

Design der verteilten Echtzeit-Systemarchitektur DISTAL und Implementierung am Beispiel des humanoiden Roboters Myon

Diplomarbeit

Christian Thiele

Labor für Neurorobotik
Institut für Informatik
Humboldt-Universität zu Berlin

Betreuer: Dr. Manfred Hild

Gutachter: Prof. Dr. Hans-Dieter Burkhard
Prof. Dr. Holger Schlingloff

Berlin, den 13. Juni 2014

Zusammenfassung. Autonome mobile Roboter können von Jahr zu Jahr mehr leisten, ihre Entwicklung wird entsprechend anspruchsvoller. Mit der Verfügbarkeit kleiner, leistungsfähiger und trotzdem kostengünstiger Prozessoren ist es möglich, Aufgaben wie die Berechnung von Regelschleifen nicht mehr auf einem zentralen Prozessor zu bearbeiten, sondern diese über den Roboter zu verteilen. Da so jeder Rechenknoten eine dedizierte Aufgabe übernehmen kann, können Prozesse parallel ablaufen, wodurch die Komplexität für die einzelnen Rechenknoten verringert und somit die Entwicklung vereinfacht wird. Andererseits ist die Verteilung solcher Prozesse und die Kommunikation mehrerer Rechenknoten untereinander eine anspruchsvolle Aufgabe. In der vorliegenden Arbeit wird die *Distributed Architecture for Large Neural Networks* (DISTAL) vorgestellt, eine verteilte Echtzeit-Systemarchitektur, die ohne einen zentralen Rechenknoten auskommt. Sie ist ausgelegt für die Regelung mittels künstlicher neuronaler Netze, die mit der Desktop-Software *BrainDesigner* erstellt werden können. Dem *BrainDesigner* ist in dieser Arbeit ein eigenes Kapitel gewidmet. Beispielhaft wird die Implementierung der Architektur für den humanoiden Roboter Myon erläutert.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Entstehung der DISTAL-Architektur	6
1.2	Aufbau der Arbeit	8
1.3	Abgrenzung der Inhalte	9
1.4	Grundbegriffe	9
2	Der humanoide Roboter Myon	13
3	Systemarchitektur DISTAL	19
3.1	Bestehende Systemarchitekturen für Roboter	19
3.2	Allgemeine Anforderungen	24
3.3	Zusätzliche Anforderungen aus Anwendungsszenarien	25
3.4	Beschreibung der DISTAL-Architektur	27
3.4.1	Datenübertragung	27
3.4.2	SpinalCord-Timing	32
3.4.3	SpinalCord-Datenstruktur	34
3.4.4	Transparent Mode	36
3.4.5	Synchronisierung und Laufzeit-Metamorphose	38
3.5	Die DISTAL-Architektur am Beispiel des Roboters Myon	39
3.5.1	Überblick über die Teilnehmer beim Roboter Myon	40
3.5.2	Datenübertragung beim Roboter Myon	41
3.5.3	SpinalCord-Timing beim Roboter Myon	43
3.5.4	SpinalCord-Datenstruktur beim Roboter Myon	45
3.5.5	Transparent Mode beim Roboter Myon	47
3.6	Implementierung auf dem AccelBoard3D	47
3.6.1	Aufbau des AccelBoard3D	48
3.6.2	Speicherbereiche und Bootloader	50
3.6.3	AccelBoard3D-Systemsoftware	54
3.7	Implementierung auf dem BrainModule	57
3.7.1	Aufbau des BrainModules	57
3.7.2	Struktur der FPGA-Konfiguration	60

4 Software BrainDesigner	61
4.1 Erstellung von Regelschleifen mit der Software BrainDesigner	61
4.2 Plugins und Konfigurationen	75
4.3 Der neuronale Bytecode.....	80
4.4 Verwendung des BrainDesigners mit dem Roboter Myon	83
4.4.1 Kompilierung für ARM-Cortex-M3-Prozessoren	84
4.4.2 Maximale Netzgrößen beim AccelBoard3D	90
5 Fazit und Ausblick	95
5.1 Ausblick	96
Anhang.....	99
Anhang 1: SpinalCord-Datenfelder beim Roboter Myon	99
Anhang 2: Schaltplan AccelBoard3D.....	107
Anhang 3: Kommunikationsprotokoll Bootloader AccelBoard3D.....	109
Anhang 4: Befehle des neuronalen Bytecodes	111
Literatur	125
Danksagung	133

1 Einleitung

Kybernetik ist die Wissenschaft, die sich mit der Steuerung und Regelung in Maschinen, Lebewesen oder auch sozialen Einheiten beschäftigt. Die technische Kybernetik beschäftigt sich mit der Regelung von technischen Systemen. Teilbereiche der Kybernetik sind unter anderem Regelungstechnik, Energietechnik, Mechatronik und Biomechanik. Ein komplexer Roboter ist ein Beispiel für ein anspruchsvolles System, bei dessen Entwicklung alle genannten Bereiche relevant sind.

Die ersten Maschinen, die man heute im weitesten Sinne als Roboter bezeichnen kann, wurden erdacht, lange bevor die tschechischen Schriftsteller Josef und Karel Čapek Anfang der 1920er-Jahre erstmals diesen Begriff für künstliche Maschinen verwendeten. Bereits vor 2400 Jahren soll Archytas von Tarent eine mechanische Taube konstruiert haben, die von Ast zu Ast geflogen sein soll. Aufzeichnungen, wie diese Taube funktioniert haben soll, sind aber nicht überliefert.

Der erste detailliert beschriebene, programmierbare Automat – obgleich auch hier unklar, ob je gebaut – ist ein im 9. Jahrhundert in Bagdad von den Banū-Mūsā-Brüdern entwickelter Flöten- bzw. Orgelspieler (siehe Abbildung 1) [Koe01]. Auf einem sich drehenden Zylinder befinden sich Zähne, die über eine mechanische Verbindung Löcher einer Orgelpfeife öffnen oder schließen können.

Mit der Eroberung des östlichen Teils des Arabischen Reichs durch die Mongolen im 13. Jahrhundert geriet dieses Wissen wieder in Vergessenheit [Koe01]. Spätestens mit Beginn des 18. Jahrhunderts wurden auch in Europa ähnlich komplexe Maschinen gebaut. 1738 stellte Jacques de Vaucanson (1709–1782) einen automatischen Flötenspieler vor, der ähnlich programmiert werden konnte wie derjenige der Banū-Mūsā-Brüder neunhundert Jahre zuvor [Mor07, Koe01].

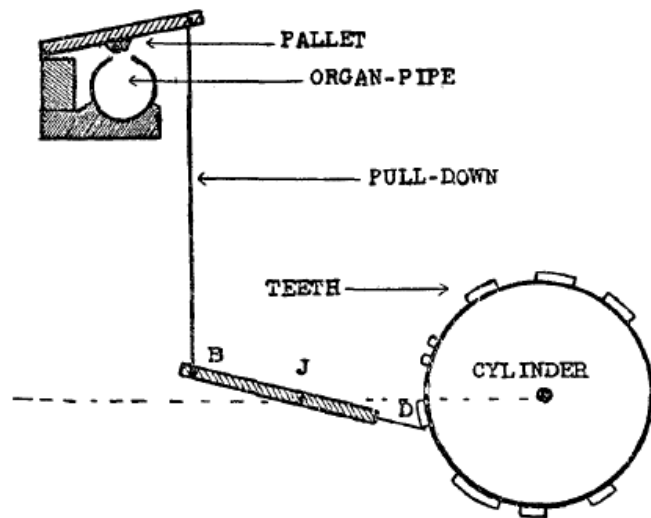


Abbildung 1: Eine Pfeife des automatischen Orgelspielers der Banū-Mūsā-Brüder. Auf dem sich drehenden Zylinder ist durch Zähne vorgegeben, ob das Loch geöffnet oder geschlossen ist (Abbildung einer Rekonstruktion aus [Far31]).

Vaucanson wurde 1741 zum Inspekteur der französischen Seidenindustrie ernannt und entwickelte in dieser Funktion in den folgenden Jahren den ersten vollautomatischen Webstuhl, der ebenfalls mit kodierten Zylindern „programmiert“ werden konnte. Dieser war jedoch kein Erfolg und geriet zunächst in Vergessenheit, bis die Einzelteile eines Exemplars mehr als fünfzig Jahre später im *Conservatoire des Arts et Métiers* in Paris von Joseph-Marie Jacquard (1752–1834) wiederentdeckt wurden [Koe01, Poa02]. Jacquard verbesserte die Konstruktion und stellte 1804 seinen vollautomatischen Webstuhl vor, dessen Muster sich durch Lochstreifen vorgeben ließen [Koh73] (siehe Abbildung 2).

Mit der industriellen Revolution wurden Automaten für die Industrie wie Jacquards Webstuhl immer wichtiger und auch heute noch werden Roboter größtenteils in der Industrie eingesetzt: weltweit sind derzeit 1,5 Millionen Industrieroboter installiert, allein 2012 wurden durch ihren Verkauf 26 Milliarden US-Dollar umgesetzt [IFR13].

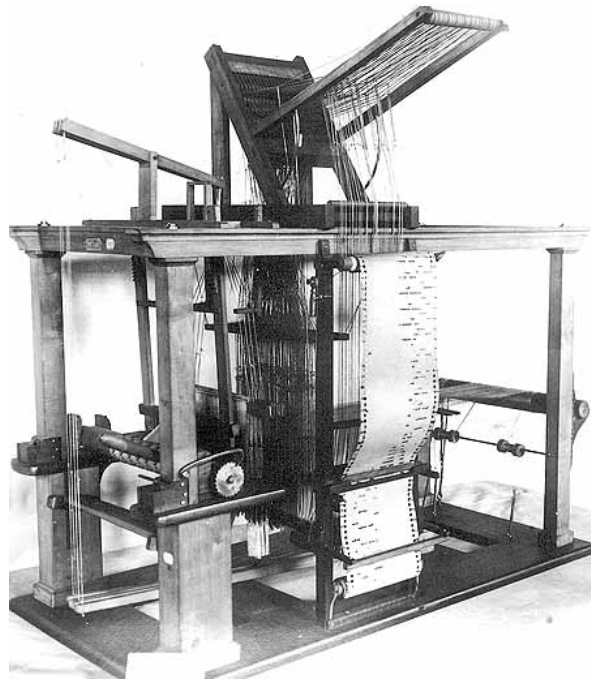


Abbildung 2: Mittels Lochstreifen programmierbarer Webstuhl von Joseph-Marie Jacquard, Anfang des 19. Jahrhunderts (Fotoquelle: The Computer History Museum, www.computerhistory.org).

Während Roboter heutzutage vorrangig funktionale Industriemaschinen sind, spielten in der Kunst immer wieder humanoide Roboter die Hauptrolle. Die Literatur war dabei der Realität Jahrhunderte voraus: Bereits aus dem 12. Jahrhundert sind Geschichten vom *Golem* überliefert, einer menschenähnlichen Figur aus Lehm, die mittels Magie zum Leben erweckt wird. Maschinen in Menschenform gibt es seit dem frühen 19. Jahrhundert in Science-Fiction-Geschichten wie E. T. A. Hoffmanns *Die Automate* von 1814. Fritz Langs Film *Metropolis* von 1927 mit dem Maschinenmenschen *Maria* als perfekter Kopie eines Menschen prägte das Bild von Humanoiden nachhaltig.

Die theoretischen Vorteile eines humanoiden Roboters liegen auf der Hand. Er könnte in der vom Menschen geschaffenen Umwelt agieren, ohne dass diese an ihn angepasst werden müsste. Ein radgetriebener Roboter bleibt vor Treppen stehen, wohingegen ein kleiner Krabbelroboter wegen fehlender Größe womöglich nicht einmal ein Fenster öffnen kann. Des Weiteren haben Untersuchungen

gezeigt, dass Roboter bei Menschen eine höhere Akzeptanz erreichen, wenn sie nach ihrem eigenen Bild gestaltet sind. Bei [Duf03] wird ein anthropomorphes Design für Mensch-Maschine-Schnittstellen gar als „unvermeidlich“ bezeichnet.

In der Realität werden humanoide Roboter bisher aber noch nicht in großen Stückzahlen gefertigt und größtenteils entweder als Spielzeug oder in der Forschung eingesetzt. Dabei existieren schon von Leonardo da Vinci Pläne für einen humanoiden Roboter und mit *Elektro* wurde bereits bei der Weltausstellung 1939 in New York ein Humanoider vorgestellt, der sogar Zigaretten rauchen konnte. Wieso ist die Forschung dann noch nicht weiter fortgeschritten? Die Schwierigkeit ergibt sich aus der Komplexität des Problems: Das biologische Vorbild Mensch ist über Jahrmillionen zu einem der komplexesten Systeme evolviert, das uns bekannt ist. Vieles dessen, was wir tagtäglich ohne Nachdenken leisten, ist tatsächlich eine Meisterleistung der Natur, allem voran der aufrechte Gang. Das bipedale Laufen ist heute noch eine der anspruchsvollsten Aufgaben bei der Entwicklung eines humanoiden Roboters, auch *Elektro* konnte nicht wirklich laufen. Er hatte kleine Räder unter den Füßen und seine Beine verließen nicht den Boden.

Erste einfache Laufmaschinen entstanden erst Ende der 1960er-Jahre vor allem in Japan. 1973 stellt Ichirō Katō *WABOT-1* vor, den ersten menschengroßen anthropomorphen Roboter, der laufen konnte [KOK+74, LT07]. In den 1980ern intensivierte sich die Forschung, 1986 stellte Honda seinen ersten bipedalen Roboter *E0* vor, der später zu einem vollständigen Humanoiden wurde und seither ständig weiterentwickelt wird. Seit dem Jahr 2000 trägt dieser Roboter den Namen *ASIMO*. Ausführliche Informationen zur historischen Entwicklung humanoider Roboter finden sich in [AS10] und [BES+13].

In den letzten Jahren hat sich die Zahl der Forschungsprojekte, die die Entwicklung humanoider Roboter zum Ziel haben, stark erhöht. Da ein Humanoider auf seine Umwelt reagieren muss, sind vielfältige Sensorik und reaktive Regelungsschleifen Grundvoraussetzung. Mit zunehmenden technischen Möglichkeiten rückt die Frage nach geeigneten Systemarchitekturen immer mehr in den Mittelpunkt. Ein autonomes, robustes System, das einfach programmiert werden kann, der Forschung zugleich aber alle Freiheiten lässt, kann dabei hilfreich sein, die Herausforderungen humanoider Robotik besser bewältigen zu können. Wenn die dahinterstehende Architektur und die Werkzeuge gleichzeitig für verschiedenste Roboter verwendet werden können, ist es für den einzelnen Forscher zudem möglich, seine Algorithmen auf all diesen Plattformen auszuprobieren.

Der Wunsch nach solch einem System war die Motivation zur Entwicklung der *Distributed Architecture for Large Neural Networks*: der DISTAL-Architektur.

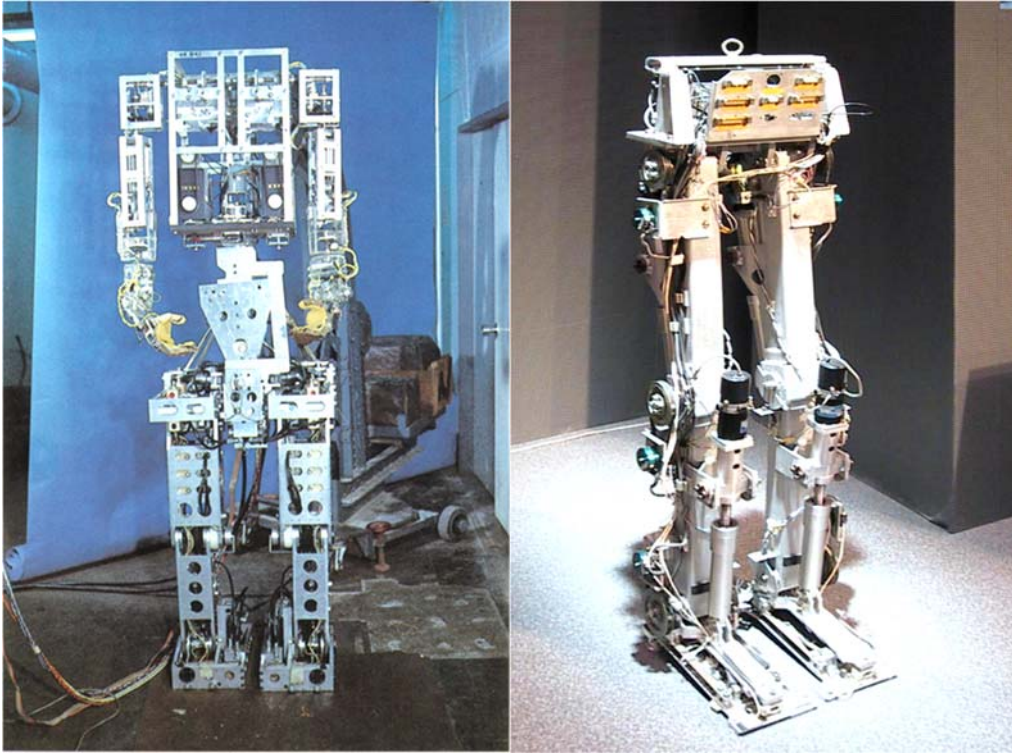


Abbildung 3: Frühe Laufmaschinen: links WABOT-1 (1973; Fotoquelle: waseda.ac.jp) und rechts Honda E0 (1986; Fotoquelle: plasticpals.com).

1.1 Entstehung der DISTAL-Architektur

Die Systemarchitektur ist in der Forschung an autonomen Systemen von zentraler Bedeutung. Aus ihren Konzepten und Eigenschaften leitet sich ab, wie die Hardware aufgebaut sein kann. Auch die Werkzeuge für die Arbeit mit dem System müssen auf die Architektur abgestimmt sein. Das HSW-Säulenmodell (siehe Abbildung 4) illustriert die Wichtigkeit der drei relevanten Teilaspekte Hardware, Systemarchitektur und Werkzeuge.

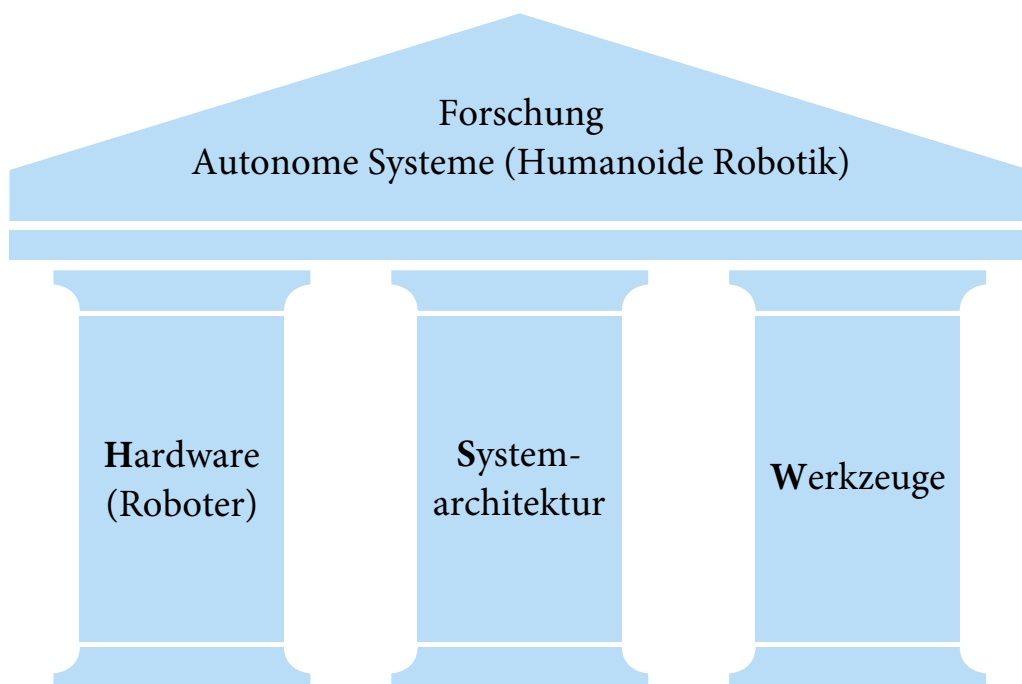


Abbildung 4: Das HSW-Säulenmodell: Die Robotik-Forschung steht auf den Säulen Hardware, Systemarchitektur und Werkzeuge.

Für Forschungsroboter gibt es spezielle Anforderungen [HSB+12]. Auf die folgenden wurde bei der Entwicklung der Systemarchitektur DISTAL besonders geachtet:

- **Verfügbarkeit:** Bei den meisten Forschergruppen existiert ein Engpass an Robotern. Umso kostengünstiger ein einzelner Roboter ist, desto mehr dieser Systeme kann sich eine Forschergruppe leisten. Ein modularer Roboter

ermöglicht es zusätzlich, Experimente schon mit einzelnen Modulen durchzuführen, wodurch sich mehrere Forscher einen Roboter teilen können. Auch für einen Gesamtroboter hat die Modularität Vorteile: Fällt ein Modul aus, muss während der Reparatur nur auf dieses Modul verzichtet werden, nicht auf einen kompletten Roboter.

- **Flexibilität:** Jedes Experiment soll ohne lange Vorbereitungen möglichst überall durchführbar sein, egal ob es sich um ein einfaches Experiment mit Einzelmodulen oder komplexe Aufbauten mit mehreren Robotern handelt. Experimente lassen sich schneller präsentieren, wenn auch Einzelmodule unabhängig von Steckdosen betrieben werden können. Neben der Modularität ist somit auch völlige Autonomie notwendig. Jedes Robotermodul muss für sich autonom sein, was ein verteiltes Energie- und Rechensystem voraussetzt.
- **Einfachheit:** Oftmals ist die Einarbeitung in ein System zeitintensiv und vor allem für Studierende, die in nur wenigen Monaten eine Abschlussarbeit schreiben sollen, fast unmöglich. Es ist also wichtig, dass möglichst schnell mit dem System gearbeitet werden kann. Außerdem sollte das System nach dem Einschalten praktisch sofort einsatzfähig sein. Programme müssen somit im nicht-flüchtigen Speicher des Roboters abgelegt werden.
- **Transparenz:** Forscher sollten jederzeit in der Lage sein, nachzuvollziehen, was im Roboter passiert. Alle Ergebnisse sollten reproduzierbar sein. Echtzeitfähigkeit und somit garantierte Ausführungszeiten sind dafür unabdingbar. Es sollte möglich sein, alle relevanten Daten während des Betriebs mitzuloggen.
- **Robustheit:** Beim Ausfall einer Komponente oder wenn ein Kabel bricht, sollte nicht das Gesamtsystem komplett ausfallen (graceful degradation). Bricht ein humanoider Roboter in sich zusammen, weil ein unwichtiger Sensor am Arm ausfällt, kann dies hohe, aber vermeidbare Kosten verursachen.

