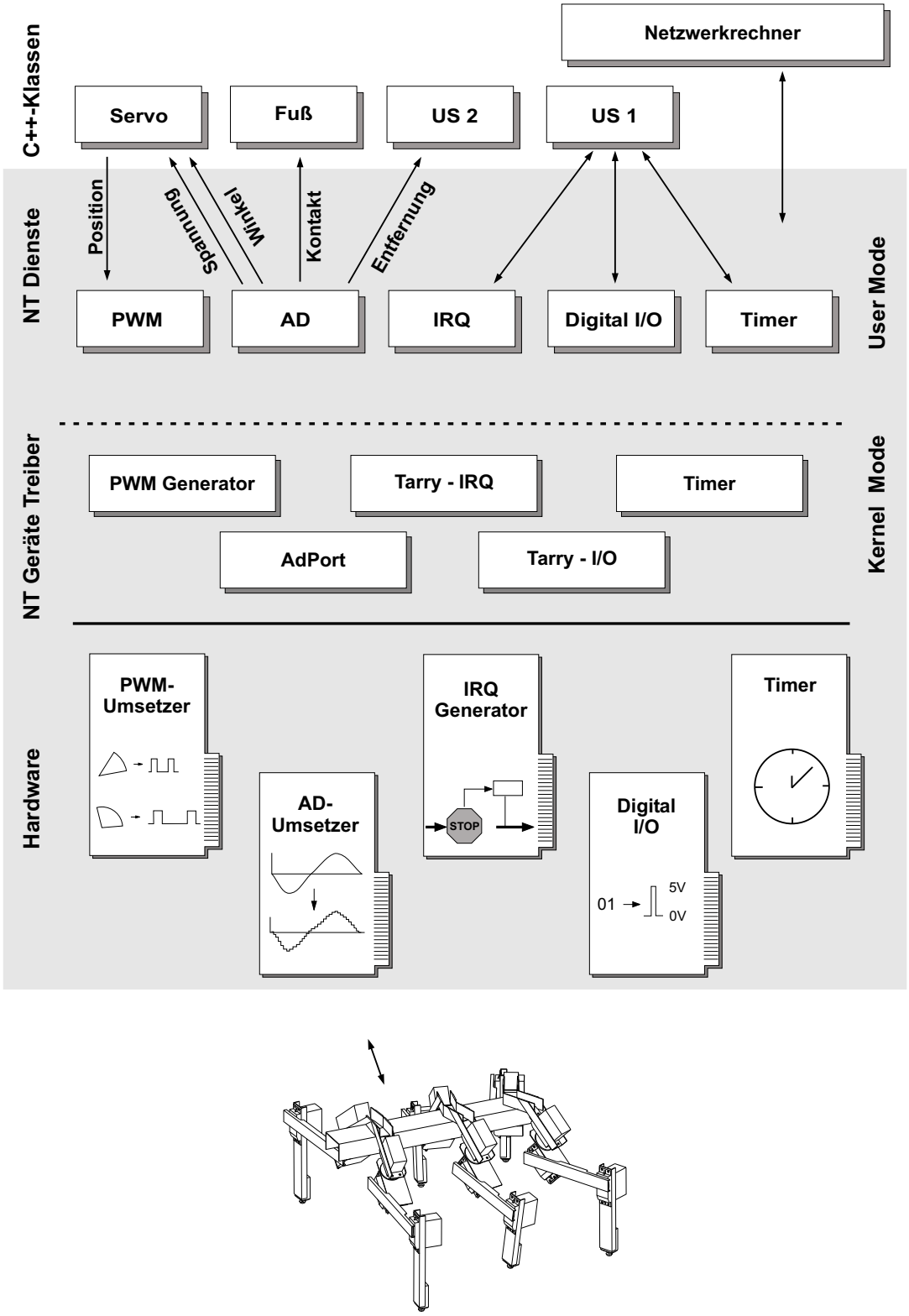

Abbildung 23: Schematische Darstellung der Ansteuerung

Bei der Architektur und der Implementation der verwendeten Module standen also unter anderem Anforderungen der Portierbarkeit und der Netzwerkfähigkeit im Vordergrund. Die einzelnen Module lassen sich dabei in zwei Bereiche einteilen (siehe Abbildung 23). Der erste Bereich beinhaltet die Gerätetreiber, die direkt mit der Hardware kommunizieren sowie die Dienste, die diese Funktionalität nach außen zur Verfügung stellen. Diese Module enthalten zwangsläufig maschinen- und systemabhängigen Programmcode. Im vorliegenden Fall wurde zur Implementation dieser Module Microsoft Windows NT 4.0 gewählt. Die genannten Elemente sind als sogenannte „Device Driver“ beziehungsweise „Services“ ausgeführt. Der zweite Bereich besteht aus allen weiteren Modulen. Diese Module sollen vom Trägersystem, das die Hardwaresteuerung übernimmt, unabhängig sein. Dies beinhaltet insbesondere die Möglichkeit, ein anderes Betriebssystem zu verwenden und die notwendigen Daten über eine hinreichend schnelle Netzwerkverbindung zu übertragen.

Um die notwendige prozeß- und auch rechnerübergreifende Kommunikation der Komponenten zu gewährleisten, waren entsprechende Mechanismen erforderlich. Dabei mußte eine Methode gewählt werden, die auf allen relevanten Systemen zur Verfügung steht und dabei einen möglichst geringen Overhead erzeugt.

Als Übertragungsprotokoll wurde das RPC (Remote Procedure Call) Protokoll gewählt, das im Rahmen der DCE (Desktop Computing Environment) der OSF (Open Software Foundation) standardisiert ist [RT93]. Als zusätzlicher Vorteil ergibt sich dabei, daß die Subsysteme von Windows NT intern sehr