Beim Entwurf der DISTAL-Architektur (siehe Kapitel 3) wurden die Erfahrungen, die das Labor für Neurorobotik zuvor mit all diesen Aspekten an verschiedenen Robotern gesammelt hat, gebündelt, um eine Systemarchitektur zu entwickeln, die nicht nur für den zeitgleich entwickelten Roboter Myon (siehe Kapitel 2) geeignet ist, sondern auch auf verschiedenen anderen Robotik-Plattformen eingesetzt werden kann. Abbildung 5 zeigt den zeitlichen Verlauf der Entwicklung der verschiedenen Plattformen.

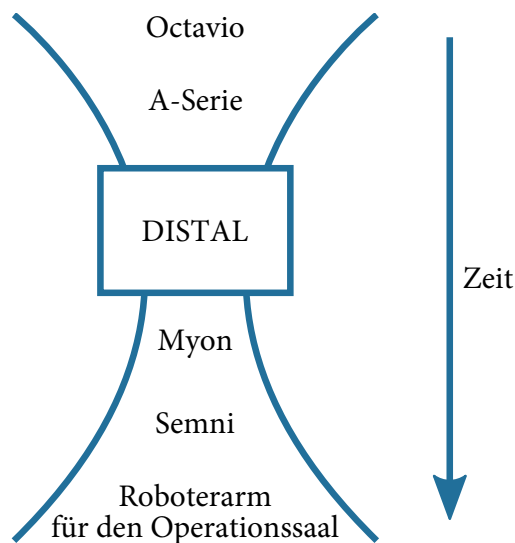


Abbildung 5: Die Entwicklung der DISTAL-Architektur ist von älteren Robotersystemen (Octavio [THS+12], A-Serie [HJS06, HMS07]) beeinflusst. Sie wurde während der Entwicklung des Roboters Myon konzipiert und wird heute auch im Roboter Semni [Hil13] und einem Roboterarm für den Operationssaal eingesetzt.

Neben der Hardware und der Systemarchitektur sind die Werkzeuge die dritte Säule der Robotik-Forschung. DISTAL ist auf die Verwendung großer neuronaler Netze ausgelegt. Somit ist es von Vorteil, diese ohne großen Aufwand erstellen zu können. Hierfür wurde als zentrales Werkzeug die Software BrainDesigner (siehe Kapitel 4) entwickelt.

1.2 Aufbau der Arbeit

Entsprechend der drei tragenden Säulen des HSW-Säulenmodells gliedert sich die Arbeit in drei Kapitel. In Kapitel 2 wird der humanoide Roboter Myon vorgestellt, an dem die im Rahmen dieser Arbeit entwickelte Systemarchitektur DISTAL später beispielhaft vorgestellt wird. Im darauffolgenden Kapitel 3 werden zunächst bestehende Roboterarchitekturen betrachtet und dann mittels Anwendungsszenarien Anforderungen an die DISTAL-Architektur formuliert.

Diese wird beschrieben und am Beispiel des Roboters Myon die Implementierung erläutert. Im Kapitel 4 wird die Software BrainDesigner vorgestellt, mittels derer große neuronale Netze hierarchisch erstellt werden können. Auch hier wird der Einsatz am Roboter Myon erläutert und vorgestellt, wie die erstellten Netze autonom auf dem Roboter ausgeführt werden können.

1.3 Abgrenzung der Inhalte

Die Entwicklung des Roboters Myon ist ein Prozess, an dem das gesamte Team des Labors für Neurorobotik der Humboldt-Universität zu Berlin beteiligt ist. Die in dieser Arbeit vorgestellte Systemarchitektur wurde von mir mit Unterstützung des Leiters des Labors, Dr. Manfred Hild, konzipiert. Die Implementierung der Systemarchitektur für den Roboter Myon sowie die Programmierung der in Kapitel 4 vorgestellten Software BrainDesigner wurden eigenständig von mir durchgeführt.

1.4 Grundbegriffe

In diesem Kapitel sollen einige Grundbegriffe erläutert werden, die zum Verständnis der Arbeit notwendig sind.

Systemarchitektur

Der Begriff System stammt vom griechischen σύστημα, was „das Verbundene“ bedeutet. Ein System ist eine Menge von Komponenten, die zusammen ein integriertes Ganzes ergeben. Ein Beispiel für ein System ist ein Roboter in seiner Gesamtheit von mechanischen Elementen bis hin zu Softwarekomponenten.

Eine Systemarchitektur besteht laut ISO 42010 aus „fundamentalen Konzepten oder Eigenschaften eines Systems in seiner Umwelt, ausgedrückt durch seine Bestandteile, Beziehungen und Design- und Entwicklungsprinzipien“¹.

¹ „fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution“ (ISO/IEC/IEEE 42010: Systems and software engineering – Architecture description)

Echtzeitfähigkeit

Ein mögliches Designprinzip einer Systemarchitektur ist die Echtzeitfähigkeit. Bei einem hart echtzeitfähigen System existieren feste Timing-Bedingungen, die nicht verletzt werden dürfen. Die Korrektheit des Systems bezieht sich also nicht ausschließlich auf das Ergebnis einer Berechnung sondern auch auf zeitliche Bedingungen.

Bei einem System, bei dem die Regelungsschleife einen festen Takt hat, muss sichergestellt sein, dass dieser Takt eingehalten wird. Die Sensorerfassung, die Berechnung der Ansteuerungswerte und die Motoransteuerung dürfen unter keinen Umständen länger dauern als die zur Verfügung stehende Zeit. Ein Echtzeitsystem ist für solch eine Anwendung also obligatorisch.

Sensomotorische Schleife

In der Neurologie wird die sehr vereinfachte Grundstruktur der Informationsverarbeitung des zentralen Nervensystems als sensomotorische Schleife bezeichnet [MN10]. Davon inspiriert wird dieser Begriff in der Robotik für eine Regelungsschleife verwendet, bei der der aktuelle Zustand des Systems in der Umwelt durch Sensoren erfasst wird und daraus neue Motorsignale generiert werden.

Abbildung 6 zeigt schematisch eine sensomotorische Schleife. Der aktuelle Umweltzustand wird mittels Sensoren erfasst. Im Anschluss berechnet ein Regelungsprogramm neue Werte, mit denen die Motoren angetrieben werden, was wiederum die Umwelt beeinflusst. Die Schleife ist also über die Umwelt geschlossen.

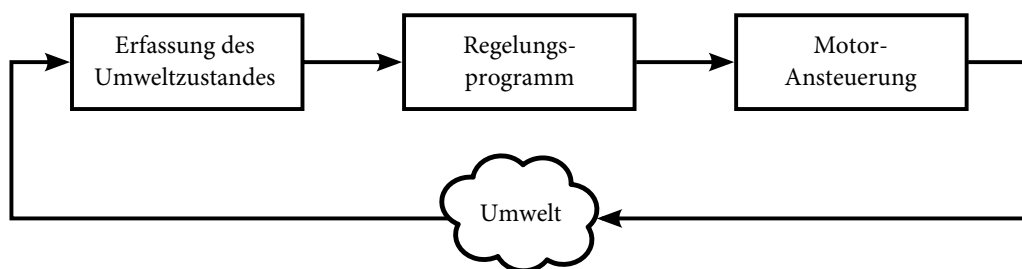


Abbildung 6: Schematische Darstellung einer sensomotorischen Schleife.

Bei einem Roboter soll beispielsweise an einem Gelenk ein bestimmter Winkel gehalten werden (Sollwert). Der aktuelle Winkel kann mittels geeigneter Sensorik gemessen werden (Istwert). Auf einen Motor wird daraufhin Spannung gegeben, die dafür sorgt, dass der Motor sich in Richtung des Zielwinkels dreht (Stellgröße). Ist dieser erreicht, muss zunächst keine Spannung mehr auf den Motor gegeben werden, es sei denn, durch äußere Einflüsse – wie zum Beispiel die Schwerkraft – verlässt das Gelenk wieder den Zielwinkel. Wird dies in einer ausreichend hohen Frequenz durchgeführt, hält das Gelenk den vorgegebenen Winkel.

Künstliche neuronale Netze

Eine Möglichkeit, Regelungsprogramme zu erstellen, ist die Nutzung künstlicher neuronaler Netze. Dabei handelt es sich um ein biologisch inspiriertes Rechenmodell, das auf miteinander verbundenen, künstlichen Neuronen beruht. Erstmals vorgestellt wurde das Modell eines solchen Neurons als allgemeine Grundlage mathematischer Berechnungen von Warren S. McCulloch und Walter Pitts im Jahr 1943 [MP43]. Eine ausführliche Darstellung künstlicher neuronaler Netze zur Bewegungssteuerung von Robotern findet sich in [Hil07]. In dieser Arbeit werden ausschließlich zeitdiskrete Modelle berücksichtigt, kontinuierliche Modelle enthalten numerisch aufwendige Integrationsprozesse und sind daher für die Robotik weniger interessant, vor allem wenn die Netze auf Mikroprozessoren berechnet werden sollen.

Ein universelles Neuronenmodell besitzt n Eingänge x_1, x_2, \dots, x_n und einen Ausgang y . Die Eingänge werden gewichtet zu einer internen Neuronenaktivität a aufaddiert. Mittels einer Abbildung f wird aus a das Ausgangssignal y für den nächsten Zeitschritt berechnet. f wird als Transferfunktion bezeichnet. Übliche Transferfunktionen sind Identität, Sprungfunktion, Sigmoidfunktionen und Tangens Hyperbolicus. Letztere wird bei den Beispielen dieser Arbeit als Standardtransferfunktion genutzt. Die folgende Formel fasst das Neuronenmodell mit Tangens Hyperbolicus als Transferfunktion zusammen:

$$y(t + 1) := \tanh \left(\sum_{i=1}^n w_i(t) x_i(t) \right), \quad w_i(t), x_i(t) \in \mathbb{R}, t \in \mathbb{N}$$

Grafisch dargestellt werden Neuronen üblicherweise als Kreise, die Eingänge als Pfeile zu diesem Kreis. Das Gewicht $w_i(t)$ wird an den Pfeil geschrieben (siehe Abbildung 7). Der Ausgang wird als Pfeil dargestellt, der vom Neuron wegführt.

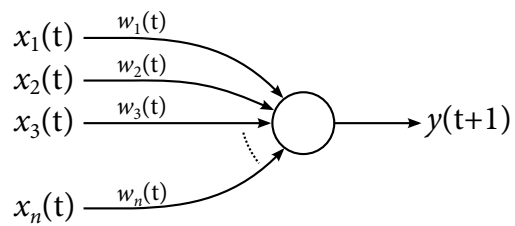


Abbildung 7: Darstellung eines Neurons mit n Eingängen und einem Ausgang.

Ein neuronales Netz entsteht durch die Verknüpfung mehrerer Neuronen. Ein solches Netz bildet einen gerichteten Graphen. Ein zyklensfreier Graph wird als Feed-Forward-Netz bezeichnet, ansonsten spricht man von einem rekurrenten Netz.

2 Der humanoide Roboter Myon

Der humanoide Roboter Myon wurde im Labor für Neurorobotik der Humboldt-Universität zu Berlin entwickelt. Er wurde vollständig von Mitarbeitern des Labors entworfen und umgesetzt. Von der Konstruktion über die Elektronik bis hin zu umfangreicher Software wurde dabei kaum etwas zugekauft. Die mechanischen Teile und sämtliche Platinen-Prototypen wurden in der laboreigenen Werkstatt hergestellt. Nur wenige Dinge wurden außer Haus gefertigt, wie die endgültigen Platinen und die von Bayer Material Science tiefgezogenen Außenschalen.

Der Roboter hat eine humanoide Form, deren Proportionen an die eines siebenjährigen Kindes angelehnt sind. In diesem Alter ist der Körperbau von Mädchen und Jungen noch sehr ähnlich, kurze Zeit später beginnen sich die Geschlechter stärker zu differenzieren. Somit ist auch die gesamte Gestalt des Roboters androgyn gehalten. Myon hat eine Körpergröße von 125 cm, und ist damit geringfügig kleiner als Siebenjährige Anfang des 21. Jahrhunderts in Deutschland (Mädchen: 126 cm, Jungen: 128 cm [SKB07]). Mit Außenschalen wiegt der Roboter 16 kg, somit ist er nur circa halb so schwer wie ein Kind gleicher Größe.

Myon ist als Forschungsroboter konzipiert. Ziel war es, eine Plattform zu schaffen, um biologisch inspirierte sensomotorische Schleifen zu erforschen. Experimente zum Laufen und stabilisierten Stehen sollten ebenso möglich sein, wie das Manipulieren der Umwelt durch einen Greifer. Wegen des Manipulators wurde Myon unter dem Arbeitstitel „M-Serie“ entwickelt. Konzepte und Designprinzipien sind in [HSB+11] und [HSB+12] ausführlich dargestellt.

Der Roboter ist modular aufgebaut. Dies gilt in zweierlei Hinsicht. Er besteht aus einem Torso, an den zwei Beine, zwei Arme und ein Kopf mittels eines speziell entwickelten Flanschs angesteckt werden können, der sowohl die Körperteile mechanisch fest verbindet als auch eine elektrische Verbindung herstellt. An jeden Arm kann über einen kleineren Flansch eine Hand angesteckt werden, die mit einem Motor sowohl greifen als auch zeigen kann.

Abbildung 8 zeigt ein Foto des verschalteten, vollständigen Roboters und ein computergenerierte Darstellung der einzelnen Module. Tabelle 1 listet die Körperteile, Massen, Freiheitsgrade und Anzahl der Aktuatoren auf.

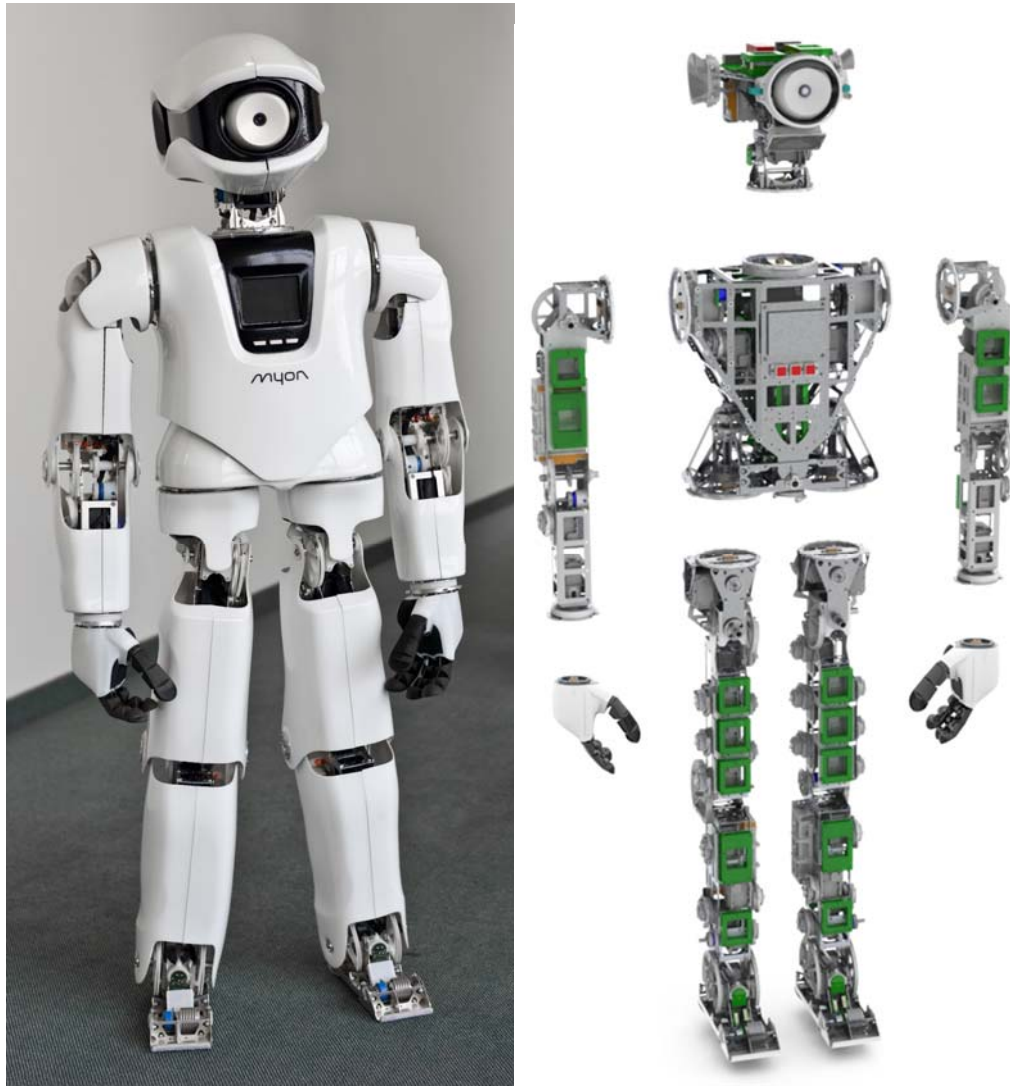


Abbildung 8: links: Roboter Myon mit Schalen (Fotografie); rechts: in einzelne Module zerlegt (computergenerierte Darstellung).

Körperteil	Masse	Gelenk	Freiheitsgrade	Anzahl Aktuatoren
Kopf	1,89 kg	Auge (mit Augenlidern)	4	4
		Hals	3	3
Arm (2×)	1,47 kg	Schulter	1	1
		Ellenbogen	1	1
		Handgelenk	1	1
Hand (2×)	0,24 kg	Finger	1	1
Torso	3,72 kg	Schulter (2×)	1	2
		Taille	1	1
		Hüfte (Drehung; 2×)	1	1
Bein (2×)	3,55 kg	Hüfte (vorne-hinten)	1	3
		Hüfte (links-rechts)	1	2
		Knie	1	3
		Sprunggelenk (vorne-hinten)	1	4
		Sprunggelenk (links-rechts)	1	1
		Zeh (passiv)	1	-
Gesamt	16,13 kg		32	48

Tabelle 1: Überblick über die Körperteile des Roboters Myon. Angegeben sind die Massen und für jedes Gelenk die Freiheitsgrade und Anzahl der Aktuatoren. Durch die Multiaktuierung einiger Gelenke ergibt sich eine höhere Anzahl Aktuatoren als Freiheitsgrade. Bei allen Gelenken außer dem Auge werden Aktuatoren vom Typ Dynamixel RX-28 von Robotis verwendet. Daten aus [HSB+11].

Der zweite Aspekt der Modularität ist, dass bei der Konstruktion versucht wurde, mit möglichst wenigen unterschiedlichen Modulen auszukommen. So wird fast überall der gleiche Motorentyp verwendet. Dort wo mehr Kraft benötigt wird, werden bis zu vier dieser Motoren zusammengeschaltet. An diesen Gelenken können auch antagonistische Antriebsverfahren erforscht werden. Auch bei der Elektronik wurde dieses modulare Prinzip beachtet. Die später in dieser Arbeit vorgestellte Platine AccelBoard3D (siehe Kapitel 3.6.1), auf der die zur Motoransteuerung nötigen Regelungsschleifen berechnet werden, findet sich 21 Mal im Roboter Myon.

Wie in Kapitel 1.1 erläutert, sollte nicht nur der gesamte Roboter vollständig autonom agieren können, auch jedes Körperteil² sollte autonom betrieben werden können. Nur die wechselbaren Hände können ausschließlich an einem Arm betrieben werden. Die Autonomie bezieht sich auf die erforderliche Rechenleistung ebenso wie auf die Stromversorgung. Deshalb befinden sich in jedem Körperteil Recheneinheiten, auf denen Regelungsprogramme fest abgelegt werden können. Die Bewegungssteuerung funktioniert so auch ohne andere Körperteile. In jedes Körperteil kann ein Akku eingesteckt werden, wobei auch hier – gemäß des Prinzips möglichst weniger Modultypen – überall das gleiche Akkumodul verwendet wird. Benötigt ein Körperteil weniger Leistung, kann Strom auch in andere Körperteile fließen. Ein Beispiel eines Experiments mit nur einem einzelnen Körperteil ist in [KBH11] beschrieben.

Im Roboter sind 44 Motoreinheiten vom Typ Dynamixel RX-28 vom Hersteller Robotis [Rob08] verbaut. Diese treiben 26 aktive Freiheitsgrade an – entsprechend des benötigten Drehmoments mit ein bis vier Motoren. Dazu kommen vier durch kleine Servomotoren betriebene Freiheitsgrade am Auge (zwei zur Bewegung des Auges, zwei zum Bewegen der Augenlider) sowie zwei passive Freiheitsgrade in den Zehen. Jede Motoreinheit enthält einen Prozessor, mit dem seriell über eine 1-Mbit-RS-485-Leitung kommuniziert wird. Der Prozessor steuert mittels Leistungstreibern einen DC-Getriebemotor an. Der Winkel wird innerhalb des Dynamixels in einem 300°-Bereich per Potentiometer bestimmt,

² Das Wort *Teil* wird im Deutschen in Bezug auf leblose Dinge im Neutrum verwendet, in Bezug auf lebende und abstrakte Dinge im Maskulinum. Das Wort *Körperteil* hat entsprechend in der deutschen Sprache eigentlich ein maskulines Genus. Da die Roboterteile leblos sind, wird in dieser Arbeit *Körperteil* als Neutrum verwendet: *Das Körperteil*.

zusätzlich wird die Temperatur am DC-Motor gemessen. Der Dynamixel-Prozessor kann lokal eine Positions- und Geschwindigkeitsregelung durchführen. Dies wird im Myon jedoch nicht genutzt, da die genaue Implementierung unbekannt ist. Zur Erforschung von Regelschleifen ist es jedoch notwendig, das Verhalten des Motors genauestens zu kennen.

Um bei den zu erforschenden sensomotorischen Schleifen möglichst viele Freiheiten zu haben, wurde vielfältige Sensorik verbaut: Winkelpositionen, Beschleunigungen, Spannungen, Ströme und Temperaturen werden an vielen Stellen gemessen, insgesamt werden 238 Sensorsignale in sechs Sensorqualitäten mit 100 Hz erfasst. Tabelle 2 listet diese auf. Im Kopf befinden sich zusätzlich zwei Mikrofone zum Stereo-Hören und eine Kamera. Sinnesähnliche Sensorik wurde an Stellen verbaut, an denen auch der Mensch die entsprechenden Wahrnehmungsorgane hat. Da nur eine Kamera verwendet wird, weicht der Myon-Kopf vom humanoiden ab und ist zyklisch. Auf den Großteil der Sensorik des Roboters Myon wird in [Ben10] ausführlich eingegangen.

Die vom Kölner Designbüro Frackenhohl Poulheim entworfenen Außenschalen haben in mehrerer Hinsicht wichtige Funktionen. Sie nehmen Momente und Kräfte (vorrangig Torsionskräfte) auf und entlasten damit das Endoskelett. Die Elektronik wird ebenso geschützt wie der Anwender davor sich zu verletzen. Nicht zuletzt erfüllen sie eine emotionale Funktion und senken die Hemmschwelle erheblich, mit dem Roboter zu interagieren.

Qualität	Position und Messmethode	Messbereich	Anzahl
Winkel	Innerhalb des Dynamixel RX-28 Potentiometer am Ruderhorn	300°	44
Winkel	An Gelenken Potentiometer (Vishay Spectrol Model 358) an ADC121C027	340°	22
Kraft	Fußkontakt Faraday-Kraftsensor von IB Hoch an Kapazitätssensor AD7150	0 – 100 N	8
Beschleunigung	Auf AccelBoard3D 3-Achs-Beschleunigungssensor MMA7455L	±8g	21×3
Strom	Auf AccelBoard3D in Motor- Stromzuführung 20mΩ-Strommesswiderstand an Strommessverstärker LMP8601 an ADC des Prozessors STM32	0,0 – 8,6 A	36
Spannung	Auf AccelBoard3D ADC des Prozessors STM32	0,0 – 26,5 V	21
Temperatur	Dynamixel RX-28 am DC-Motor	0 – 85 °C	44
Gesamt			238

Tabelle 2: Übersicht über die Sensorik des Roboters Myon. Insgesamt 238 unterschiedliche Sensorsignale werden im Takt der sensomotorischen Schleife erfasst. In der Tabelle nicht dargestellt sind zwei Mikrofone und eine Kamera, da diese nicht mit 100 Hz erfasst werden.

3 Systemarchitektur DISTAL

In den letzten Jahren wurden erstaunliche Erfolge im Bereich der Robotik erzielt: Fahrerlose Autos von Google haben bereits über eine Million Kilometer ohne Unfall zurückgelegt, ferngesteuerte Fahrzeuge – sogenannte Rover – erkunden autonom den Mars und während 1986 noch hunderttausende „Liquidatoren“ in den havarierten Block 4 des Atomkraftwerks Tschernobyl geschickt wurden, konnten viele ihrer Aufgaben nach der Nuklearkatastrophe 2011 im japanischen Kernkraftwerk Fukushima Daiichi von Robotern übernommen werden [NKO+13]. Bereits im Jahr 2000 wurde in [CS00] festgestellt, dass solche modernen Robotersysteme zu komplex sind, um mit konventionellen Programmier-techniken entwickelt zu werden. Als Lösung werden speziell auf die Bedürfnisse angepasste Systemarchitekturen vorgeschlagen.

In diesem Kapitel werden bestehende Robotersysteme analysiert und deren Architekturen untersucht. Im Anschluss werden Anforderungen formuliert und die DISTAL-Architektur vorgestellt. Das Kapitel schließt mit der Beschreibung der Implementierung für den Roboter Myon.

3.1 Bestehende Systemarchitekturen für Roboter

Der erste kommerziell eingesetzte Roboter *Unimate* wurde 1961 an einem General-Motors-Fließband in den USA installiert [BY08]. Bei frühen Industrierobotern wie *Unimate* handelte es sich um Systeme, die in sich abgeschlossen waren und eine einfache spezifische Aufgabe hatten. Ein zentraler Prozessor erfasste alle Sensoren und steuerte alle Motoren, dazu wurden intern proprietäre Kommunikationssysteme genutzt [Neu07]. Ab Mitte der 1980er-Jahre nahm die Anzahl an Robotern in großen Industrie-Fertigungsstraßen stark zu und es wurden Feldbussysteme eingeführt. Diese wurden nach und nach standardisiert, die

wichtigste Norm ist dabei die IEC 61158 von 1999³. Einen Überblick über diese Feldbusse liefert [Neu07]. In solchen großen Systemen kommunizieren hunderte Teilnehmer, meist gesteuert von einer zentralen Instanz.

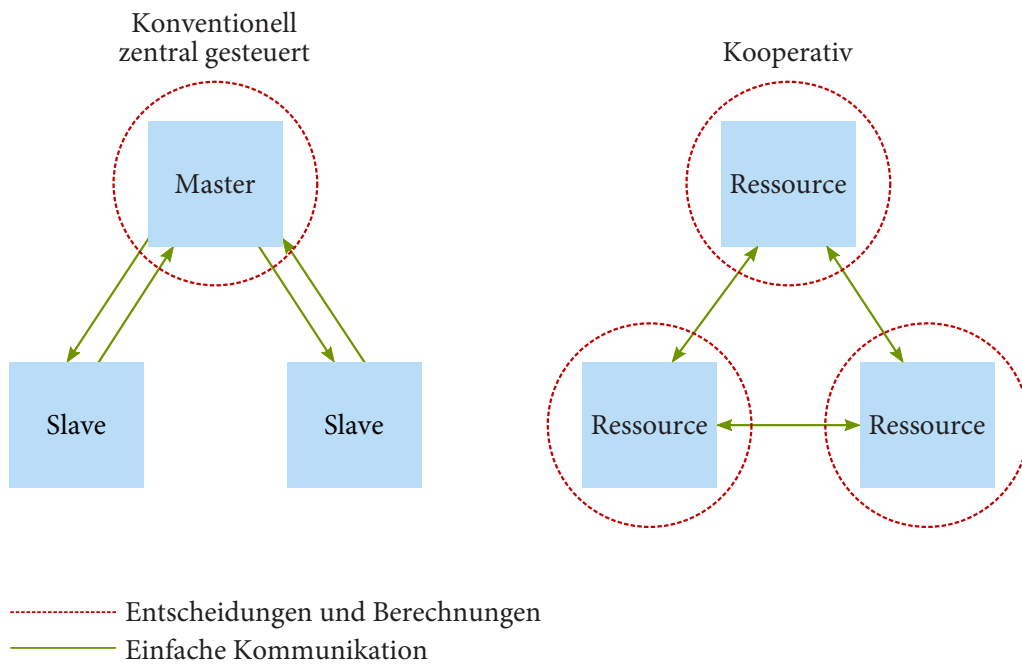


Abbildung 9: Vergleich von konventionellen, zentral gesteuerten Industriesystemen (links) zu dezentralen, kooperativen Systemen (rechts). Abbildung einer Grafik aus [MM05] nachempfunden.

Im Rahmen von „Industrie 4.0“, einem Projekt in der Hightech-Strategie der Bundesregierung, wird seit 2012 Forschung an intelligenten, autonomen Systemen unterstützt (BMW-Programm „Autonomik in Industrie 4.0“). Die zunehmende Komplexität großer Fertigungssysteme in der Industrie soll dabei reduziert werden, indem kleinere, autonome Systeme einzelne Aufgaben übernehmen und miteinander über „intelligente Netze“ kommunizieren. Es ist die Rede

³ IEC 61158: *Digital data communication for measurement and control – Fieldbus for use in industrial control systems*. Ergänzt wird dieser Standard durch IEC 61784 (*Industrial communication networks – Profiles*), in dem sogenannte Communication Profile Families (CPF) definiert werden.

vom „Paradigmenwechsel von einer zentralen zu einer dezentralen, erweiterten (augmentierten) Steuerung“ [HKK14]. Abbildung 9 zeigt die Unterschiede schematisch. In der Informatik werden kollaborative Agentensysteme als Teil der Künstlichen Intelligenz bereits seit Jahrzehnten unter dem Stichwort Multiagentensysteme untersucht.

Bei humanoiden Robotern handelt es sich im Vergleich zu großen Industriestraßen um kleine, in sich geschlossene Systeme. Häufig wird, wie bei frühen Industrierobotern, mit einem zentralen Prozessor gearbeitet. Laut [Pfe05] arbeiteten 2005 nur 8,3 % der untersuchten zweibeinigen Laufmaschinen mit einem dezentralen Rechnersystem. Damit ist ein System gemeint, das zwar mehrere Prozessoren umfasst, aber auch immer einen zentralen Hauptrechner beinhaltet. Der Trend zu autonomen Subsystemen ist aber auch bei humanoiden Robotern erkennbar.

Beispielhaft lässt sich die Entwicklung an den Robotern der Technischen Universität München beobachten. Erste sechs- und achtbeinige Roboter (genannt *MAX* und *MORITZ*) besaßen mehrere Platinen mit Mikroprozessoren, die die Sensordatenerfassung und Low-Level-Regelung übernahmen. Diese mussten jedoch mit parallelen CAN-Bussen an einen externen PC angeschlossen werden, der die Planung und Steuerung übernahm.

1997 wurde an der TU München mit der Entwicklung von *JOHNNIE* begonnen, einem 1,80 m großen humanoiden Roboter [LLG+04, Pfe05]. Da *JOHNNIE* autonom sein sollte, wurde ein PC-System konzipiert, das in den Roboter integriert werden konnte. Mikroprozessoren wurden weiterhin für die Sensordatenerfassung und Low-Level-Regelung eingesetzt, jedoch nun per PCI-Erweiterungskarte direkt in den PC integriert, der mit einem Echtzeit-Linux betrieben wurde.

Bei der Entwicklung von *LOLA*, dem Nachfolger von *JOHNNIE*, wurde ein dezentraleres System gewählt, um es modularer und leichter erweiterbar zu halten [UBL06]. Die Low-Level-Regelung wurde wieder mit speziell entwickelten Platinen lokal und verteilt durchgeführt. Die Gesamtsteuerung lag weiterhin bei einem Standard-PC. Zur Kommunikation wurde ein Feldbus aus der Industrierobotik gewählt, das auf Ethernet basierende SERCOS III.

Erkennbar ist hier eine oft zu findende Aufteilung. Einfache Aufgaben wie die Steuerung eines Motors oder die Sensorvorverarbeitung werden getrennt von der Gesamtsteuerung des Systems beziehungsweise höheren kognitiven Prozessen (beispielsweise Bildverarbeitung oder Spracherkennung) gehandhabt. Wenn

nur ein Prozessor zur Verfügung steht, werden diese auf verschiedene Threads oder Prozesse verteilt [STH10]. Weitere Beispiele für diese Trennung sind das Saphira/Aria-System [Kon02] (siehe Abbildung 10) und der humanoide Roboter HRP-3. Bei Letzterem übernimmt ein zentrales System die Steuerung der Arme und des Kopfes, wohingegen der Torso, die Hände und die Beine von einem verteilten System gesteuert werden [KHK+08]. Zum Teil sind höhere Verarbeitungsstufen auch weiterhin extern, beispielsweise beim Roboter iCub [MSV+08].

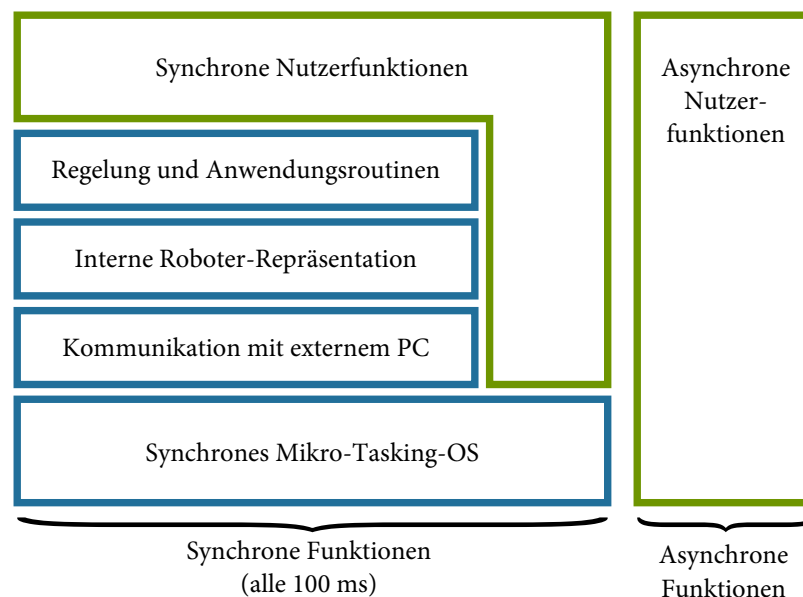


Abbildung 10: Beispiel für eine Systemarchitektur mit verschiedenen Prozessen: Das Saphira/Aria-System [Kon02]. Blau sind Funktionen des Systems dargestellt, grün Nutzerfunktionen. Das System führt in einem 10-Hz-Takt synchrone – mitgelieferte oder selbst erstellte – Funktionen aus, beispielsweise zur Motorsteuerung. Selbst erstellte Funktionen können zusätzlich asynchron laufen, dies sind normalerweise rechenintensivere Prozesse zur Planung oder Bildverarbeitung.

Noch mehr Stufen hat das Konzept von *ARMAR-III*, einem humanoiden Roboteroberkörper auf einer fahrenden Plattform [ARA+06]. Hier werden an einen herkömmlichen PC, der als Master fungiert, vier weitere PCs angeschlossen, die

unterschiedliche Aufgaben übernehmen, unter anderem ein PC für die Bildverarbeitung und einer für Audio-Synthese und -Erkennung. An einen weiteren PC sind zwölf Steuerungsplatinen (hier „Universal Controller Module“ genannt) angeschlossen, die jeweils einen DSP und ein FPGA enthalten und die Low-Level-Regelung und Sensorverarbeitung übernehmen.

Auf allen PCs in den genannten Robotiksystemen wird Linux als Betriebssystem eingesetzt. Häufig wird auf eine echtzeitfähige Variante wie RTLinux oder ART-Linux zurückgegriffen. Selten werden auch andere Betriebssysteme genutzt, beispielsweise Windows XP bei *KHR-2* des südkoreanischen Korea Advanced Institute of Science and Technology (KAIST) [KPO04] oder das Echtzeitbetriebssystem QNX bei *Wabian-2* der japanischen Waseda-Universität [OAS+06]. Selten werden Eigenentwicklungen eingesetzt, beispielsweise das inzwischen eingestellte AperiOS von Sony. Für Embedded-Prozessoren gibt es eine Vielzahl von einfachen Betriebssystemen, beispielsweise TinyOS, Contiki, RIOT-OS oder Mantis OS.

Die auf dem Betriebssystem aufsetzende Software ist entweder eine vollständige Eigenentwicklung oder nutzt als Grundlage ein Software-Framework wie beispielsweise MCA2 [UZ07]. Dabei handelt es sich um ein modulares, echtzeitfähiges C/C++-Framework, das dediziert zur Entwicklung von Robotersteuerungen entworfen wurde. Einige Software-Frameworks für humanoide Roboter werden in [Reg10] und [OC03] vorgestellt und verglichen. Erwähnenswert sind zusätzlich Urbi [Bai05] und das Robot Operating System (ROS) [QGC+09]. Letzteres ist entgegen dem Namen kein Betriebssystem im klassischen Sinne, sondern ebenfalls ein Software-Framework für Robotiksysteme. Kommerzielle Roboter bringen meist ihr eigenes Framework mit (z. B. NaoQi beim Roboter Nao von Aldebaran Robotics). Zum Teil lassen sich verschiedene Systeme kombinieren, beispielsweise ROS mit Urbi, oder setzen aufeinander auf. Auch diese Frameworks basieren auf dem Vorhandensein eines zentralen Prozessors, so gibt beispielsweise Urbi immer eine einzelne Anwendungsdatei aus.

Bei der Entwicklung der Laufmaschine *Octavio* [THS+12] wurde ein anderer Weg gewählt. Es gibt keinen zentralen Prozessor. Somit ist es möglich, die Roboterkonfiguration zu ändern. Beispielsweise kann Octavio als 4-, 6- oder auch 8-Beiner betrieben werden. Mit Octavio wurde gezeigt, dass auch komplexe neuronale Netze mit über 100 Neuronen pro Bein auf 16-Bit-Prozessoren berechnet werden können.

Am Labor für Neurorobotik wurde bereits vor *Myon* ein humanoider Roboter entwickelt, die *A-Serie* [STH10]. Sie ist 42 cm groß, wiegt knapp über zwei Kilogramm und besitzt 21 Freiheitsgrade [HJS06, HMS07]. Der Roboter ist vollständig autonom. Die höhere Steuerung und aufwendige kognitive Prozesse wie die Bildverarbeitung erfolgen auf einem PDA mit Windows Mobile als Betriebssystem. Die sensomotorische Regelung wird auf acht Rechnerknoten verteilt. Ein Master-Board ist für den Betrieb nötig, es gibt den Kommunikationstakt vor und kommuniziert mit dem PDA. Die Steuerung ist in [Thi07] beschrieben.

Bei allen untersuchten Architekturen gibt es mindestens eine Rechenkomponente, ohne die das Gesamtsystem nicht funktioniert. Dies führte zu der Entscheidung, eine neue Systemarchitektur zu entwickeln, die vollständig dezentral arbeitet: Die DISTAL-Architektur.

3.2 Allgemeine Anforderungen

Eine der Hauptanforderungen war, dass keine zentrale Recheneinheit nötig ist. Die Berechnungen werden über den Körper verteilt von verschiedenen Knoten durchgeführt. Jede Recheneinheit soll dabei für sich autonom funktionieren, auch wenn alle anderen Einheiten fehlen oder während des Betriebs ausfallen. Dabei gibt es natürliche, nicht zu vermeidende Einschränkungen. Ist beispielsweise der Kopf eines humanoiden Roboters nicht vorhanden, kann ein Arm noch immer eine Zeigebewegung ausführen. Da der Roboter aber ein Objekt, auf das gezeigt werden soll, nicht mehr sehen kann, fehlen der Zeigebewegung notwendige Eingabewerte. Ein stabilisiertes Stehen kann hingegen auch ohne Kopf problemlos weiter funktionieren. Ein vollständig verteiltes System hat also den Vorteil, eine hohe Ausfallsicherheit zu haben, ohne dass auf eine zentrale Instanz für auditive oder visuelle Wahrnehmung verzichtet werden muss.

Da es sich um ein verteiltes System handelt, muss jeder Teilnehmer für sich Regelungswerte berechnen und lokal angeschlossene Motoren betreiben. Über einen gemeinsamen Bus sollen zwischen den Knoten Daten ausgetauscht werden, womit ein Gesamtsystem entsteht. Übertragen werden neben Sensorwerten auch einzelne Aktivierungen neuronaler Netze. Die Sensorwerte und auch die angesteuerten Motorspannungen sollen in jedem Zeitschritt übermittelt werden. Jedem Teilnehmer stehen daher jederzeit alle Sensordaten des Systems zur Verfügung. Der Fokus bei der Datenübertragung zwischen den Teilnehmern liegt auf

diesen Sensordaten. Neuronale Netze werden lokal berechnet, die Übertragung von Zwischenergebnissen in Form von Aktivierungen einzelner Neuronen ist möglich, soll aber auf wenige Neuronen beschränkt bleiben.

Die Übertragung aller Sensordaten in jedem Zeitschritt hat eine entscheidende Implikation für das Gesamtsystem. Gibt es während der Übermittlung der Daten Übertragungsfehler ist eine Neuansforderung dieser Daten überflüssig – schon kurze Zeit später werden ohnehin aktualisierte Daten übertragen. Das Protokoll kann entsprechend einfach gestaltet werden.

Die sensomotorischen Schleifen sollen in einem festen Zeitraster berechnet werden. Die Übertragung der Sensorwerte muss entsprechend auch einem festen Timing unterliegen. Konsequenterweise ist das gesamte System ein Echtzeitsystem. Bei der Planung für ein konkretes Robotersystem werden feste Zeiten vorgegeben, die während der Implementierung immer wieder mithilfe eines Oszilloskops überprüft werden. Das System soll aber robust gegen kleinere Abweichungen sein und Mechanismen haben, um mit falschem Timing anderer Teilnehmer umzugehen.

Die geforderte Modularität macht es notwendig, in jedem Robotermodul Recheneinheiten zu haben. Einfache Knoten mit kostengünstigen Mikroprozessoren ermöglichen es, je nach Komplexität eines Moduls unterschiedlich viele Rechenknoten in diesem einzusetzen. Die Vielzahl der Rechenknoten erhöht die Wahrscheinlichkeit, dass beim Ausfall eines Knotens das Gesamtsystem eingeschränkt weiterarbeiten kann (graceful degradation). Der Einsatz eines komplexen Betriebssystems, wie bei vielen Robotik-Architekturen üblich, ist dabei nicht möglich. Dies muss jedoch kein Nachteil sein: Middleware mit Hardwareabstraktionsschichten erleichtert zwar die Arbeit in der Servicerobotik, für die Erforschung sensomotorischer Schleifen kann es aber vorteilhaft sein, zugunsten der Reaktivität auf diese zu verzichten.

3.3 Zusätzliche Anforderungen aus Anwendungsszenarien

Die DISTAL-Architektur ist für die Bewegungssteuerung mobiler Roboter konzipiert. Bei der Planung der Architektur wurden zunächst, anhand der im Labor für Neurorobotik vorhandenen Erfahrungen beim Umgang mit Robotern, zwei

maßgebliche Anwendungsszenarien definiert, die auch in Abbildung 11 dargestellt sind.

- Im *Transparent Mode* soll es möglich sein, den Roboter an einen externen PC anzuschließen und von dort zu steuern. Alle Sensordaten sollen dem PC zur Verfügung stehen, der daraus Ansteuerungswerte für die Motoren berechnet. Dieser Modus hat den Vorteil, dass Parameter der Berechnung oder auch die gesamte Berechnung am PC per Mausklick angepasst werden können – er bietet also maximale Flexibilität. Die Ansteuerung von festen, vorberechneten Datenreihen ist auf diese Weise ebenfalls einfach möglich. Problematisch ist die Tatsache, dass PCs im Allgemeinen keine Echtzeitsysteme sind, dieser Modus ist daher weniger zuverlässig.
- Im *Deployment Mode* wird der Code zur Motoransteuerung im nicht-flüchtigen Speicher auf dem Roboter abgelegt. In diesem Fall kann der Roboter vollkommen autonom betrieben werden, Experimente werden nicht durch Kabel behindert. Die Echtzeitanforderungen können eingehalten werden, Änderungen an der Berechnung sind aber nicht so flexibel möglich.