intensiv über RPCs mit einem als LPC (Local Procedure Call) oder auch LRPC (Lightweight Remote Procedure Call) bezeichneten Protokoll kommunizieren. Damit ist es möglich, eine gleichzeitig portierbare und netzwerkfähige Kommunikationsstruktur zu verwenden, die auf dem gewählten Zielsystem ausgesprochen effizient umgesetzt ist. Versuche haben ergeben, daß die Geschwindigkeit dieser lokalen Kommunikation auch mit wesentlich weniger universellen Methoden nur unwesentlich zu steigern wäre.

Die Implementierung mit Hilfe von RPCs hat sich als zuverlässig erwiesen und ermöglicht eine kompatible Datenübermittlung zu den hardwarenahen Windows NT Diensten [FGLK98]. Die Kompatibilität konnte durch Koordination der Maschine über Programme unter den Betriebssystemen HP-UX und SUSE Linux über das Netzwerk nachgewiesen werden.

3.1 Beteiligte Hardwarekomponenten

Die eingangs erwähnten Hardwarekomponenten, die mit der Maschine kommunizieren, entstammen drei verschiedenen Bereichen:

- Digitale Input/Output Karten (Digital-I/O),
- Erzeugung pulsweitenmodulierter Signale (PWM-Signale) und
- Analog/Digital-Wandler (A/D-Wandler).

3.1.1 Digital-I/O

Über eine digitale Digital-I/O Karte können digitale Schaltzustände gemessen und erzeugt werden. Die zur Kontrolle der Gehmaschine verwendeten Karten enthalten zudem jeweils mehrere Timer-Bausteine. Dadurch wird es möglich, Schaltereignisse mit dem Starten oder Stoppen von Zählern zu verbinden. Diese Funktionalität kann unter anderem zur Messung von Signallaufzeiten, beispielsweise von Ultraschall-Signalen, verwendet werden (siehe auch Kapitel 2.3.5). Die weiter unten angesprochene PWM-Erzeugung beruht ebenfalls auf Timer-Bausteinen der Digital-I/O Karten.