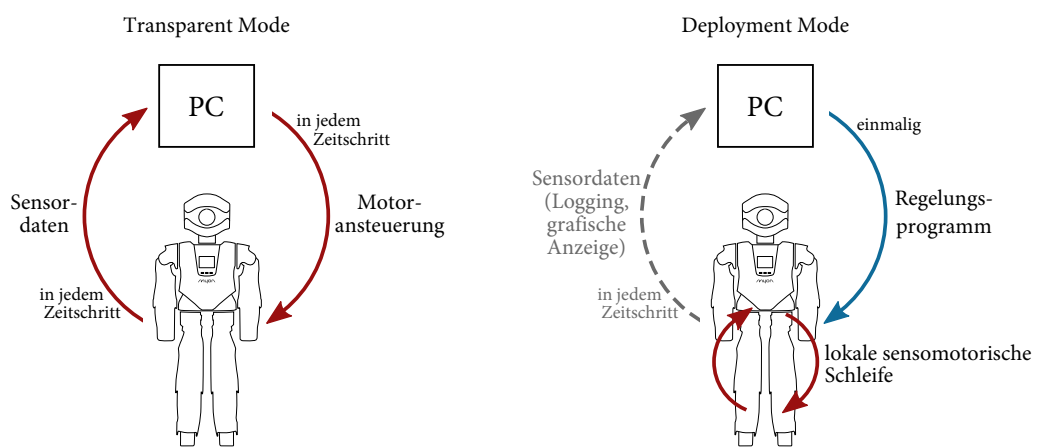


Abbildung 11: Transparent Mode und Deployment Mode im Vergleich. Beim Transparent Mode läuft die sensomotorische Schleife über den PC, beim Deployment Mode wird ein Regelungsprogramm auf den Roboter gespielt und dort ausgeführt. Alle Daten können auch im Deployment Mode vom PC mitgeloggt oder grafisch angezeigt werden.

Bei der Arbeit mit dem Roboter soll dabei möglichst einfach zwischen beiden Modi gewechselt werden können.

Integraler Bestandteil der Architektur soll eine Desktop-Software sein, mit der komplexe neuronale Netze einfach erstellt werden können. Ein Roboter soll direkt an den PC angeschlossen werden und aus der Software heraus im Transparent Mode betrieben werden können. Die gleichen Netze sollen aber auch ohne Umweg per „Deployment“ fest auf den Roboter gespielt werden können, sodass ein autonomer Betrieb des Roboters möglich ist.

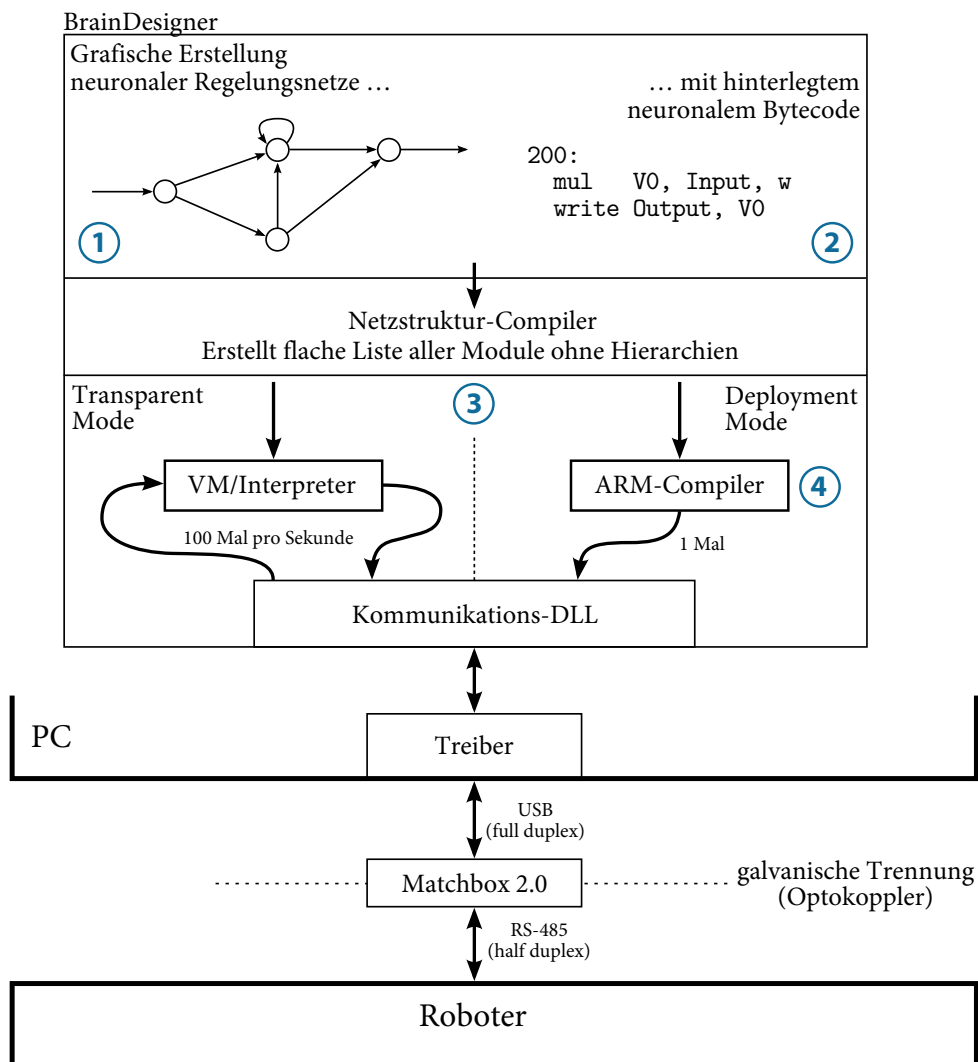
3.4 Beschreibung der DISTAL-Architektur

In einem DISTAL-System kommunizieren mehrere Teilnehmer an einem gemeinsamen Bus in einem festen Zeitraster miteinander und tauschen ihre Sensordaten im Takt der sensomotorischen Schleife aus. Dieser physikalische Bus wird SpinalCord-Bus genannt, das Protokoll ist das SpinalCord-Protokoll⁴. Die einzelnen Teilnehmer können unterschiedlich sein, von kleinen Embedded-Prozessoren über FPGAs bis hin zu externen PCs. Die Abbildungen 12 und 13 zeigen eine Gesamtübersicht der DISTAL-Architektur für den Roboter Myon.

3.4.1 Datenübertragung

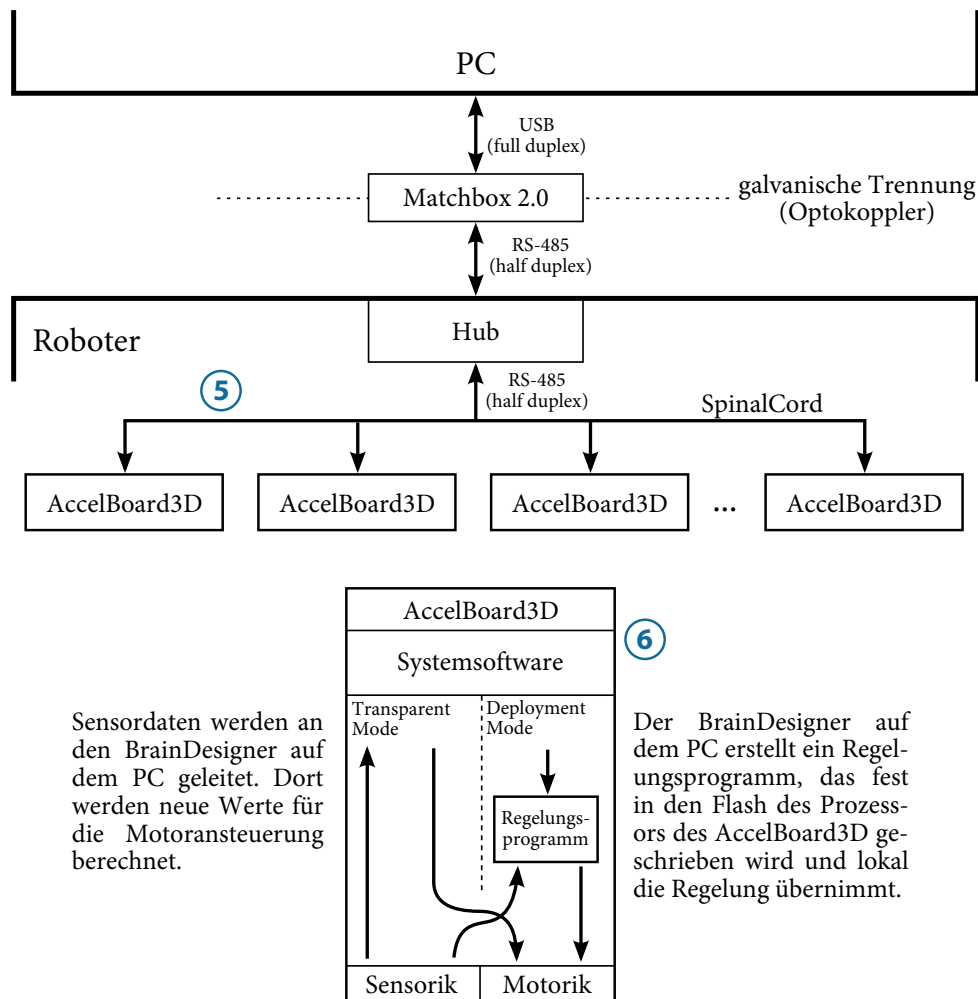
Für die Implementierung der DISTAL-Architektur auf einem realen System muss man sich für eine Art der Datenübertragung entscheiden. Je nach Übertragungsart werden dabei unterschiedliche Schichten des OSI-Modells durch die entsprechenden Standards vorgegeben. Üblicherweise sind dabei mindestens die Schichten 1 (Bitübertragungsschicht; Physical Layer) und 2 (Sicherungsschicht; Data Link Layer) definiert. Die verbreitetsten Systeme basieren auf Ethernet, CAN oder RS-485 in Kombination mit UART [Neu07]. [PMV+06] beschreibt erste Versuche, Feldbusse basierend auf Funk (WLAN bzw. Bluetooth) zu entwickeln, kommt aber zu dem Schluss, dass dies derzeit noch nicht zuverlässig einsetzbar ist.

⁴ Spinal cord ist der englische Begriff für das Rückenmark. Wegen der Analogie zur Signalübertragung von Neuronen wurde dieser Begriff gewählt.



- ① Bedienung BrainDesigner: Kapitel 4.1
- ② Neuronaler Bytecode: Kapitel 4.3
- ③ Nutzung verschiedener Konfigurationen: Kapitel 4.2
- ④ Deployment für ARM-Prozessoren: Kapitel 4.4

Abbildung 12: Gesamtübersicht der DISTAL-Architektur (Teil 1). Auf dem PC werden in der Software BrainDesigner grafisch neuronale Netze erstellt, denen neuronaler Bytecode hinterlegt ist. Diese werden entweder bei angeschlossenem Roboter auf dem PC berechnet oder für die Accel-Board3D-Platinen kompiliert.



⑤ Kommunikation zwischen Teilnehmern: Kapitel 3.4/3.5

⑥ AccelBoard3D/Systemsoftware: Kapitel 3.6

Abbildung 13: Gesamtübersicht der DISTAL-Architektur (Teil 2). Alle Teilnehmer kommunizieren untereinander über den gemeinsamen Bus (SpinalCord). Je nachdem ob der Roboter im Transparent Mode oder im Deployment Mode betrieben wird, wird die sensomotorische Schleife entweder vom PC berechnet oder von einem im Flash des AccelBoard3D-Prozessors vorhandenen Regelungsprogramm.

Ethernet (IEEE 802.3) definiert die OSI-Schichten 1 und 2 und wird in der Industrieautomation häufig genutzt. Ethernet ist auch die Grundlage verschiedener Echtzeit-Aufsätze wie SERCOS III, das beispielsweise beim Roboter *LOLA* der Technischen Universität München eingesetzt wird [UBL06]. SERCOS-III benötigt jedoch einen dedizierten Master und kann nicht in einer Stern-Topologie verwendet werden, da die Echtzeitfähigkeit durch Netzwerk-Switches verloren geht [Fel05]. Weitere Ethernet-Echtzeitaufsätze wie EtherCAT oder PROFINET haben dieselben Probleme und benötigen für die Slaves zusätzlich sogar spezifische ASICs [Pry08]. Für mobile Robotik ist Ethernet jedoch auch prinzipiell schlecht geeignet. Die hohe Datenübertragungsrate (100 Mbit) erfordert stark geschirmte Kabel, die entsprechend unflexibel sind. Ethernet-PHY-Chips wie der DP83848 oder der KSZ8051 sind alleine so groß wie kleine Embedded-Prozessoren und wenn man die vom Standard vorgegebenen RJ-45-Stecker verwendet, nehmen entsprechende Buchsen viel Bauraum ein. Schwere Kabel, große Platinen und volumenreiche Stecker und Buchsen sind für mobile Roboter jedoch ungeeignet.

Auch bei der Entwicklung des humanoiden Roboters HRP-3 wurde zunächst beim Prototypen ein Ethernet-basiertes System verwendet [AKK+05], dann jedoch auf ein CAN-basiertes System gewechselt, um die „Zuverlässigkeit und Wartbarkeit des Systems zu verbessern“ [KHK+08]. CAN (Controller Area Network) wird auch in verschiedenen anderen Robotern verwendet, beispielsweise [Dil05, KPO04, MSV+08]. CAN ist in der ISO 11898 standardisiert und definiert ebenfalls die OSI-Schichten 1 und 2. Für Echtzeitsysteme werden wiederum entsprechende Aufsätze in höheren OSI-Schichten benötigt, beispielsweise TTCAN [LH02]. Treiberchips für die physikalische Schicht sind dabei klein, beispielsweise ist der SN65HVD230 im 8-Pin-SOIC-Gehäuse erhältlich. Auch wenn häufig recht große 9-Pol-Sub-D-Stecker verwendet werden, gibt es für die Steckverbinder keine verbindlichen Vorgaben. Die Signalpegel sind ausgehend von einer 5-Volt-Versorgungsspannung definiert, wobei es auch 3,3-Volt-CAN-Transceiver gibt, die problemlos zusammen mit 5-Volt-Transceivern an einem Bus betrieben werden können [BM13]. Eine niedrigere Spannung ist in mobilen Systemen aufgrund der geringeren Leistungsaufnahme vorzuziehen. Die maximale Datenübertragungsrate ist von der Leitungslänge abhängig, wobei gemäß ISO 11898-2 (High-Speed CAN) bei Leitungen unter 40 Metern eine maximale Übertragungsrate von 1 Mbit/s erlaubt ist [Eng02]. Diese Leitungslänge ist für die meisten mobilen Roboter ausreichend.

RS-485 [Kug08] (auch bekannt als EIA-485) arbeitet in der physikalischen Schicht (OSI-Schicht 1) ähnlich wie CAN. Es wird über zwei Leitungen ein differentiellles Signal übertragen, was dieses unempfindlicher gegenüber Gleichtaktstörungen macht. Die beiden differentiellen Leitungen sollten am Ende entsprechend ihrer Leitungsimpedanz terminiert werden, um Reflektionen zu vermeiden (üblich sind Kabel mit einer Impedanz von 120 Ohm). RS-485-Busteilnehmer sollten daher in Reihe verschaltet werden, lange Stichleitungen wie sie bei Stern-Topologien nötig sind, sollten vermieden werden. Im Standard werden maximale Datenraten von 10 Mbit/s empfohlen, es sind aber auch Transceiver für höhere Datenraten kommerziell erhältlich. Auch hier hängt die erzielbare Transferrate von der Kabellänge ab [Kin04]. Für die Kompaktheit einer Platine ist vorteilhaft, dass Transceiver-Chips klein sind und es keine Vorgabe für Stecker gibt. Die Spannungsdifferenz der beiden Signalleitungen muss beim Transmitter mindestens 1,5 V betragen, beim Receiver mindestens 0,2 V. RS-485 kann also bei den für Microcontroller üblichen 3,3 V betrieben werden.

RS-485 definiert ausschließlich die physikalische Schicht (Bitübertragungsschicht). Zur Übertragung von Daten kann ein UART (Universal Asynchronous Receiver Transmitter) verwendet werden. Eine UART-Übertragung besteht aus einem Startbit gefolgt von fünf bis neun Datenbits, optional einem Parity-Bit und einem Stoppbit, wobei dieses die einfache, anderthalbfache bzw. zweifache Bitlänge haben kann. Es handelt sich um eine asynchrone Schnittstelle. Die Synchronisierung geschieht mittels der übertragenen Start- und Stoppbits.

RS-485 in Kombination mit UART stellt keine Sicherungsschicht (OSI-Schicht 2) zur Verfügung. Dass jeweils nur ein Teilnehmer sendet und es somit nicht zu Bus-Kollisionen kommt, muss von der darüberliegenden Software sichergestellt werden.

Wegen der genannten Nachteile von Ethernet-basierten Lösungen und der geringen maximalen Datenübertragungsrate von CAN-basierten Lösungen wurde für die DISTAL-Architektur RS-485 gewählt. Tabelle 3 fasst die Vor- und Nachteile der untersuchten Bussysteme zusammen.

Beim gewählten RS-485-Bus ist eine Stern-Topologie mittels eines Hubs möglich. Dieser hat darüber hinaus eine entscheidende Sicherheitsfunktion. Da immer nur Signale aus einem Hub-Anschluss an die anderen durchgeleitet werden können, sind Bus-Kollisionen lokal beschränkt.

	Ethernet	CAN	RS-485
Maximale Datenrate	+	-	o
Biegeflexibilität der Kabel bei ordnungsgemäßer Schirmung	-	+	+
Kompaktheit des elektromechanischen Aufbaus (PHY-Chip und Buchsen)	-	+	+
Echtzeit-Stern-Topologie mittels Hub möglich	-	+	+

Tabelle 3: Eigenschaften der drei untersuchten Bussysteme. Ein Plus kennzeichnet im Vergleich zu den anderen Bussen positive Eigenschaften, ein Minus negative, ein kleiner Kreis durchschnittliche Leistungen.

3.4.2 SpinalCord-Timing

Da RS-485 keine Sicherungsschicht enthält, muss das Protokoll sicherstellen, dass niemals zwei Knoten gleichzeitig senden. Dazu wird ein Zeitschlitzverfahren verwendet, die Kommunikation über den SpinalCord-Bus folgt dabei einem festen Zeitraster. Bei der konkreten Implementierung für ein Robotersystem werden alle Zeiten und Datenmengen für jeden Teilnehmer festgelegt. Abbildung 14 zeigt schematisch das Timing der Architektur.

Der Datenaustausch zwischen den Teilnehmern erfolgt in einer festen Taktrate. Eine reaktive Regelung mit sensomotorischen Schleifen ist bei Robotern laut [Kon02] schon bei 10 Hz möglich, üblich sind höhere Taktraten zwischen 100 Hz und 1 kHz. Bei höheren Datenübertragungsraten oder weniger Daten (bei kleineren Robotern beispielsweise) sind höhere Taktraten möglich als bei niedrigeren Datenübertragungsraten oder mehr Daten.

Taktraten in diesen Dimensionen ermöglichen Reaktionszeiten wie bei biologischen Systemen. Beim Menschen beträgt die Nervenleitgeschwindigkeit zwischen 10 und 110 m/s [Hur39]. Ein Signal vom Fuß eines Menschen zum Gehirn und zurück benötigt also mindestens 30 ms. Die Latenz bei Reflexen liegt beispielsweise beim Unterkieferreflex bei rund 8 ms [OG74] und beim Kniesehnenreflex bei rund 25 ms [RFY+90].

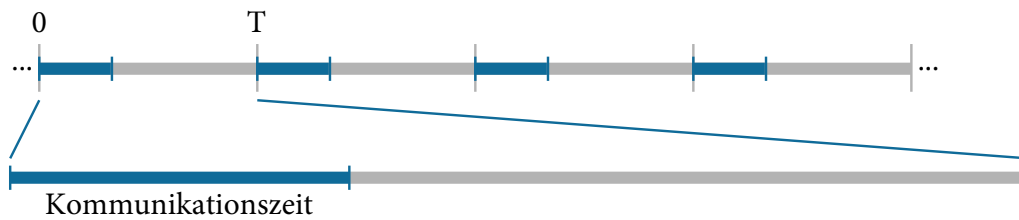


Abbildung 14: Allgemeines Schema der SpinalCord-Kommunikation. Die Kommunikation erfolgt im SpinalCord-Takt (hier mit der Taktlänge T dargestellt). Eine bestimmte Zeit wird kommuniziert (blau), in der restlichen Zeit (grau) können Sensoren erfasst, Berechnungen durchgeführt und Motoren angesteuert werden. In dieser normalerweise kommunikationsfreien Zeit können Transparent-Mode-Datenpakete (siehe Kapitel 3.4.4) versendet werden.

Ein Takt von 100 Hz bedeutet dabei nicht, dass 10 ms lang Daten ausgetauscht werden. Die Kommunikation der Teilnehmer untereinander findet immer am Anfang eines Zeitschlitzes (Slots) statt. Je nach Performanz der teilnehmenden Rechenknoten wird während dieser Zeit exklusiv kommuniziert oder es werden gleichzeitig noch andere Rechenoperationen durchgeführt. Bei Ersterem muss die Kommunikationsdauer so kurz gewählt werden, dass genügend Zeit für die notwendigen anderen Berechnungen verbleibt. Bei Letzterem ist es auch möglich, die sensomotorische Schleife mit einer höheren Taktrate als die Kommunikation zu betreiben.

Um es auch Systemen, die nicht in Echtzeit arbeiten (beispielsweise einem externen PC), zu ermöglichen, Befehle über den gleichen Bus zu versenden, können innerhalb der sonst kommunikationslosen Zeit definierte Datenpakete gesendet werden. Dies wird Transparent Mode genannt und in Kapitel 3.4.4 beschrieben.

Die für die Kommunikation vorgesehene Zeit ist wiederum unter den verschiedenen Teilnehmern aufgeteilt, wie in Abbildung 15 beispielhaft dargestellt. Zur Unterscheidung hat jeder Teilnehmer eine eindeutige ID und erhält abhängig von dieser einen festen Zeitslot innerhalb der gesamten Kommunikationszeit. Nicht jeder Teilnehmer muss dabei dieselbe Kommunikationszeit haben – die Zeit kann entsprechend der notwendigen Datenmenge variieren. Eine geringe Anzahl unterschiedlicher Slotlängen erleichtert die Synchronisierungslogik, es bietet sich an mit zwei Längen auszukommen (zum einen die Minimallänge, zum anderen eine Länge für viele Nutzdaten).

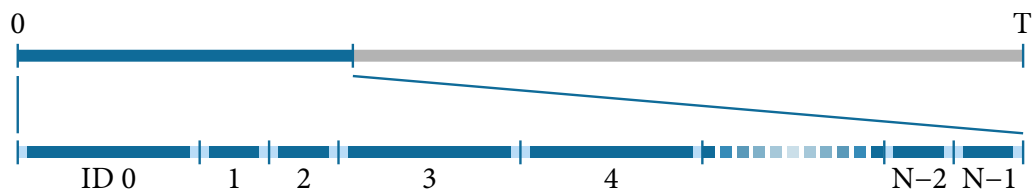


Abbildung 15: Timing eines SpinalCord-Slots. In der Kommunikationsphase kommunizieren N Teilnehmer hintereinander. In diesem Beispiel gibt es zwei Slotlängen: lang (z. B. ID 0 und 3) und kurz (z. B. ID 1 und 2). Jeweils am Anfang und am Ende der Kommunikation eines jeden Teilnehmers gibt es eine kurze Totzeit (hellblau dargestellt).

Die Kommunikationszeit eines jeden Teilnehmers enthält eine kleine Totzeit am Anfang und am Ende, um geringfügige Abweichungen im Timing tolerieren zu können. Diese sollte sich in der Dimension bewegen, die zur Übertragung eines Bytes benötigt wird.

3.4.3 SpinalCord-Datenstruktur

Jeder SpinalCord-Teilnehmer kann eine definierte Anzahl 16-Bit-Werte übertragen. Physikalisch wird bei jedem Byte zunächst das niederwertigste Bit übertragen. Die Byte-Reihenfolge ist entsprechend, es wird also zunächst das niederwertige Byte übertragen, gefolgt vom höherwertigen (Little Endian).

Ein spezifischer Wert an einer festen Position wird als SpinalCord-Feld bezeichnet. Diese sind durchgehend nummeriert. Überträgt das Board mit der ID 0 beispielsweise die Felder 0 bis 26, starten die Felder der Board-ID 1 bei 27. Die ersten drei Felder eines Teilnehmers haben eine feste Bedeutung, woraus sich auch eine Mindestlänge von drei 16-Bit-Werten pro Teilnehmer ergibt. Abbildung 16 zeigt die ersten drei Werte schematisch.

Im ersten Feld befindet sich immer ein fester Synchronisierungswert, der auf $0x5555$ festgelegt ist. Dies entspricht dem Bitmuster 0101 0101 0101 0101, es ist also eine Gleichverteilung der Bits bei gleichmäßigem Muster und maximaler Frequenz zu beobachten. Dieses Muster ist zu Debuggingzwecken auch einfach auf dem Oszilloskop zu erkennen.

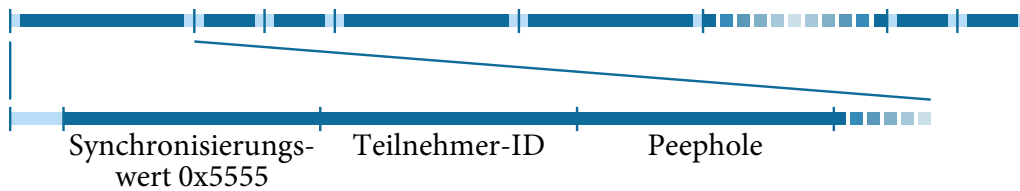


Abbildung 16: Die ersten drei SpinalCord-Felder eines jeden Teilnehmers: Synchronisierungswert, Teilnehmer-ID und Peephole.

Es folgt ein Feld, das die ID des Teilnehmers enthält. Bei Nutzung aller 16 Bit sind also bis zu 65.536 Bus-Teilnehmer möglich. Da so viele Teilnehmer in der Praxis nicht notwendig sind, ist auch die Nutzung einzelner Bits für andere Informationen (Modus, Zustand des Systems) denkbar und muss dann entsprechend bei der Implementierung der Synchronisierung berücksichtigt werden.

Das dritte Feld wird „Peephole“ (engl. für Guckloch) genannt und dient zur Übertragung sich langsam ändernder Daten. In jedem Zeitschritt werden dabei andere Daten übertragen. Die 16 Bit des Werts werden dazu aufgeteilt in Index-Bits und Bits für die eigentlichen Daten. Beispielsweise ist es bei einer Aufteilung auf 4 Index-Bits und 12 Daten-Bits möglich, innerhalb des Peepholes $2^4 = 16$ Werte mit je 12 Bit zu übertragen. Die Index-Bits sind die höherwertigen Bits und werden immer aufsteigend durchgegangen. Tabelle 4 zeigt die möglichen Kombinationen von Index- und Daten-Bits bei bis zu 8 Index-Bits. Jedes normale Datenfeld entspricht im Prinzip einem Peephole mit 0 Index-Bits.

Da die Index-Bits immer aufsteigend durchgezählt werden, können diese als unterste Bits einer Uptime verwendet werden. Höhere Uptime-Bits können in Datenfeldern des Peepholes abgelegt werden. So steht die gesamte Uptime zwar bei einem einzelnen SpinalCord-Slot nicht zur Verfügung, durch das Mitlesen von mehreren Slots lässt sich aber die korrekte Uptime eines Teilnehmers ermitteln.

Hat ein SpinalCord-Teilnehmer mehr als 3 Felder zur Verfügung, können diese beliebig genutzt werden. In diesen Feldern werden beispielsweise Sensor-Daten, neuronale Aktivierungen oder auch aktuelle Ansteuerungswerte übertragen. Diese Werte werden üblicherweise als Festkommawerte im Intervall $[-1, +1]$ mit 15 Nachkomma-Bits interpretiert.

Index-Bits	Daten-Bits	Mögliche Anzahl Werte	Übertragungshäufigkeit eines Wertes bei 100 Hz
1	15	2	alle 20 ms (50 Hz)
2	14	4	alle 40 ms (25 Hz)
3	13	8	alle 80 ms (12,5 Hz)
4	12	16	alle 160 ms (6,25 Hz)
5	11	32	alle 320 ms (3,125 Hz)
6	10	64	alle 640 ms (~1,56 Hz)
7	9	128	alle 1,28 s (~0,78 Hz)
8	8	256	alle 2,56 s (~0,39 Hz)

Tabelle 4: Mögliche Kombinationen von Index-Bits und Daten-Bits für das Peephole (für 1 bis 8 Index-Bits).

Die Systemsoftware muss sicherstellen, dass bei Wegfall eines Kommunikationsteilnehmers die Daten aller SpinalCord-Felder dieses Teilnehmers auf null abklingen. Die ersten drei Felder (Sync, ID und Peephole) werden sofort auf null gesetzt. Ob ein Teilnehmer vorhanden ist, kann einfach durch Prüfen des Sync-Werts festgestellt werden. Alle Felder ab dem vierten Feld klingen langsam ab (linear; innerhalb weniger Sekunden auf Null). Auf diese Weise ist sichergestellt, dass Regelungsschleifen keine sich plötzlich ändernden Sensorwerte erhalten.

3.4.4 Transparent Mode

Als Transparent Mode wird die Möglichkeit bezeichnet, während der kommunikationsfreien Zeit Datenpakete ohne exaktes Timing über den Bus zu senden. Dies ist dann möglich, wenn das Zeitfenster so lang ist, dass ein nicht echtzeitfähiges Gerät ein Transparent-Mode-Datenpaket senden kann.

Da ein solches Gerät das Timing nicht garantieren kann, ist dieses Verfahren nur bedingt zuverlässig. Mit einem Oszilloskop kann man die Reaktionsgeschwindigkeiten der zu nutzenden Geräte testen und dementsprechend das Zeitfenster lang genug wählen. Als Reaktionszeit wird in diesem Fall die Zeit zwischen dem

Ende eines SpinalCord-Datenpakets und dem Versenden des Transparent-Mode-Datenpakets durch das externe Gerät bezeichnet.

Bei gleichbleibender Baudrate erhöht ein längeres Zeitfenster die Zuverlässigkeit, ein kürzeres Zeitfenster ermöglicht die Übertragung von mehr Daten oder die Erhöhung der SpinalCord-Taktrate. Wird beispielsweise ein externer Desktop-PC mit einem USB-zu-Serial-Adapter angeschlossen, sollte die einzuplanende Reaktionszeit mindestens 5 ms betragen. Dazu kommt die Übertragungszeit des Transparent-Mode-Datenpakets. Diese sollte also möglichst kurz gewählt werden.

Wie im SpinalCord-Protokoll werden auch im Transparent Mode 16-Bit-Werte übertragen. Der Transparent Mode kann für zwei unterschiedliche Funktionen genutzt werden. Zum einen ist es möglich, Motoransteuerungswerte zu senden (Transparent-Mode-Motorsteuerung), zum anderen können einzelne Befehle gesendet werden (Transparent Command). Beide Möglichkeiten unterscheiden sich durch den ersten 16-Bit-Wert (0xCCCC bei der Transparent-Mode-Motorsteuerung, 0xDDDD beim Transparent Command). Der letzte 16-Bit-Wert ist immer eine Prüfsumme, die berechnet wird, indem alle vorherigen 16-Bit-Werte addiert und dann bitweise invertiert werden.

Bei der Transparent-Mode-Motorsteuerung (Start-Wert 0xCCCC) empfängt ein externes Gerät (in diesem Fall normalerweise ein PC) die Sensordaten des SpinalCords, berechnet neue Motoransteuerungsdaten und sendet diese als Transparent-Mode-Paket. Alle angeschlossenen Teilnehmer übernehmen die Werte entsprechend einer festgelegten Zuordnung in ihre SpinalCord-Felder, lokale Regelungsprogramme werden nicht ausgeführt. Da dieser Modus zur Motoransteuerung genutzt wird, muss die Anzahl der Werte im Transparent-Mode-Datenpaket mindestens der Anzahl der Motoren des Roboters entsprechen.

Beim Transparent Command (Start-Wert 0xDDDD) wird als zweiter 16-Bit-Wert ein Befehl gesendet, die folgenden Werte können Parameter für diesen Befehl sein. Während Datenpakete der Transparent-Mode-Motoransteuerung dauerhaft in der Taktrate des SpinalCords gesendet werden, werden Transparent-Command-Pakete normalerweise nur einmalig gesendet. Je nach Anwendungszweck können eigene Befehle definiert werden, die Kommandos mit den niedrigsten Codes sind jedoch fest belegt und in Tabelle 5 dargestellt. Der Befehl 0x0001 veranlasst die Teilnehmer dazu, die Kommunikation einzustellen, der Bus befindet sich dann in einem Zustand, in dem ohne exaktes Timing mit den

Teilnehmern kommuniziert werden kann. Dies kann beispielsweise für Updates der Software eines Teilnehmers verwendet werden. Weitere Kommandos sind definiert zum Starten und Stoppen der Berechnung der sensomotorischen Programme bzw. zum Fortsetzen der Berechnung (Starten ohne Zurücksetzen der internen Zustandsvariablen).

Kommando	Befehl
0x0001	Beende SpinalCord-Kommunikation.
0x0002	Starte Berechnung des sensomotorischen Programms.
0x0003	Beende Berechnung des sensomotorischen Programms.
0x0004	Setze Berechnung des sensomotorischen Programms fort (keine Initialisierung interner Zustandsvariablen).

Tabelle 5: Mögliche Transparent-Command-Befehle.

Natürlich kann ein Transparent-Mode-Paket nicht nur von einem Gerät ohne Echtzeitfähigkeiten gesendet werden sondern auch von einem Echtzeit-Teilnehmer, beispielsweise einem SpinalCord-Teilnehmer selbst.

3.4.5 Synchronisierung und Laufzeit-Metamorphose

Die Besonderheit der beschriebenen Systemarchitektur ist, dass es keinen ausgezeichneten Master gibt. Jeder Teilnehmer ist alleine und in jeder Kombination mit anderen Teilnehmern vollständig funktionsfähig. Das System ist außerdem so ausgelegt, dass die Änderungen auch während der Laufzeit stattfinden können (Laufzeit-Metamorphose). Teile eines Roboters können so im Betrieb hinzugefügt bzw. entfernt werden. In diesem Kapitel wird der Synchronisierungsprozess beschrieben.

Wird ein Teilnehmer gestartet, hört er zunächst auf dem Bus mit (mindestens ein SpinalCord-Zeitfenster). Empfängt er in dieser Zeit die Daten eines anderen Teilnehmers, berechnet er aus dessen ID, wo im Zeitfenster sich die Kommunikation gerade befindet und sendet ab dem nächsten Zeitfenster im für ihn vorgesehenen Slot. Er integriert sich also nahtlos in die Übertragung und die bereits

vorhandenen Teilnehmer senden wie gewohnt weiter ohne aussetzen zu müssen. Sendet jedoch kein anderer Teilnehmer, fängt der synchronisierende Teilnehmer selbst an zu senden.

Üblicherweise werden beim Anschalten eines Roboters alle Teilnehmer gleichzeitig gestartet. Damit diese nicht gleichzeitig anfangen zu senden, warten die Teilnehmer zusätzlich entsprechend ihrer SpinalCord-ID. Pro ID wird ein SpinalCord-Zeitfenster gewartet. Diese Verzögerung ist einmalig und relativ kurz, bei der Arbeit mit einem Roboter ist sie kaum zu bemerken.

Da unterschiedliche Quarze leicht unterschiedlich schwingen, wird nicht nur einmalig synchronisiert. Im Laufe der Zeit würden die verschiedenen Teilnehmer auseinanderdriften und es würde zu Kollisionen auf dem Bus kommen. In jedem Zeitfenster synchronisieren sich alle Teilnehmer deshalb auf den Teilnehmer mit der niedrigsten ID, also den, der als erster sendet. Somit richtet sich das Zeitfenster nach dem Takt des Teilnehmers mit der niedrigsten ID, der in dieser Zeit eine Art Master-Rolle einnimmt. Bemerkt ein Teilnehmer jedoch, dass dieser „Master“ weggefallen ist, also bis zu seinem eigenen Sende-Zeitslot kein Board gesendet hat, übernimmt er nahtlos diese Rolle und alle folgenden richten sich nach ihm, da er nun wiederum der sendende Teilnehmer mit der niedrigsten ID ist.

Stellt ein Board fest, dass in der eigentlich sendefreien Zeit ein Synchronisierungswert (0x5555) gesendet wurde, geht es von einem Synchronisierungsfehler aus und startet eine Neusynchronisierung. Es wartet dabei wieder entsprechend seiner ID, zusätzlich jedoch noch eine zufällige Zeit, um die Wahrscheinlichkeit einer erneuten Kollision zu verringern.

3.5 Die DISTAL-Architektur am Beispiel des Roboters Myon

In diesem Kapitel wird die Auslegung der DISTAL-Systemarchitektur für den humanoiden Roboter Myon (siehe Kapitel 2) erläutert.

3.5.1 Überblick über die Teilnehmer beim Roboter Myon

Bei Myon gibt es verschiedene Kommunikationsteilnehmer. Zum einen Boards mit einem ARM-Prozessor zur sensomotorischen Steuerung (AccelBoard3D, siehe Kapitel 3.6), zum anderen ein Board mit einem FPGA zur Bild-/Audioverarbeitung und weiterer höherer Verarbeitung (BrainModule, siehe Kapitel 3.7).

Weiterhin gibt es in jedem Körperteil ein EnergyModule, an das ein Akku angeschlossen ist. Dies versorgt das Körperteil mit Strom, kann aber bei Verbindung mit anderen EnergyModules auch den Strom anderer Akkus verwenden (Mischung mittels idealer Dioden, hier LTC4412). Die EnergyModules untereinander sind über den Extended SpinalCord miteinander verbunden. Über diesen werden die Stromquellen aller EnergyModules miteinander verbunden, aber auch die Signale des SpinalCords über den Körper verteilt. Die derzeitigen EnergyModules haben keinen Prozessor zur Kommunikation, eine entsprechende Erweiterung ist für die Zukunft vorgesehen.

Ein humanoider Roboter besitzt eine natürliche Stern-Architektur. Vom Torso aus gibt es fünf sternförmig abgehende Körperteile: zwei Arme, zwei Beine und den Kopf. Da RS-485 wie in Kapitel 3.4.1 beschrieben wegen der nötigen Abschlusswiderstände nicht für sternförmige Topologien geeignet ist, wurde im Torso ein speziell entwickelter Hub verbaut. Dieser hat neben den sechs Ports für die sechs Körperteile einen siebten Port zum Anschluss eines externen PCs.

Im Roboter Myon gibt es bis zu 32 SpinalCord-Teilnehmer. Die ID 0 wird vom FPGA auf dem BrainModule im Kopf betrieben, die IDs 7 bis 31 von den AccelBoard3D-Platinen. Die IDs 1 bis 6 sind für erweiterte EnergyModules vorgesehen, die den Ladezustand der Batterien übertragen können. Obwohl die Entwicklung dieser Module noch nicht abgeschlossen ist, wurden bei der Planung des Protokolls die Slots bereits vorgesehen. Da das Fehlen eines Kommunikationsteilnehmers keine Auswirkungen auf das Gesamtsystem hat, können diese Module nachgerüstet werden, sobald sie verfügbar sind, ohne etwas am System ändern zu müssen.

Ebenfalls um zukünftigen Entwicklungen Rechnung zu tragen, wurden vier IDs zunächst nicht belegt. Diese können für Testaufbauten verwendet werden, die jederzeit an den Roboter Myon angeschlossen werden können. Vorgesehen ist außerdem ein Logging des SpinalCords auf Festplatte oder Flash-Speicher. Die Logging-Einheit könnte ebenfalls über eine der freien IDs kommunizieren und

beispielsweise den Füllstand des Datenspeichers mitteilen. Tabelle 6 zeigt die 32 möglichen Teilnehmer des SpinalCords beim Roboter Myon.

Anzahl	Teilnehmer	IDs
1	BrainModule (FPGA-Modul)	0
6	EnergyModule (vorgesehen)	1–6
21	AccelBoard3D	7–25, 30–31
4	Freie IDs für Erweiterungen	26–29

Tabelle 6: Die 32 möglichen SpinalCord-Teilnehmer des Roboters Myon.

Abbildung 17 auf der nächsten Seite zeigt für den Roboter Myon schematisch alle Teilnehmer des SpinalCords und deren IDs. Die Positionierung der einzelnen IDs ist weitgehend willkürlich. Hier wurde eine aufsteigende Nummerierung von unten nach oben gewählt, wobei ungerade IDs auf der linken Körperhälfte platziert wurden.

3.5.2 Datenübertragung beim Roboter Myon

Als Datenübertragungsrate wurde 4,5 MBaud gewählt, da dies die maximale Datenrate ist, die die im Prozessor der AccelBoard3D-Platinen integrierte UART-Peripherie zulässt. Zur Datenübertragung wurde die übliche Kombination von acht Datenbits, keinem Paritybit und einem Stoppbit gewählt (8N1). Zur Übertragung eines Bytes sind insgesamt also 10 Bits nötig. Die Übertragung eines Bits dauert 222,22 ns, die eines Bytes somit 2,22 μ s und die eines 16-Bit-Werts 4,44 μ s.

Als RS-485-Transceiver werden auf allen Teilnehmern im Roboter Myon die Chips LTC2850 (ohne integrierten Terminator) bzw. LTC2854 (mit hinzuschaltbarem integriertem Terminator) verwendet. Um auf den SpinalCord-Bus vom PC aus zugreifen zu können, wird ein eigens entwickelter, galvanisch entkoppelter, Adapter mit dem RS-485-zu-USB-Konverter-Chip MCS7820 verwendet („Matchbox 2.0“).

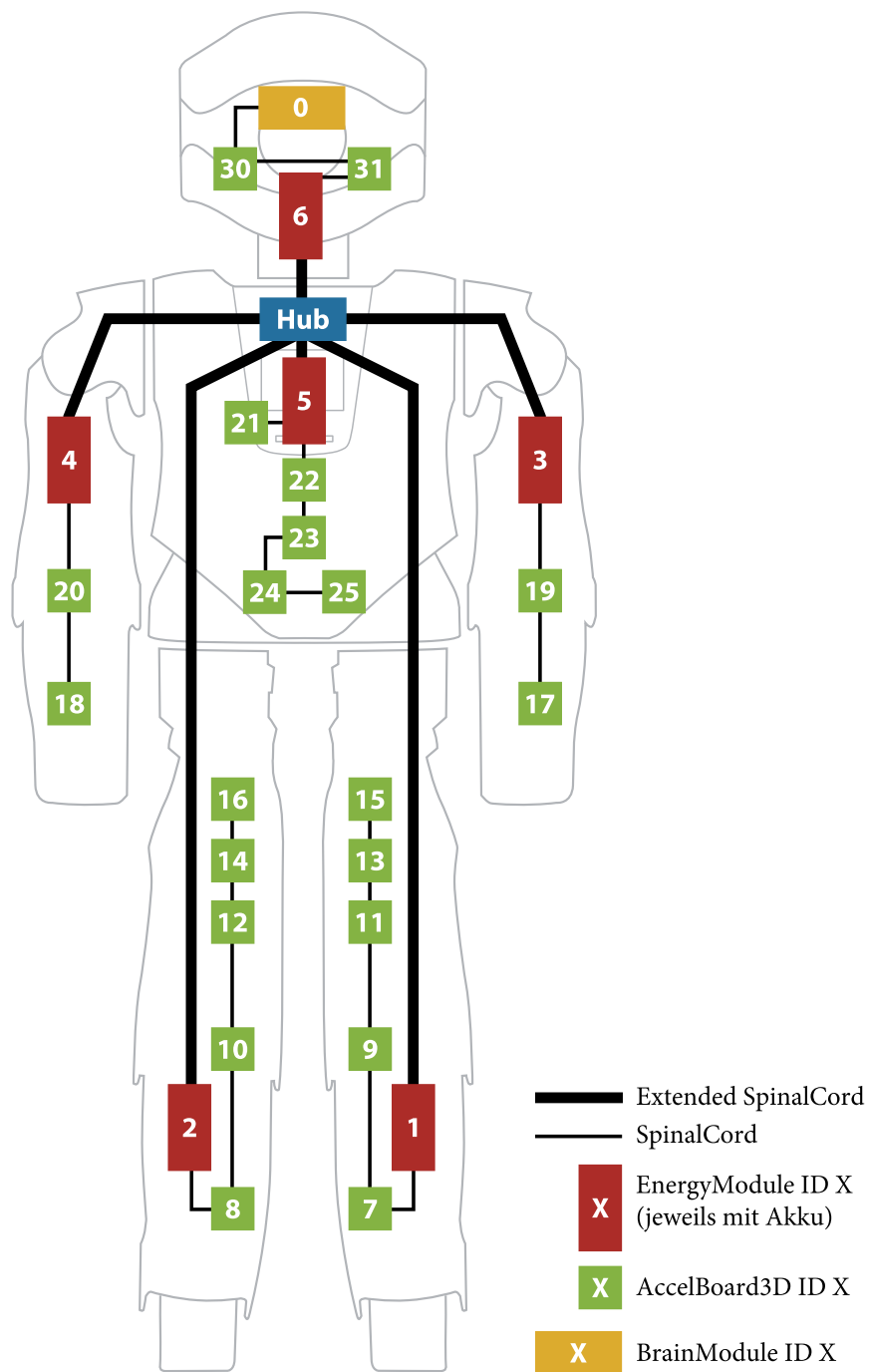


Abbildung 17: Schematische Darstellung des Roboters Myon mit den Kommunikationsteilnehmern des SpinalCords. Im Bein sitzen die Akkus wegen des tieferen Schwerpunkts unten.