Die hier verwendeten Karten der Firmen Kolter und Wasco haben jeweils 48 Kanäle und drei Zählbausteine und basieren auf dem Ein-/Ausgabebaustein 82C55 und dem Timer-Baustein 82C54 bzw. 82C53. Die Karten arbeiten mit TTL-Pegeln[2] für die ein- und ausgehenden Signale.

Neben den bereits erwähnten Haupteinsatzgebieten, der Ultraschallmessung und der Implementierung der PWM-Erzeuger, können die Karten zur Messung einfacher Schaltzustände dienen.

3.1.2 PWM-Generator

Die Ansteuerung der verwendeten Servomotoren erfolgt mit Hilfe pulsweitenmodulierter Eingangssignale. Über die Dauer der anliegenden Spannung wird die gewünschte Soll-Position an den internen Servoregler übertragen (siehe auch Abschnitt 2.3.1).

Dabei entspricht der Rechtsausschlag eines Servomotors einer Impulsdauer von ca. 850 μs, der Linksausschlag einer Impulsdauer von ca. 2250 μs. Die Stellgrößen der Servomotoren werden über drei Datenkanäle an die Maschine übertragen. Zu diesem Zweck wird das sogenannte Multiplexing-Verfahren verwendet, bei dem Signale auf einem Kanal nacheinander gesendet werden. Auf der Empfängerseite, der Gehmaschine selbst, werden diese Daten wieder getrennt und an die einzelnen Stellglieder verteilt. Da jeweils sieben Servomotoren über eine Steuerleitung versorgt werden, ergibt sich eine maximale Sendedauer von 15,75 ms bei Vollausschlag aller Motoren. Diese Zeit begrenzt auch den maximal möglichen Takt, mit dem die entsprechenden Winkel neu an die Maschine übermittelt werden können.

[2] Transistor -Transistor Logic (TTL) sind binäre Signale mit einem Pegel von $U_E \leq 0{,}8$ V, $U_A \leq 0{,}4$ V für den *Low*-Pegel und $U_E \geq 2$ V, $U_A \geq 2{,}4$ V für den *High*-Pegel bei einer Betriebsspannung von 5 V.

3.1.3 A/D-Wandler

Die Messung von elektrischen und nichtelektrischen Größen erfolgt in den meisten Fällen mit Hilfe einer elektrischen Spannung, die sich proportional zur Meßgröße verhält. Um diese Spannungen mit einem Digitalrechner weiterzuverarbeiten, werden A/D-Wandler eingesetzt, die diese Spannung in ein quantisiertes, zeitdiskretes digitales Signal umsetzen. Die hier verwendeten Karten und ihre grundlegenden Eigenschaften sollen kurz vorgestellt werden[Sch98].

Die Auflösung eines A/D-Wandlers wird in der einfachsten Form in Bit angegeben. Die für TARRY II verwendeten Umsetzer basieren auf dem Baustein ADC774 und besitzen eine Auflösung von 12 Bit. Daraus resultiert die Quantisierung des Meßbereiches in $2^{12} = 4096$ Werte. Bei einem unipolaren Spannungsbereich von $0 - 5$ Volt, in dem an den Maschinen gemessen wird, ergibt sich damit eine Auflösung von

$$\Delta U = \frac{5\ \mathrm{V}}{2^{12}} = 1{,}22\ \mathrm{mV} = \mathrm{LSB}\ \text{(Least significant byte)}$$

oder, bezogen auf die Maximalspannung

$$A_{\%} = \frac{100\ \%}{2^{12}} = 0{,}024\ \%.$$

Der verwendete Umsetzer arbeitet nach dem Prinzip der sukzessiven Approximation. Für jeden der sechzehn Kanäle wird eine Wandlungszeit von 7 μs angegeben.