3.5.3 SpinalCord-Timing beim Roboter Myon

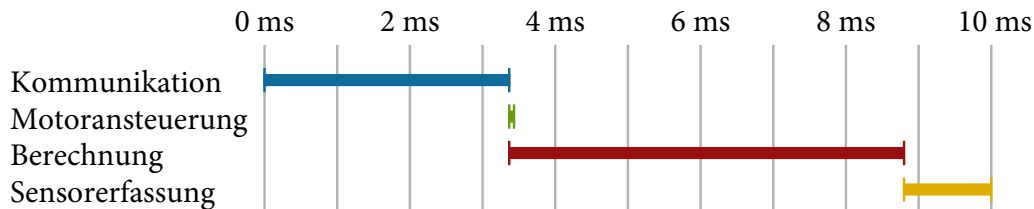


Abbildung 18: Timing innerhalb eines 10-ms-Slots beim Roboter Myon: Innerhalb der ersten 3,36 ms wird kommuniziert, anschließend werden die Motoren angesteuert und Berechnungen durchgeführt. In den letzten 1,2 ms vor der nächsten Kommunikation werden Sensordaten erfasst.

Der Datenaustausch zwischen den Teilnehmern des Roboters Myon erfolgt alle 10 ms, die sensomotorische Schleife läuft somit bei 100 Hz. Für die Datenübertragung sind die ersten 3,36 ms vorgesehen. Das gesamte Timing ist in Abbildung 18 dargestellt. Die Motoransteuerung, Sensorerfassung und Berechnung neuronaler Netze werden in der kommunikationsfreien Zeit durchgeführt. Bei ausreichend leistungsfähiger Hardware kann die Berechnung auch parallel zur Kommunikation ablaufen. Dies wurde in einem anderen Projekt⁵ umgesetzt, bei dem die sensomotorische Schleife mit 1 kHz betrieben wird, wohingegen die Kommunikation zwischen mehreren Teilnehmern weiterhin kompatibel zu der des Roboters Myon ist.

Die Motoransteuerung wird nicht direkt nach dem Ende der Berechnung durchgeführt, sondern zu einem festen Zeitpunkt nach der Kommunikation. Dadurch wird sichergestellt, dass alle Motoren zur identischen Zeit angesteuert werden. Gleichzeitig wird das Regelungsprogramm (der Code zur Berechnung der sensomotorischen Schleife) gestartet. Es muss 1,2 ms vor dem Ende des 10-ms-Zeitfensters abgeschlossen sein. Zur Berechnung stehen also 5,44 ms und somit über 50 % der Rechenzeit zur Verfügung. Die im Kapitel 4 vorgestellte Software zur Erstellung der Regelungsprogramme (BrainDesigner) stellt für den Roboter

⁵ Da diese Arbeit veröffentlicht wird, können wegen einer Verschwiegenheitserklärung gegenüber dem Auftraggeber keine detaillierteren Angaben zu diesem Projekt gemacht werden.

Myon sicher, dass diese Zeit eingehalten wird. In den letzten 1,2 ms werden Sensorwerte ermittelt, die somit bei Beginn der Übertragung des SpinalCords maximal 1,2 ms alt sind. Da die Sensordaten direkt vor der Kommunikation erhoben werden, stehen zur Berechnung der Motordaten im nächsten Zeitschritt die aktuellen Sensordaten aller angeschlossenen Boards zur Verfügung.

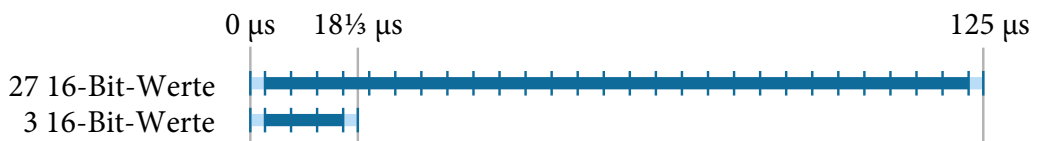


Abbildung 19: Timing eines SpinalCord-Slots mit 27 16-Bit-Werten und eines Slots mit 3 16-Bit-Werten. Am Anfang und am Ende befinden sich jeweils 2,5 µs Totzeit. Die Übertragung von jeweils einem 16-Bit-Wert benötigt 4,44 µs.

Es gibt zwei verschiedene Übertragungszeiten für SpinalCord-Teilnehmer. Für die EnergyModules wird die minimale Feldanzahl von drei verwendet, für alle anderen Teilnehmer sind 27 SpinalCord-Felder vorgesehen. Zusammen mit einer kurzen Totzeit von jeweils 2,5 µs am Anfang und am Ende ergibt sich eine Übertragungsdauer von 18 1/3 µs für drei SpinalCord-Felder und 125 µs für 27 SpinalCord-Felder. Das Timing für beide Varianten ist in Abbildung 19 dargestellt, das gesamte Timing in Abbildung 20.

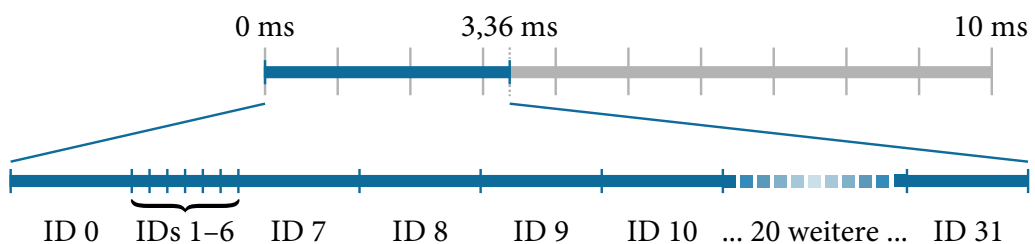


Abbildung 20: Timing der unterschiedlichen SpinalCord-Teilnehmer beim Roboter Myon. Die IDs 1 bis 6 übertragen weniger Daten als die IDs 0 und 7 bis 31.

3.5.4 SpinalCord-Datenstruktur beim Roboter Myon

Beim Roboter Myon schreiben die Rechenknoten mit den IDs 0 und 7 bis 31 jeweils 27 Datenfelder, die Rechenknoten mit den IDs 1 bis 6 jeweils drei Datenfelder. Die ID 0 belegt die Felder 0 bis 26, die ID 1 die Felder 27 bis 29 usw. Insgesamt ergeben sich somit 720 Datenfelder, die vollständig in Anhang 1 dargestellt sind.

Vom ID-Feld jedes Teilnehmers (zweites Feld) werden nur die unteren fünf Bits genutzt, womit sich 32 Teilnehmer kodieren lassen. Die anderen Bits bleiben 0. Das Peephole (drittes Feld) wird in 4 Index-Bits und 12 Daten-Bits aufgeteilt, die einzelnen Werte werden also alle 160 ms aktualisiert. In Peephole-Index 0 werden die Bits 5 bis 16, in Peephole-Index 1 die Bits 17 bis 28 der Uptime übertragen. Zusammen mit den vier Index-Bits ergibt sich eine 28-Bit-Uptime. Bei 100 Hz SpinalCord-Frequenz ergibt sich ein Überlauf erst nach über 31 Tagen durchgängiger Betriebszeit. Zusätzlich zur Uptime werden die gemessene Versorgungsspannung, die Version der Systemsoftware und Motor-Temperaturen im Peephole übertragen.

Nach dem Peephole folgen Sensor-Daten. Beim AccelBoard3D werden beispielsweise im vierten bis sechsten Wert die gemessenen Beschleunigungen in drei Achsen übermittelt. Berechnete Ansteuerungswerte für Motoren sind am Ende der Slots positioniert.

Abbildung 21 zeigt für die fünf Boards des linken Beins beispielhaft die Belegung der Felder. Nicht gefüllte Felder können von der Berechnung zur sensomotorischen Ansteuerung verwendet werden, um Zwischenergebnisse zwischen den Boards auszutauschen. An die gezeigten Boards ist eine unterschiedliche Anzahl Motoren angeschlossen, was zu einer fragmentierten Verteilung freier Felder führt. Diese können trotzdem beliebig genutzt werden. Die Peephole-Indizes 8 bis 15 werden derzeit nicht genutzt und werden deshalb nicht dargestellt.

7	Left Leg Lower Bottom	9	Left Leg Lower Top	11	Left Leg Upper Bottom	13	Left Leg Upper Middle	15	Left Leg Upper Top
45	SYNC	99	SYNC	153	SYNC	207	SYNC	261	SYNC
46	ID	100	ID	154	ID	208	ID	262	ID
47	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M1 5 6 7	101	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M3 5 Temp M5 6 Temp M7 7 Temp M9	155	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M13 5 Temp M15 6 Temp M11 7	209	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M17 5 Temp M19 6 Temp M21 7	263	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M23 5 6 Temp M25 7
48	AccelX	102	AccelX	156	AccelX	210	AccelX	264	AccelX
49	AccelY	103	AccelY	157	AccelY	211	AccelY	265	AccelY
50	AccelZ	104	AccelZ	158	AccelZ	212	AccelZ	266	AccelZ
51	Current DS A M1	105	Current DS A M3+M5	159	Current DS A M13+M15	213	Current DS A M17+M19	267	Current DS A M23
52		106	Current DS B M7+M9	160	Current DS B M11	214	Current DS B M21	268	Current DS B M25
53	DS A, Position 1 Position 1	107	DS A, Position 1 Position 3	161	DS A, Position 1 Position 13	215	DS A, Position 1 Position 17	269	DS A, Position 1 Position 23
54		108	DS A, Position 2 Position 5	162	DS A, Position 2 Position 15	216	DS A, Position 2 Position 19	270	
55		109	DS B, Position 1 Position 7	163	DS B, Position 1 Position 11	217	DS B, Position 1 Position 21	271	DS B, Position 1 Position 25
56		110	DS B, Position 2 Position 9	164		218		272	
57	Anale LToes	111	Anale LAnklePitch	165	Anale LKnee	219	Anale LHipPitch	273	Anale LHipRoll
58	Anale LAnkleRoll	112		166		220		274	
59	Force LLeftBack	113		167		221		275	
60	Force LLeftFront	114		168		222		276	
61	Force LRightBack	115		169		223		277	
62	Force LRightFront	116		170		224		278	
63		117		171		225		279	
64		118		172		226		280	
65		119		173		227		281	
66		120		174		228		282	
67		121		175		229		283	
68	DS A, Torque 1 Motor 1	122	DS A, Torque 1 Motor 3	176	DS A, Torque 1 Motor 13	230	DS A, Torque 1 Motor 17	284	DS A, Torque 1 Motor 23
69		123	DS A, Torque 2 Motor 5	177	DS A, Torque 2 Motor 15	231	DS A, Torque 2 Motor 19	285	
70		124	DS B, Torque 1 Motor 7	178	DS B, Torque 1 Motor 11	232	DS B, Torque 1 Motor 21	286	DS B, Torque 1 Motor 25
71		125	DS B, Torque 2 Motor 9	179		233		287	

Abbildung 21: SpinalCord-Felder beim Roboter Myon. Gezeigt werden die Felder der fünf Boards des linken Beins. Sync- und ID-Feld sind türkis dargestellt, das Peephole rosa, Sensordaten gelb, Ansteuerungsdaten für Motoren grün. Felder, die von sensomotorischen Programmteilen frei genutzt werden können sind blau.

3.5.5 Transparent Mode beim Roboter Myon

Ein Transparent-Mode-Datenpaket besteht beim Roboter Myon immer aus 64 16-Bit-Werten, also 128 Byte insgesamt. Die Übertragung dieses Pakets benötigt bei 4,5 MBaud insgesamt 284,44 μ s. Das Paket muss verarbeitet werden, bevor die nächste SpinalCord-Kommunikation stattfindet. Daher muss es 1,2 ms vor Ende des Zeitslots vollständig empfangen worden sein, also zu Beginn der Sensorverarbeitung. Das gesamte mögliche Zeitfenster ist somit 5,44 ms lang. Frühestmöglicher Zeitpunkt des Kommunikationsstarts innerhalb des 10-ms-Zeitfensters ist nach der SpinalCord-Kommunikation (zum Zeitpunkt 3,36 ms), spätestmöglicher Startzeitpunkt ist 1,2 ms plus Übertragungszeit vor der nächsten SpinalCord-Kommunikation (zum Zeitpunkt 8,52 ms).

Es hat sich gezeigt, dass die Performanz eines modernen PCs in Verbindung mit den 1-ms-USB-Zeitscheiben bei einem vollständigen Roboter nicht zuverlässig genug ist, um die sensomotorische Schleife über den PC laufen zu lassen. Für ein einzelnes Körperteil kann sie jedoch per Transparent Mode betrieben werden. Das Versenden von einzelnen Transparent Commands funktioniert auch beim vollständigen Roboter zuverlässig.

3.6 Implementierung auf dem AccelBoard3D

Das AccelBoard3D ist eine im Labor für Neurorobotik entwickelte Platine mit Prozessor, Sensorik und verschiedenen Anschlussmöglichkeiten. Das Board wurde zwar für den Roboter Myon entwickelt, dabei aber bewusst so allgemein gehalten, dass es auch für viele andere Zwecke eingesetzt werden kann. So wird beispielsweise auch der Roboter Semni [Hil13] von einem AccelBoard3D betrieben. Ebenso ist es in mehreren Testaufbauten zur Sensordatenerfassung im Einsatz.

Die Software teilt sich in zwei Bereiche: Die *Systemsoftware* sorgt dafür, dass sämtliche Sensordaten 100 Mal pro Sekunde erfasst und zwischen den Körperteilen ausgetauscht werden. Die Berechnung von Ansteuerungsdaten für die Motoren aus den Sensordaten geschieht auf dem Roboter mit der *Anwendungssoftware*. In der DISTAL-Architektur werden künstliche neuronale Netze zur Bewegungssteuerung verwendet. Die Anwendungssoftware wird in diesem Fall mit der Desktop-Software BrainDesigner erstellt (siehe Kapitel 4).

3.6.1 Aufbau des AccelBoard3D

Die Platine des AccelBoard3D ist 45×49 mm groß. Kernstück ist ein ARM-Cortex-M3-Prozessor des Herstellers ST Microelectronics mit 48 Pins (STM32F103CBT6 [STM08]), der mittels eines externen 12-MHz-Quarzes und einer PLL bei 72 MHz betrieben wird. Der ARM-Cortex-M3-Prozessor wurde wegen seiner guten Echtzeitfähigkeiten gewählt, beispielsweise hat diese Prozessorfamilie nur eine Interrupt Latency von 12 Zyklen und unterstützt Tail-Chaining von Interrupts.

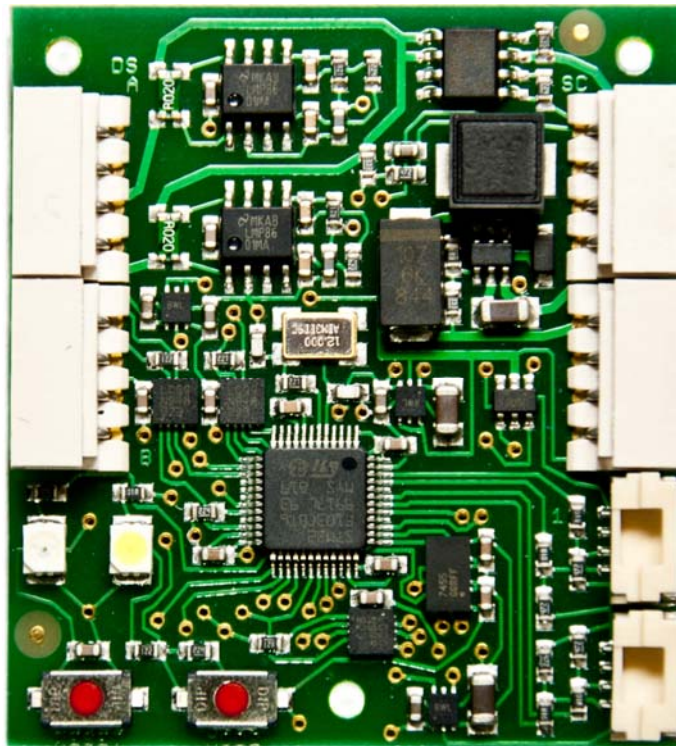


Abbildung 22: AccelBoard3D: Im Original misst die Platine 45×49 mm, ein Viertel der hier dargestellten Größe.

Da diese Platine in mobilen autonomen Robotern eingesetzt wird, wurde darauf geachtet, die Größe, das Gewicht und den Stromverbrauch möglichst gering zu halten. Abbildung 22 zeigt eines der Boards. Der vollständige Schaltplan findet sich in Anhang 2.

Zur Interaktion gibt es neben zwei LEDs und einem Taster zum Zurücksetzen des Prozessors (Reset; zieht bei Aktivierung den Eingang NRST des Prozessors auf Masse) einen weiteren Taster, der während des Prozessorbetriebs als normaler Eingang genutzt werden kann, bei Aktivierung aber auch den BOOT0-Eingang des Prozessors auf dessen Versorgungsspannung zieht. Ist dies während des Prozessor-Resets der Fall, wird der interne Bootloader des Prozessors aktiv und neue Software kann über eine serielle Schnittstelle aufgespielt werden. Für diesen Zweck gibt es auf der Platine Löcher zur Aufnahme eines Nadelinterfaces, über das die serielle Schnittstelle des Prozessors mit einem externen PC verbunden werden kann. Da dieses Vorgehen umständlich und zeitaufwendig ist wenn ein Board schlecht zugänglich in einem mobilen Roboter verbaut ist, wurde ein zusätzlicher Bootloader (DISTAL-Bootloader) implementiert, der das Selbst-flashen über den gemeinsamen SpinalCord-Bus erlaubt (siehe Kapitel 3.6.2).

Im Roboter Myon werden die AccelBoard3D-Platinen mit rund 15 V betrieben (bei Akkubetrieb nominell 14,8 V). Die Spannung kann an die an das Board angeschlossenen Motoren durchgeleitet werden. Für den Prozessor und dessen Peripherie wird die Eingangsspannung zweistufig auf 3,3 V heruntertransferiert. Ein Schaltregler (MAX1836) erzeugt zunächst eine Zwischenspannung von 3,9 V. Mehrere Linearregler (drei Dual TPS71933-33) erzeugen daraus sechs voneinander unabhängige 3,3-V-Pegel. Verschiedene Teile (beispielsweise der Beschleunigungssensor oder Treiberchips für die seriellen Schnittstellen) können sich so gegenseitig nicht stören und getrennt an- und abgeschaltet werden. Müssen keine externen Motoren mit Spannung versorgt werden, kann das AccelBoard3D bei einem Spannungsbereich des Schaltreglers von 4,5 V bis 24 V betrieben werden.

Auf dem AccelBoard3D befinden sich sowohl links oben als auch rechts oben zwei vierpolige Anschlussbuchsen (siehe Abbildung 22). Diese Anschlüsse haben eine ähnliche Belegung: Eine Leitung ist Masse, über eine läuft die Versorgungsspannung und über die zwei verbliebenen Leitungen wird ein differentielles RS-485-Datensignal übertragen (als RS-485-Transceiver kommen LTC2854 zum Einsatz). Rechts oben befinden sich zwei identische Anschlüsse für den gemeinsamen Datenbus der AccelBoard3D-Platinen (SpinalCord-Bus, auf der Platine mit „SC“ gekennzeichnet), links oben gibt es zwei getrennte sogenannte „DoubleStrands“. An diese werden im Myon Motoren des Typs Dynamixel RX-28 angeschlossen. Die beiden DoubleStrands – genannt DoubleStrand A (DSA,

oben) und DoubleStrand B (DSB, unten) – können vom Prozessor an- und abgeschaltet werden. Dies bezieht sich dann sowohl auf die Durchleitung der Versorgungsspannung als auch auf die Aktivierung der RS-485-Transceiver. Somit ist es möglich, die Motoren gänzlich auszuschalten um Strom zu sparen. Vier angeschlossene Dynamixel RX-28 benötigen einen Stand-by-Strom von jeweils 50 mA [Rob08], insgesamt also 200 mA, wohingegen der Prozessor als einziges nicht-abschaltbares Element des AccelBoards (neben der Stromversorgung) im Betrieb maximal 150 mA verbrauchen kann [STM08]. Umgekehrt ist es aber auch bei ausgeschalteten DoubleStrands über einer Rücklaufdiode möglich, das AccelBoard3D über diese Anschlüsse mit Strom zu versorgen.

Zusätzlich zu den vier SpinalCord- und DoubleStrand-Anschlüssen gibt es zwei kleinere ebenfalls vierpolige Anschlüsse (siehe Abbildung 22; rechts unten). Über diese kann zusätzliche Sensorik mittels I²C angeschlossen werden. Neben der Masse-, Clock- und Datenleitung wird eine Versorgungsspannung übermittelt. Dabei wird die Zwischenspannung von 3,9 V übertragen. Diese wird dann auf anzuschließenden Sensorplatinen lokal auf 3,3 V transformiert, um nah am Sensorchip eine möglichst störungsfreie Versorgungsspannung zu generieren. Die beiden Anschlüsse werden als I²C1 (oben) und I²C2 (unten) bezeichnet und sind unabhängig voneinander. An jeden der Anschlüsse können mehrere I²C-Chips angeschlossen werden, sofern diese unterschiedliche I²C-Adressen haben.

Als Sensorik befindet sich ein 3-Achs-Beschleunigungssensor (MMA7455L) auf dem AccelBoard3D, der diesem auch den Namen gab. Außerdem ist es möglich, die Versorgungsspannung und für jeden DoubleStrand den dort fließenden Strom zu messen (20-mΩ-Strommesswiderstand an Strommessverstärker LMP8601).

3.6.2 Speicherbereiche und Bootloader

Der auf dem AccelBoard3D verwendete Prozessor (STM32F103) hat 128 kB internen Flash-Speicher, der in 1 kB großen Blöcken separat gelöscht werden kann (Adressbereich 0x08000000 bis 0x0801FFFF). Dieser Speicher ist frei verwendbar und wurde beim Roboter Myon in vier separate Blöcke aufgeteilt (siehe Abbildung 23).

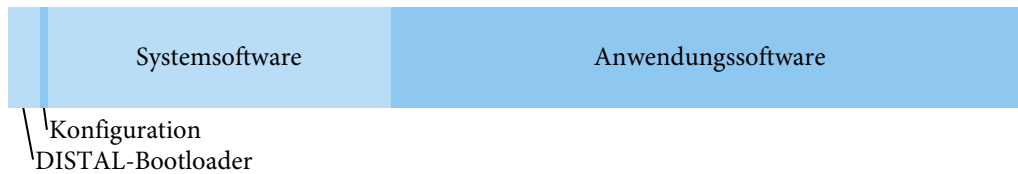


Abbildung 23: Speicheraufteilung des Nutzer-Flash-Speichers des Prozessors auf dem AccelBoard3D: Dem DISTAL-Bootloader (4 kB) folgt ein Konfigurationsbereich (1 kB), die Systemsoftware (43 kB) und ein Bereich für Programmcode für die sensomotorische Steuerung (Anwendungssoftware; 80 kB).

Wie im vorhergehenden Kapitel beschrieben, ist das Flashen mittels des im Prozessor integrierten Bootloaders eines AccelBoard3D im eingebauten Zustand nicht komfortabel möglich. Daher wurde ein separater Bootloader entwickelt, der die grundlegendsten Funktionen umfasst und es ermöglicht, ein AccelBoard3D über den gemeinsamen SpinalCord-Bus zu flashen. Dies betrifft alle Bereiche außer diesen DISTAL-Bootloader selbst. Für diesen sind die ersten 4 kB des Speichers vorgesehen.

Das fünfte kB des Speichers umfasst Konfigurations- und Kalibrierungsdaten (Adressbereich 0x08001000 bis 0x080013FF). Das erste Byte dieses Speicherbereichs enthält dabei die eindeutige Board-ID entsprechend der SpinalCord-Datenstruktur. Diese ID wird auch vom Bootloader verwendet um ein Board am Bus eindeutig adressieren zu können.

Ab Adresse 0x08001400 folgt die Systemsoftware, für die 43 kB Speicher reserviert sind. Die restlichen 80 kB sind für sich häufiger ändernden Programmcode vorgesehen (die Anwendungssoftware), der üblicherweise vom PC-Programm BrainDesigner aus neuronalen Netzen erzeugt wird (siehe Kapitel 4). Es ist aber auch möglich, eigenen Code, der beispielsweise in C geschrieben wurde, für den Prozessor zu kompilieren und in diesen Speicherbereich zu flashen.

Alle Speicherbereiche außer dem DISTAL-Bootloader selbst können dabei von Flash-Funktionen des DISTAL-Bootloaders umgeflasht werden. So sind über die gleichen Routinen Änderungen an der Konfiguration, der System- und der Anwendungssoftware möglich.

Der DISTAL-Bootloader besteht nur aus den notwendigsten Funktionen und muss normalerweise nicht aktualisiert werden. Wird dennoch eine neue Version

benötigt, kann auch dies über den Bus passieren, indem eine Bootloader-Kopie als Systemsoftware geflasht und dort ausgeführt wird. Mittels dieser Kopie kann dann auch der Bootloader-Speicherbereich neu geflasht werden.

Der Prozessor startet immer mit dem DISTAL-Bootloader. Passiert nach dem Prozessorstart 100 ms nichts und ist eine Systemsoftware vorhanden, schaltet der Bootloader auf diese um. Wenn etwas umgeflasht werden muss, kann die Systemsoftware auf Anforderung von außen (per Transparent Command) zurück zum DISTAL-Bootloader schalten.

Sollte eine fehlerhafte Systemsoftware im Flash-Speicher sein, kann dem DISTAL-Bootloader während der 100 Millisekunden nach dem Start mitgeteilt werden, diese nicht zu starten um ein funktionierendes Programm aufspielen zu können. Um dies zu erreichen, muss der Prozessor während der 100 ms über die serielle Schnittstelle mindestens 26 Bytes des Werts 170 (0xAA) empfangen haben, bei maximal einem Byte mit einem anderen Wert. Praktisch wird dies realisiert indem ein angeschlossener PC dauerhaft 0xAA sendet und der Prozessor dann neu gestartet wird.

Vom DISTAL-Bootloader wird die grundlegende Peripherie bereits vorbereitet. Die Prozessorpins werden entsprechend ihrer Funktion konfiguriert, der externe Quarz aktiviert und der Prozessor auf 72 MHz getaktet. Zusätzlich wird die serielle Schnittstelle USART1 für den SpinalCord-Bus in Betrieb genommen. Funktionen für den SpinalCord-Bus und zum Flashen werden vom DISTAL-Bootloader der Systemsoftware zur Verfügung gestellt.

Ist der DISTAL-Bootloader in Betrieb kann über ein einfaches Protokoll mit diesem kommuniziert werden. Neben Befehlen zum Abfragen der Board-ID gibt es Befehle zum Lesen, Löschen und Schreiben von Speicherbereichen, die sich jeweils auf ein Board mit einer spezifischen Board-ID beziehen und auch Statusbytes als Antwort senden. Lese- und Schreibbefehle arbeiten immer mit Blöcken aus 64 Bytes, wobei die Schreiboperation mittels Checksumme gesichert ist. Ein weiterer Befehl startet die Systemsoftware und verlässt den Bootloader. Das gesamte Protokoll zur Kommunikation mit dem Bootloader ist in Anhang 3 beschrieben.

Zum Ändern der ID und anderer Konfigurationswerte sowie zum Flashen der Systemsoftware wurde das Desktop-Programm AB3DConfig entwickelt. Es ist in Abbildung 24 dargestellt.

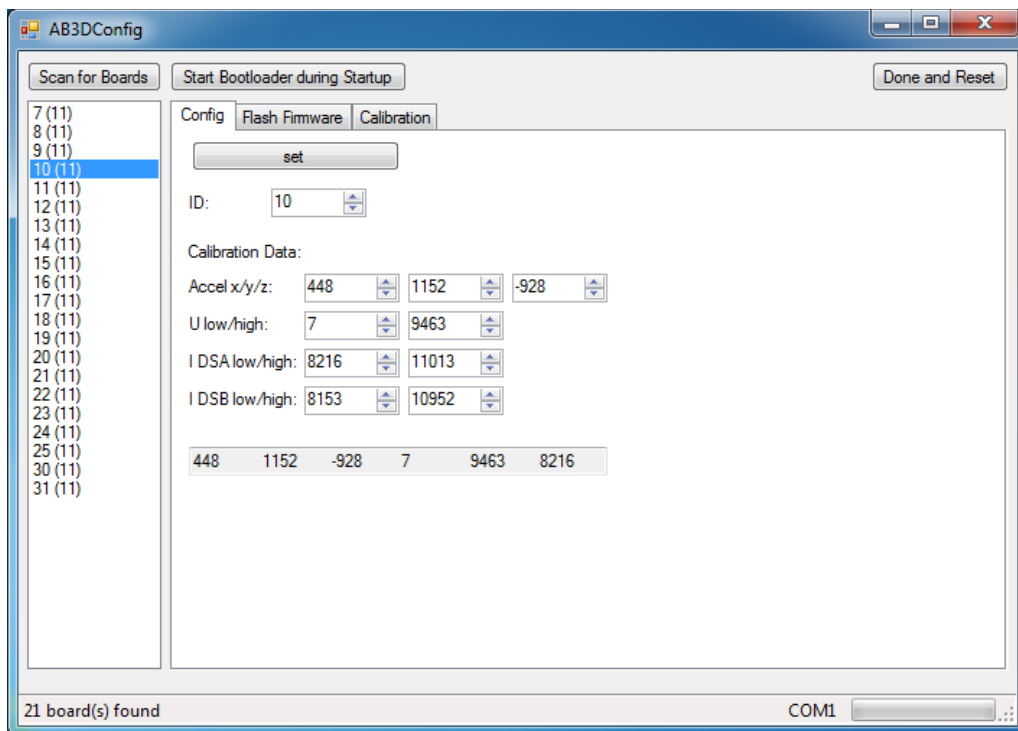


Abbildung 24: Screenshot des Desktop-Programms AB3DConfig zum Schreiben von Konfigurationswerten und zum Flashen der Systemsoftware.

Die Software benötigt das .NET-Framework und erkennt beim Start eine angeschlossene Matchbox 2.0 automatisch. Ein Klick auf „Scan for Boards“ sendet das Transparent Command 0x01, das alle angeschlossenen Boards in den DISTAL-Bootloader umschaltet. Es wird abgefragt, welche Boards vorhanden sind und zusätzlich aus dem Peephole-Wert die Version der Systemsoftware bestimmt. Beides wird in der Liste im linken Fensterbereich angezeigt. Ist keine korrekte Systemsoftware vorhanden, kann mit dem Button „Start Bootloader during Startup“ das dauerhafte Senden von 0xAA gestartet werden. Die angeschlossenen Boards müssen dann gestartet werden und gehen automatisch in den DISTAL-Bootloader-Modus.

Für ein in der linken Liste markiertes Board können über die Registerkarten „Config“ und „Calibration“ Konfigurations- und Kalibrierungswerte gesetzt

werden, unter anderem kann so die Board-ID geändert werden. Über die Registerkarte „Flash Firmware“ kann eine neue Systemsoftware aufgespielt werden, wobei für diesen Schritt mehrere Boards aus der Liste gewählt werden können.

3.6.3 AccelBoard3D-Systemsoftware

Die Systemsoftware ist die eigentliche AccelBoard3D-Implementierung der DISTAL-Architektur. Nach dem DISTAL-Bootloader wird die Systemsoftware gestartet. Diese fragt die Sensordaten ab, steuert die Motoren an und kommuniziert über den SpinalCord-Bus. Ebenso kann von ihr Deployment-Code gestartet werden.

Beim Start der Systemsoftware werden verschiedene Konfigurationen vorgenommen. Anschließend läuft das Programm so lange in einer Schleife, bis die Variable `programRunning` auf 0 gesetzt wird. Dies geschieht ausschließlich durch Empfang des Transparent Commands 0x0001, was dazu führt, dass die Systemsoftware verlassen und zum DISTAL-Bootloader zurückgekehrt wird.

Innerhalb der `programRunning`-Schleife wird zunächst die SpinalCord-Synchronisierung durchgeführt wie in Kapitel 3.4.5 „Synchronisierung und Laufzeit-Metamorphose“ beschrieben. Solange das Board synchronisiert ist, bleibt die Variable `synced` auf 1 und die Hauptschleife läuft endlos. Die Synchronisierung kann beispielsweise verloren gehen, wenn zur falschen Zeit innerhalb des SpinalCord-Zeitfensters ein Paket eines anderen Boards empfangen wird.

In der Hauptschleife wird die Variable `fs` (Frame State) abgefragt, die von einem Timer innerhalb des 10-ms-Zeitfensters zu definierten Zeitpunkten von 0 bis 3 hochgezählt wird. Diese Zeitpunkte entsprechen weitgehend dem SpinalCord-Timing wie in Kapitel 3.5.3 beschrieben und sind in Tabelle 7 dargestellt.

Ändert sich die Zustandsvariable `fs` startet ein neuer Zeitabschnitt und entsprechender Code wird ausgeführt. Bei `fs = 0` werden ausschließlich die LEDs gesetzt, der Prozessor wartet während des SpinalCord-Empfangs.

Bei `fs = 1` wird der Empfang von Transparent-Mode-Daten vorbereitet und das Senden von Ansteuerungswerten an die Motoren gestartet. Anschließend wird geprüft, ob der Taster des Boards gedrückt ist und entsprechend reagiert. Dann wird der Programmcode des sensomotorischen Regelungsprogramms gestartet, wenn die Ausführung gewünscht ist.

fs	Zeitpunkt	Beschreibung
0	0,00 ms	Start der SpinalCord-Kommunikation.
1	3,36 ms	Ende der SpinalCord-Kommunikation.
2	8,80 ms	Beginn der Sensorverarbeitung, Abschnitt 1.
3	9,40 ms	Mitte der Sensorverarbeitung, Beginn des Abschnitts Sensorverarbeitung 2.

Tabelle 7: Beschreibung der Zustände der internen Variable fs der Systemsoftware.

Die Sensorverarbeitung ist in zwei Abschnitte aufgeteilt. Die meisten Sensoren werden direkt abgefragt und liefern sofort die Werte. Die angeschlossenen Motoren vom Typ Dynamixel RX-28 antworten aber erst eine gewisse Zeit nach dem Senden des Anfragepakets. Motoren, die an unterschiedlichen DoubleStrands angeschlossen sind, können parallel abgefragt werden. Da aber auch bis zu zwei Motoren an einen DoubleStrand angeschlossen sein können, müssen diese hintereinander abgefragt werden. Beide Anfragen werden zu Beginn des entsprechenden fs-Fensters versendet und die Motoren haben daraufhin bis zu 600 μ s Zeit zu antworten.

Tabelle 8 zeigt stichpunktartig, was in den vier fs-Fenstern ausgeführt wird. Aus ihr ist auch herauszulesen, welche Sensorik abgefragt wird.

Der Empfang und das Senden der SpinalCord-Daten geschieht interruptgesteuert, um das korrekte Timing zu garantieren. Das Senden und Empfangen der Daten zu und von den Motoren wird ebenfalls interruptgesteuert durchgeführt, um während dieser Zeit auch andere Funktionen ausführen zu können.

fs	Beschreibung
0	LEDs setzen
1	LEDs setzen Vorbereitung Empfang Transparent-Mode-Daten Start des Sendens der Motor-Ansteuerungsdaten Abfrage des Tasters und Reaktion auf Taster-Druck Start des sensomotorischen Programmcodes
2	LEDs setzen Ersten Motor jedes DoubleStrands abfragen Behandeln von empfangenen Transparent-Mode-Daten Vorbereitung Datenpakete zur Ansteuerung der Motoren für nächsten Zeitslot Abfrage des Beschleunigungssensor Abfrage der externen Winkelsensoren (I ² C)
3	Zweiten Motor jedes DoubleStrands abfragen Abfrage der externen Drucksensoren (I ² C) ADC-Messungen Ströme und Spannung Peephole-Wert für nächstes SpinalCord-Senden aktualisieren

Tabelle 8: Stichpunktartige Beschreibung des Verhaltens während der Hauptschleife der Systemsoftware des AccelBoard3D-Prozessors.

Über die zwei LEDs kann jederzeit einfach der Zustand des Boards erfasst werden. Im Normalbetrieb leuchtet die blaue LED während der SpinalCord-Kommunikation. Da dies rund ein Drittel der Gesamtzeit ausmacht, glimmt die LED leicht blau. Wurden in vorherigen Zeitschritten Daten per Transparent-Mode-Motorsteuerung entgegengenommen und diese bleiben unerwartet aus, leuchtet die blaue LED weiter bis zum Beginn der Sensorverarbeitung (also 88 % und somit auffällig blau). Dieser Fehlerzustand kann also auf einen Blick erfasst werden.

Die weiße LED leuchtet nach dem Start zunächst nicht. Vom BrainDesigner erstellte Anwendungssoftware schaltet am Anfang der Berechnung die weiße LED an, sie wird dann von der Systemsoftware zu Beginn der nächsten SpinalCord-Kommunikation wieder abgeschaltet. Anhand der weißen LED kann also abgelesen werden, ob derzeit ein sensomotorisches Programm ausgeführt wird.

3.7 Implementierung auf dem BrainModule

Das BrainModule ist eine im Labor für Neurorobotik entwickelte Platine, die im Kopf des Roboters Myon verbaut ist. Auf dieser befindet sich ein FPGA (Field Programmable Gate Array), an das Kamera und Mikrofone angeschlossen sind. Das BrainModule hat innerhalb des SpinalCord-Protokolls die ID 0.

3.7.1 Aufbau des BrainModules

Das BrainModule verarbeitet visuelle und auditive Signale. Herzstück ist ein FPGA, ein Hardwarebaustein, dessen innere Struktur konfiguriert werden kann. Dies betrifft die logischen Funktionen einzelner Zellen und deren Verdrahtung untereinander. Beschrieben wird diese Struktur mit einer Hardwarebeschreibungssprache wie VHDL oder Verilog. Bild- und Audioverarbeitung sind somit in Echtzeit möglich.

Das BrainModule nimmt ein wechselbares, 55×24 mm großes, FPGA-Modul des Herstellers EBus auf, ein HydraXC 30. Dieses hat neben einem FPGA Virtex-4 XC4VLC25 von Xilinx bereits einen 12-MHz-Oszillator, Flash- und SDRAM-Speicher und einen Slot für MiniSD-Karten. Beim Start liest ein 8-Bit-Prozessor eine Datei („Bitfile“) von einer eingesteckten SD-Karte und konfiguriert mit dieser das FPGA. Die SD-Karte kann nach dem Startprozess auch frei als Datenspeicher genutzt werden.

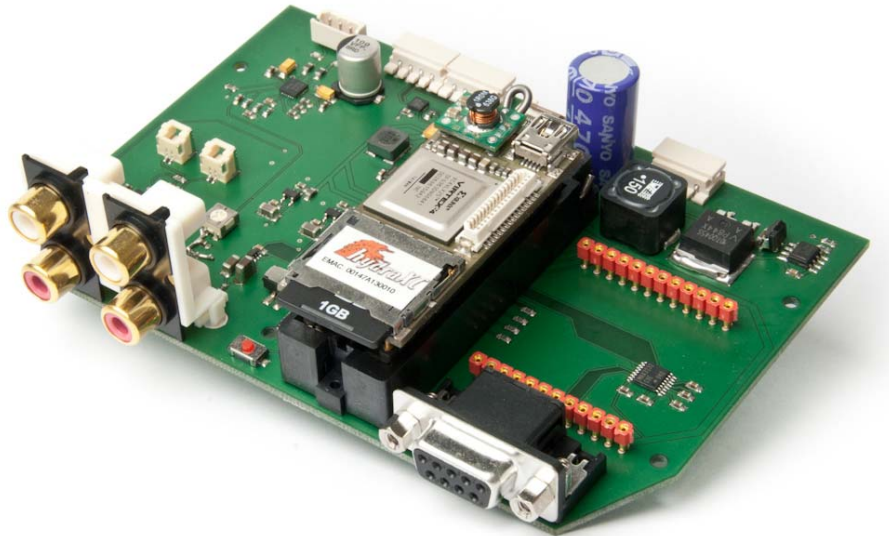


Abbildung 25: Das BrainModule mit eingestecktem HydraXC-Modul mit SD-Karte.

Abbildung 25 zeigt ein BrainModule, Abbildung 26 eine schematische Übersicht der Elemente. Zusätzlich zum HydraXC-Modul befindet sich ein RS-485-Transceiver für den Anschluss des SpinalCords auf dem Board und ein RS-232-Transceiver zur Kommunikation mit einem PC. Außerdem kann ein WLAN-Modul des Herstellers Avisaro (W20511) eingesteckt werden, das eine einfache WLAN-Verbindung ermöglicht. Das WLAN-Modul kommuniziert mit dem FPGA per SPI.

Ein Video-Mixer ermöglicht es, das analoge Videosignal (CVBS) einer angeschlossenen Kamera zu bestimmten Zeiten auszuschalten bzw. über verschiedene Widerstände hochzuziehen. Somit können einzelne Pixel in schwarz oder Graustufen bis weiß dargestellt werden. Bleibt dabei das originale Kamerabild angeschaltet, werden einzelne Pixel hochgezogen und es ergibt sich ein weißer Overlay, bei dem das Originalbild durchscheint. Dies kann sowohl für grafische Ausgaben als auch für Textausgaben genutzt werden.

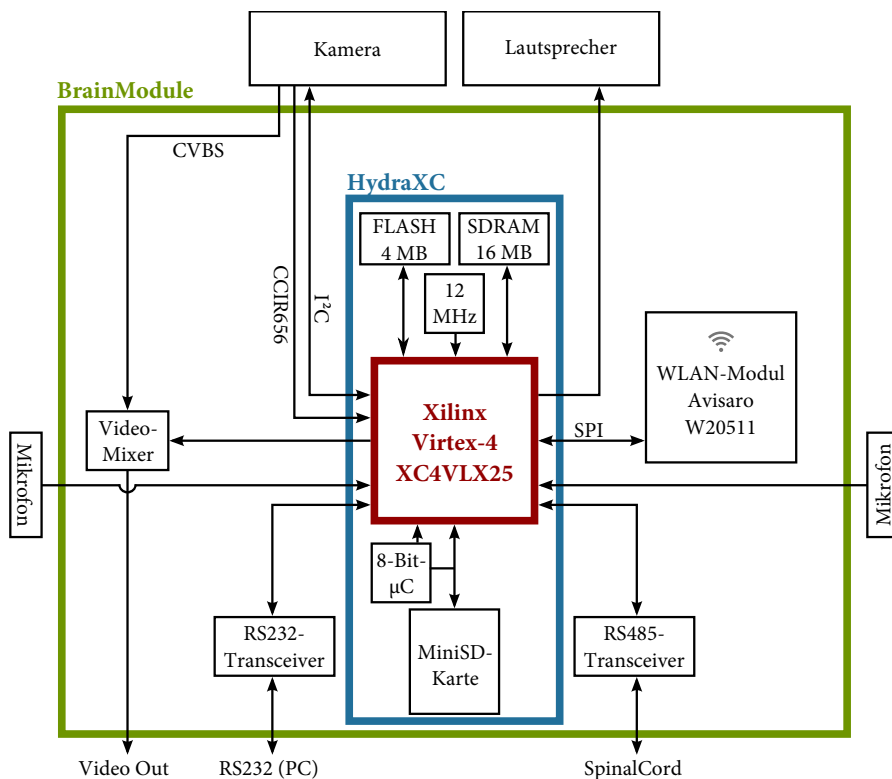


Abbildung 26: Schematische Darstellung des BrainModules (grün) mit angeschlossener Kamera, Lautsprecher und Mikrofonen. Blau dargestellt ist das aufgesteckte HydraXC-Modul mit dem Xilinx-FPGA (rot).