Im Vergleich zur Messung statischer Signale ist die binäre Codierung dynamischer Signale schwieriger. Während der Umsetzzeit des A/D-Wandlers darf sich die Eingangsgröße höchstens um $\frac{1}{2}$ LSB ändern, um das Meßergebnis nicht zu verfälschen. Bei den oben angegebenen Parametern wäre das in diesem Fall eine Änderung von höchsten 0,61 mV in der Wandlungszeit. Um diese Problematik zu umgehen, kann die Verwendung des „Sample and Hold“ hilfreich sein. Dieser Modus gestattet es, die Eingangsspannung vor der Messung zu speichern und dann den danach gemessenen Wert zu wandeln. Die Gesamtmeßzeit verlängert sich dabei allerdings um die Einschwingzeit der Speicher. An der verwendeten Meßkarte sind je vier solcher „Sample and Hold“ Kanäle vorhanden.

Versuche mit der Messung der Stellwinkel haben allerdings gezeigt, daß auf die Verwendung der Zwischenspeicherung verzichtet werden kann. Die Servomotoren bewegen sich verhältnismäßig langsam, so daß die Abweichungen zwischen den beiden Messungen nicht relevant waren.

3.2 Kommunikation der Rechner mit den Hardwarekomponenten

Um die Kommunikation des Rechners mit den verwendeten Hardwarekomponenten zu ermöglichen, müssen entsprechende Treiber zur Verfügung gestellt werden. Aufgrund der breiten Verfügbarkeit von standardisierten Hardwarekomponenten aus

dem Bereich der PC-Technologie, und um die Weiterverwendung der bereits für TARRY I eingesetzten Meßkarten zu ermöglichen, kam als Rechnerhardware damit nur die PC-Plattform in Frage.

Zum Zeitpunkt der Neukonzeption der Softwarearchitektur Ende 1996 boten sich zur Verwendung auf dieser Plattform nur Microsoft Windows NT 4.0 und Linux als anspruchsvolle Betriebssysteme an. Beide bieten die notwendigen Grundlagen, um das gewünschte System aufzubauen. Aufgrund der zur Verfügung stehenden Werkzeuge und Informationen zur Treiberprogrammierung fiel die Wahl auf Windows NT. Die vorgestellte Architektur ließe sich aber auch komplett auf einem Linux System realisieren.

Da für die bereits bei TARRY I verwendete Meßkarten keine Kernel Treiber für Windows NT erhältlich waren, ist die entsprechende Software am Fachgebiet Mechanik der Gerhard-Mercator-Universität Duisburg erstellt worden. Sie muß im sogenannten *Kernel Mode* des Prozessors betrieben werden. Lediglich von hier aus ist es möglich, direkt auf die Register und Speicher der Hardware zuzugreifen. Allerdings erfordert dies deutlich umfangreichere Kenntnisse als die gängige Programmierung im *User Mode*. Fehlerhafte Module im *Kernel Mode* bringen das System in der Regel zum Stillstand, da es hier bei Erkennen von ungültigen Manipulationen die Gesamtintegrität bedroht sieht und die Arbeit beendet. Ebenso ist die Fehlersuche selbst um einiges aufwendiger, da viele der üblichen Werkzeuge hier nicht zur Verfügung stehen. Die Funktionalität der erzeugten Treiber beschränkt sich daher auf die unbedingt notwendigen Elemente. Die Informationen zur Treiberprogrammierung entstammen hauptsächlich dem dazu notwendigen Microsoft Windows NT Device Driver Kit (DDK) [MS-96], CUSTER [Cus93], BAKER [Bak97] sowie VISCAROLO und MASON [VM99].

Um die Funktionalität der Hardwaretreiber allen beteiligten Rechnern zur Verfügung zu stellen, wird eine Zugriffschicht in Form von Windows NT Diensten verwendet. Sie arbeiten unabhängig von angemeldeten Benutzern und sind den bekannteren „Demons“ unter Unix vergleichbar. Die Dienste stellen Schnittstellen zur Verfügung, die den Zugriff auf die Hardwaretreiber kapseln. Dies hat mehrere Vorteile. Zum einen ist es auf dieser Ebene relativ leicht möglich, einen Zugriff über das Netzwerk vorzusehen. Die bereits erwähnten RPCs können hier einfach implementiert werden. Zum anderen ist mit Hilfe dieser Dienste die Forderung, den verhältnismäßig unkomfortablen direkten Zugriff auf die Treiber zu verhindern und durch einen bequemeren, leichter zu kontrollierenden Zugriff mittels der entsprechenden Dienste zu ersetzen, leicht erfüllbar. Die erhöhte Bequemlichkeit ist durch die wesentlich einfachere Programmierung und Wartung der Dienste, die sich im *User Mode* befinden, auch deutlich einfacher zu erreichen. Die Aufteilung in tatsächliche Kommunikation mit der Hardware durch die Kernel-Treiber und Bereitstellung von Zusatzfunktionalitäten durch die entsprechenden Dienste erreicht zudem eine gute funktionale Trennung der Elemente und eine Vereinfachung der Programmierung.

```
[ uuid(AB845E53-0F05-11d1-A06C-00A024B66774), version(1.0) ]
interface MeServoControl
{
 void MeServoBeginSession
  ([in] handle_t hBinding, [out] error_status_t* error);

 void MeServoEndSession
  ([in] handle_t hBinding, [out] error_status_t* error);

 short MeServoPing
  ([in] handle_t hBinding, [out] error_status_t* error);

 void MeServoSet
  ([in] handle_t hBinding, [out] error_status_t* error,
   [in, size_is(NumberOfValues)] short pWhich[],
   [in, size_is(NumberOfValues)] float Winkel[],
   [in] short NumberOfValues);

 void MeServoSetLegs
  ([in] handle_t hBinding, [out] error_status_t* error,
   [in] float Winkel[18]);

 short MeServoGetSignalPwm
  ([in] handle_t hBinding, [out] error_status_t* error);

 short MeServoSetDeltaPwm
  ([in] handle_t hBinding, [out] error_status_t* error, [in] short DeltaPwm);
}
```

Abbildung 24: Beispiel einer RPC Schnittstellenbeschreibung

3.3 Kommunikation mit den Diensten

Wie bereits angesprochen, kommunizieren die Dienste, die den Zugriff auf die Hardware kapseln, über Remote Procedure Calls mit den in der Hierarchie weiter oben liegenden Modulen. Für die RPCs wird ein genau bezeichnetes Interface beschrieben, in dem verbindlich festgelegt wird, welche Funktionen zur Verfügung stehen. Die Deklaration der Funktionen erfolgt in einer ähnlichen Form wie in der Programmiersprache C. Im Gegensatz zu C ist für die Deklarationen der IDL (Interface Definiton Language) die Größe der Datentypen und ihre Speicherung genau festgelegt [Fin94][RT93]. Aufgrund der beschreibenden Schnittstelle wird mit Hilfe eines Compilers C-Programmcode erzeugt, der dann mit dem restlichen Quellcode übersetzt und gebunden wird. Der maschinell erzeugte Code implementiert sowohl die prozeßübergreifende Kommunikation als auch die Verbindung über das Netzwerk und die dazu eventuell notwendige Ausrichtung der Datentypen. Als Kommunikationsprotokoll können unter anderem NetBEUI, IPX, UDP oder auch das bei TARRY in der Regel verwendete TCP/IP-Protokoll zum Einsatz kommen. Wird lediglich ein einzelner Windows NT Rechner zur Kontrolle eingesetzt, so kann das nur lokal verfügbare LRPC Protokoll als schnellste Alternative verwendet werden.

```
#include <iostream.h>
#include <fstream.h>

#include <neuralnet/ForwardNetwork.h>
#include <lapackpp/FloatVec.h>

#include <mechlib/PaceMaker.h>
#include <mechlib/Servo.h>

MePaceMaker PaceMaker(100);

int main(int argc, char** argv)
{
MeServo Servo
 ("tarry.mechanik.uni-duisburg.de");

if (!Servo.IsOk() ) {
     cout << "Keine Servo Verbind.";
     return 1;
     }

LaFloatVec InVec(5);
LaFloatVec ServoAngles(18);

NNForwardNetwork VL("VL.net");
NNForwardNetwork ML("ML.net");
NNForwardNetwork HL("HL.net");
NNForwardNetwork VR("VR.net");
NNForwardNetwork MR("MR.net");
NNForwardNetwork HR("HR.net");
NNForwardNetwork* Networks[6] =
    {&VL, &ML, &HL, &VR, &MR, &HR};

cout << "xDot, yDot, Drehung, Hoehe";
 cin >> InVec[0];
 cin >> InVec[1];
 cin >> InVec[2];
 cin >> InVec[3];

while (TRUE) {
 InVec[4] = PaceMaker.sin();
 InVec[5] = PaceMaker.cos();
 for (Leg=VL; Leg <= HR; Leg++) {
   Networks[Leg]->Propagate
     (InVec,
      ServoAngles(LaRange(Leg*3, 2+Leg*3)));
   }
  Servo.SetLegs(ServoAngles.data)
  Servo.GetSendSignal();
  PaceMaker++;
  }

return 0;
}
```

Abbildung 25: Beispiel eines einfachen Programms zur neuronalen Kontrolle von TARRY II

Die Darstellung der Gehmaschine durch die beschriebenen Dienste ist bezogen auf die im Rechner installierten Hardwarekarten. Um eine möglichst einfache Programmierung zu ermöglichen, ist es sinnvoll, Elemente wie Servomotoren, Fußtaster, Ultraschallsensoren etc. mit ihrer Funktionalität in C++-Klassen zu modellieren. Diese Einheiten verwalten dann die Zugriffe auf die notwendigen Dienste und implementieren Zusatzfunktionalitäten wie beispielsweise Zwischenspeicherung oder Umrechnung und Korrektur mit Hilfe von Kalibrierungsdaten. Diese Möglichkeit ist in Abbildung 23 durch die C++-Klassen auf dem lokalen und dem externen Rechner angedeutet.

Ein Beispiel hierfür ist der Dienst zur Ansteuerung der Servomotoren, dessen RPC-Schnittstellendefinition in Abbildung 24 dargestellt ist. Die Motoren selbst werden über pulsweitenmodulierte Signale angesteuert. Für den Bediener oder Programmierer sind die entsprechenden Werte wenig anschaulich und fehlerträchtig. Daher empfängt der Servo-Dienst die gewünschten Stellwinkel in Grad, rechnet sie mit Hilfe eine Kalibrierungstabelle um und sendet die ermittelten Pulsweiten an den Treiber. Dadurch ist es leicht möglich, die Koordinationssoftware durch Anpassung der Kalibrierungsdaten von Neuausrichtungen und Umprogrammierungen der Servomotoren unabhängig zu halten.

Durch diese Abstrahierung ist es möglich, ausgesprochen einfache Programme zur

Koordination zu erstellen. Ein Beispiel für eine einfache neuronale Steuerung findet sich in Abbildung 25.

Sollen externe Rechner dazu dienen, nicht nur Daten zu empfangen und zu visualisieren oder die entsprechenden Dienste direkt anzusprechen, so können problemlos zusätzliche RPC Schnittstellen definiert werden, die die Kommunikation zwischen den beteiligten Einheiten ermöglichen. Dadurch ist es auch möglich, die Ergebnisse mehrerer externer Rechner zusammenzufassen und zu bewerten und erst aufgrund dieser Ergebnisse Daten an die Dienste zu senden.