Das Kamerabild wird außerdem über ein digitales 8-Bit-Signal plus Clock-Leitung in das FPGA geleitet (digitales Format: CCIR656). Im Roboter Myon wird die Kamera 21K155DIG von Videology verwendet, eine PAL-Farbkamera mit 752×582 Pixeln. Diese ist zusätzlich per I²C an das FPGA angeschlossen und kann darüber konfiguriert werden (beispielsweise Verschlusszeit und Weißabgleich).

Für die Audio-Ein- und -Ausgabe werden zwei Mikrofone (digitale MEMS-Mikrofone MP45DT02) und ein Lautsprecher (über Audioverstärker LM4950) angeschlossen.

3.7.2 Struktur der FPGA-Konfiguration

Im Roboter Myon wird eine FPGA-Konfiguration verwendet, die auch zwei 32-Bit-Prozessoren vom Typ MicroBlaze umfasst. Einer ist zur Verarbeitung visueller Signale (Video-Prozessor), der andere zur Verarbeitung auditiver Signale (Audio-Prozessor) vorgesehen. Des Weiteren gibt es spezielle Blöcke, die in Verilog implementiert wurden. Beispielsweise wurde ein PDM-Generator zur Ansteuerung eines Klasse-D-Verstärkers für den Lautsprecher entwickelt und ein Block zur Bild-Vorverarbeitung.

Eine weitere spezielle in Verilog implementierte Einheit kümmert sich um die Synchronisierung und die Entgegennahme der SpinalCord-Daten (SpinalCord-Interface). Die SpinalCord-Daten werden in einem Dual-Port-RAM von 2 kB Größe gespeichert, über dessen zweiten Port auch der Video-Prozessor auf die Daten zugreifen kann. Die selbst vom BrainModule zu sendenden Daten (ID 0; SpinalCord-Felder 0 bis 26) können auch direkt vom Prozessor in diesen RAM geschrieben werden, von wo aus sie das SpinalCord-Interface versendet. Jeweils zum Ende der Kommunikationszeit (nach 3,36 ms des 10-ms-SpinalCord-Zeitfensters) wird im Video-Prozessor ein Interrupt generiert.

Das Verilog-Modul SpinalCordInterface nutzt die Module SC_ReadByte (zum Lesen eines Bytes) und SC_WriteByte (zum Schreiben eines Bytes). Die Implementierung hat die Einschränkung, dass erst gesendet wird, wenn Daten eines anderen SpinalCord-Teilnehmers empfangen wurde. Dies ist jedoch im Gesamtsystem keine Einschränkung, da die AccelBoard3Ds anfangen zu senden, bevor das FPGA von der SD-Karte konfiguriert wurde.

4 Software BrainDesigner

Die DISTAL-Architektur ist für große neuronale Netze ausgelegt. Zur einfachen Erstellung solcher Netze wurde das Programm BrainDesigner als Windows-Anwendung in C# entwickelt. Mit dem BrainDesigner können nicht nur neuronale Netze erstellt werden, sondern auch eine Vielzahl datenflussorientierter Regelschleifen. Erstellte Netze können als Funktionsblock gekapselt und hierarchisch in andere Netze eingebettet werden. Dies erhöht in großen Netzen die Übersicht.

In diesem Kapitel wird zunächst die Bedienung der Software BrainDesigner erläutert, um anschaulich die zum Verständnis notwendigen Begriffe einzuführen. Anschließend wird auf interne Abläufe der Software eingegangen. Der BrainDesigner ist über Plugins erweiterbar und somit für viele Zwecke geeignet, die über die hier vorgestellten Anwendungen hinausgehen. Anschließend wird der neuronale Bytecode erläutert, aus dem sich ein Regelungsprogramm zusammensetzt. Zuletzt wird die Verwendung mit dem Roboter Myon vorgestellt. Hierbei liegt der Schwerpunkt auf der Kompilierung der Netze für die DISTAL-Architektur.

4.1 Erstellung von Regelschleifen mit der Software BrainDesigner

Mit der Software BrainDesigner ist es möglich, datenflussorientierte Regelschleifen grafisch zu erstellen. Vorrangig wurde sie zur Erstellung künstlicher neuronaler Netze entwickelt. Hinter allen Grundelementen, wie Neuronen und Synapsen, steht Programmcode, der sogenannte neuronale Bytecode. Er ermöglicht es, verschiedene Berechnungsvorschriften und Lernverfahren zu implementieren. Auf den neuronalen Bytecode wird in Kapitel 4.3 detailliert eingegangen. Eigene Grundelemente können erstellt und der Funktionsumfang so von jedem Nutzer erweitert werden. Grafisch zusammengestellte Netze können wiederum in weitere Netze eingebunden werden, wodurch Netz-Hierarchien entstehen.

Als Beispiel zur Beschreibung des BrainDesigners dient ein neuronales Monoflop, wie es in [Hil07] beschrieben wurde. Ein neuronales Monoflop besteht

aus zwei Neuronen und dient der einfachen Übersetzung von Keyframe-Netzen⁶ in neuronale Strukturen. Das verwendete Neuronenmodell entspricht dem Tanh-Neuronenmodell, das in Kapitel 1.4 vorgestellt wurde. Nachdem ein Eingangssignal (Trigger) von einem hohen Pegel (nahe +1) auf einen niedrigen Pegel (nahe -1) fällt, liefert die Struktur am Ausgang eine definierte, einstellbare Zeit lang ein hohes Signal (+1), bevor es wiederum selbst Richtung -1 zurückfällt. Abbildung 27 zeigt das Zwei-Neuronen-Netz, Abbildung 28 das idealisierte Verhalten.

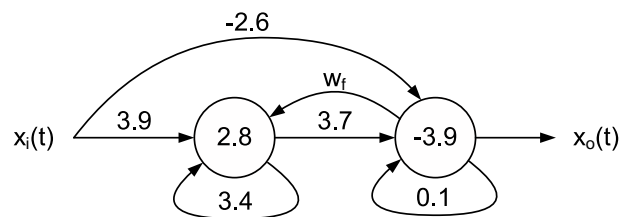


Abbildung 27: Neuronales Monoflop wie in [Hil07] dargestellt. Über das Gewicht w_f lässt sich die Zeitkonstante einstellen. Die Zahlen innerhalb der Neuronen bezeichnen Bias-Werte.

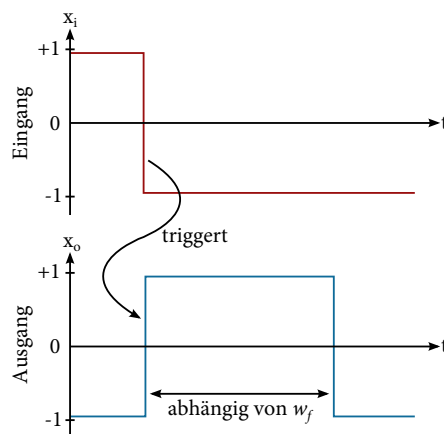


Abbildung 28: Idealisiertes Verhalten eines Monoflops: Eingang x_i oben, Ausgang x_o unten.

⁶ Keyframe-Netze sind eine weitverbreitete Technik zur Ansteuerung von Bewegungssequenzen. Ein Keyframe entspricht dabei einer bestimmten Körperpose. Interpoliert man zwischen Keyframes, bewegt sich der Roboter.

Das neuronale Monoflop wird im BrainDesigner als *Structure* implementiert. Eine Structure ist ein grafisches Netz, das aus Knoten und Kanten besteht. Knoten können zum einen *Units* sein, Berechnungsmodule hinter denen neuronaler Bytecode steht. Zum anderen können Knoten auch andere Structures sein. Auch Ein- oder Ausgänge stellen Knoten des Netzes dar, für sie gibt es spezielle Systemmodule (*Input* und *Output*). Kanten im Netz sind gerichtet, hinter ihnen liegt immer neuronaler Bytecode, sie werden im BrainDesigner *Synapse* genannt. Abbildung 29 zeigt ein einfaches Netz mit verschiedenen Knoten-Typen.

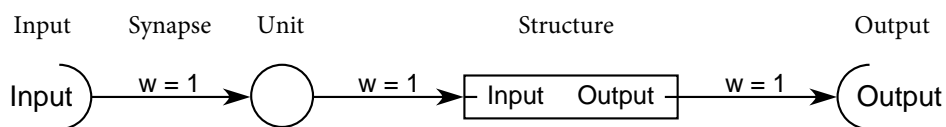


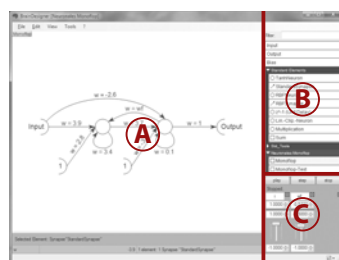
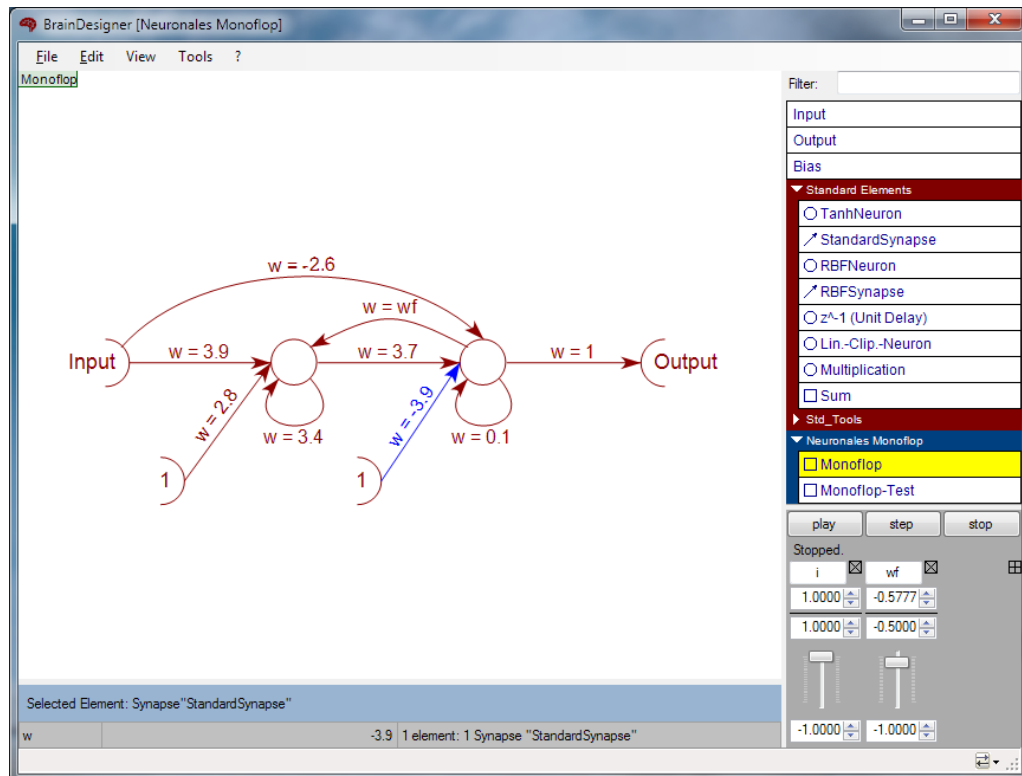
Abbildung 29: Beispiel-Netz mit Knoten-Elementen (Input, Unit, Structure, Output) und Kanten-Elementen (Synapse).

Häufig verwendete Units und Synapsen werden in *Bibliotheken* bereitgestellt. In der Standardbibliothek befinden sich unter anderem die für das Monoflop nötigen Module *Tanh-Neuron* (Unit) und *Standard-Synapse* (Synapse). Die Unit *Tanh-Neuron* ist ein Neuron mit der Transferfunktion Tangens Hyperbolicus, eine Standard-Synapse multipliziert ihren Eingangswert mit einem Parameter w . Zusammen ergeben sie das *Tanh-Neuronenmodell*. Bias-Werte – also feste Offsets, die in jedem Zeitschritt auf die Aktivierung eines Neurons addiert werden – sind im Modul *Tanh-Neuron* nicht vorgesehen. Es gibt jedoch ein Systemmodul *Bias*, das immer den festen Wert $+1$ liefert. Wird eine Standard-Synapse hinzugefügt, die ein *Bias*-Modul mit einem Neuron verbindet, entspricht dies einem Bias-Wert in Höhe des Gewichts der Synapse.

Verwendet werden Bibliotheksmodule in *Projekten*. In einer Structure eines Projekts können (neben den Modulen des Projekts selbst) Module aus allen geladenen Bibliotheken verwendet werden. Zusätzlich ist es möglich, Module aus anderen Projekten zu importieren, diese werden dann als Kopie in das aktuelle Projekt abgelegt.

Abbildung 30 zeigt einen Screenshot des BrainDesigners mit der Monoflop-Structure. Das Fenster teilt sich in drei Bereiche: Der größte Bereich links ist der eigentliche Bearbeitungsbereich, in dem die grafischen Netze erstellt werden.

Rechts oben befindet sich der Bibliotheks- und Projektbereich, darunter (beginnend mit den Schaltflächen „play“, „step“ und „stop“) das Cockpit.



- A** Bearbeitungsbereich
- B** Bibliotheks- und Projektbereich
- C** Cockpit

Abbildung 30: Oben: Screenshot der Software BrainDesigner mit geöffneter Monoflop-Structure. Unten: Die drei Programm Bereiche.

Der Bibliotheks- und Projektbereich zeigt alle geladenen Bibliotheken und Projekte an. Vorhandene Projekte und Bibliotheken bzw. neue Projekte können über das Menü „File“ hinzugefügt werden. Bibliotheken haben üblicherweise die

Dateiendung *bdl* (BrainDesigner Library), Projekte *bdp* (BrainDesigner Project). Intern ist das Dateiformat identisch, es basiert auf XML.

Im Bibliotheks- und Projektbereich wird jede Bibliothek und jedes Projekt als farbiger Balken dargestellt. Bibliotheken sind rot, das aktive Projekt blau und inaktive Projekte grau. Mit einem Klick auf diesen Balken kann die Anzeige der enthaltenen Module ein- und ausgeklappt werden. Über ein Kontextmenü ist es möglich, ein Projekt in eine Bibliothek umzuwandeln und umgekehrt. Eine Bibliothek bzw. ein Projekt kann auch gesperrt werden, sodass keine Änderungen möglich sind – diese Option ist vor allem bei Bibliotheken sinnvoll.

Innerhalb einer Bibliothek oder eines Projekts können sich *Units*, *Synapsen* oder *Structures* befinden. Units werden mit einem Kreis vor dem Namen dargestellt, Synapsen mit einem Pfeil und Structures mit einem Quadrat. Zusätzlich gibt es ganz oben die drei Systemmodule „Input“, „Output“ und „Bias“. Das aktuell im Bearbeitungsbereich geöffnete Modul wird gelb unterlegt dargestellt, ein verändertes und noch nicht gespeichertes Modul wird mit einem roten Quadrat gekennzeichnet. Module können innerhalb des Bibliotheks- und Projektbereichs per Drag & Drop verschoben werden.

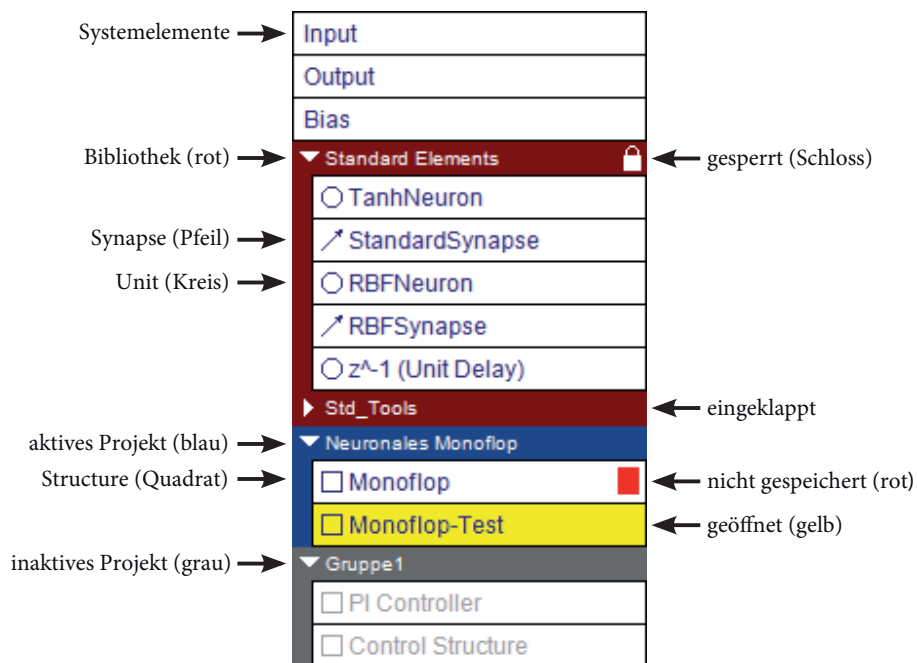


Abbildung 31: Der Bibliotheks- und Projektbereich.

Abbildung 31 zeigt beispielhaft verschiedene Bibliotheken und Projekte, wie sie innerhalb des Bibliotheks- und Projektbereichs dargestellt werden.

Bei Rechtsklick auf einen Bibliotheks- oder Projektbalken öffnet sich ein Kontextmenü, das es erlaubt, eine neue Structure, Unit oder Synapse zu diesem Projekt hinzuzufügen. Per Rechtsklick auf ein Modul kann ebenfalls ein Kontextmenü aufgerufen werden, das es ermöglicht, ein geändertes Modul zu speichern, es umzubenennen, zu löschen oder sich ein Fenster mit weiteren Informationen zum Modul anzeigen zu lassen. Ein Doppelklick auf ein Modul öffnet dieses im Bearbeitungsbereich.

Der Bearbeitungsbereich ist der größte Bereich und befindet sich im Hauptfenster links. Bei geöffneten Units und Synapsen wird dort ein Quellcode-Editor für den neuronalen Bytecode angezeigt, bei Structures ein grafischer Editor.

```

1 300:
2  mul    V0, Input, b
3  tanh   V0, V0
4  load   V1, 1
5  sub    V0, V1, V0
6  write  Output, V0
7

```

Inputs		Parameters			Internals		Outputs	
No.	Name	No.	Name	Standard	No.	Name	No.	Name
		1	b	1				




Abbildung 32: Der Quellcode-Editor. Oben wird der neuronale Bytecode eingegeben, darunter können für die Unit Inputs, Parameter, interne Werte und Outputs angegeben werden. Ganz unten wird eine Vorschau des Elements angezeigt. Diese Unit hat einen Parameter b mit dem Standardwert 1. Units können bei der grafischen Anzeige mit einem Bild hinterlegt werden.

Im Quellcode-Editor (Abbildung 32) kann neuronaler Bytecode, der hinter jeder Unit und jeder Synapse steht, bearbeitet werden. Der Editor zeigt Zeilennummern an und beherrscht Syntax-Highlighting. Der geöffnete Quellcode kann über das Hauptmenü („File“ → „Export“ → „ByteCode“) als Textdatei oder RTF-Datei mit Syntax-Highlighting exportiert werden. Unterhalb des Quellcode-Editors gibt es einen Bereich zur Festlegung der Ein- und Ausgänge und internen Zustände des Moduls sowie eine grafische Vorschau.

Der grafische Editor wird zur Bearbeitung von Structures angezeigt. Eine neue Structure hat zunächst einen leeren Zeichenbereich. Neue Instanzen von Modulen fügt man per Drag & Drop aus dem Bibliotheks- und Projektbereich hinzu.

Mit einfachem Klick im grafischen Editor markiert man Modulinstanzen. Bei Klick mit gedrückter Steuerungstaste kann man weitere Modulinstanzen zur Auswahl hinzufügen. Per Klick auf einen freien Bereich und ziehen des Mauszeigers bei gedrückter Maustaste werden alle Modulinstanzen im markierten Bereich ausgewählt.

Modulinstanzen werden normalerweise in einem dunklen Braun dargestellt, markierte Modulinstanzen blau. Befindet sich eine Modulinstanz unterhalb des Mauszeigers, wird diese grün dargestellt, alle angeschlossenen Modulinstanzen orange. Ein- und Ausgänge von Units oder Structures, an die keine Synapse angeschlossen ist, werden hellrot dargestellt. Das gilt ebenso für Synapsen, die nicht sowohl an Ein- als auch Ausgang mit einer anderen Modulinstanz verbunden sind. Einzelne Modulinstanzen können deaktiviert werden, diese werden grau dargestellt. Tabelle 9 zeigt die Farben in der Übersicht. Abbildung 33 zeigt beispielhaft verschiedene Modulinstanzen in den verschiedenen Farbtönen. Im Folgenden werden in dieser Arbeit Netze schwarz-weiß dargestellt, die Farbmarkierungen sind als Hinweise während der Bearbeitung der Netze gedacht.

Im grafischen Editor kann gezoomt werden. Über die Tastenkombination Strg+Plus wird die Anzeige vergrößert, mit der Tastenkombination Strg+Minus verkleinert. Bei gedrückter Steuerungstaste kann auch mit dem Mousrad gezoomt werden. Erweiterte Zoom-Optionen sind im Menü unter „View“ → „Zoom“ zu finden.







Bedeutung	Farbe	
Standard	Braun	
Markiert	Blau	
Unter Mauszeiger	Grün	
Angeschlossen an Modulinstanz unter Mauszeiger	Orange	
Nicht angeschlossen	Hellrot	
Deaktiviert	Grau	

Tabelle 9: Farben innerhalb des grafischen Editors des BrainDesigners.

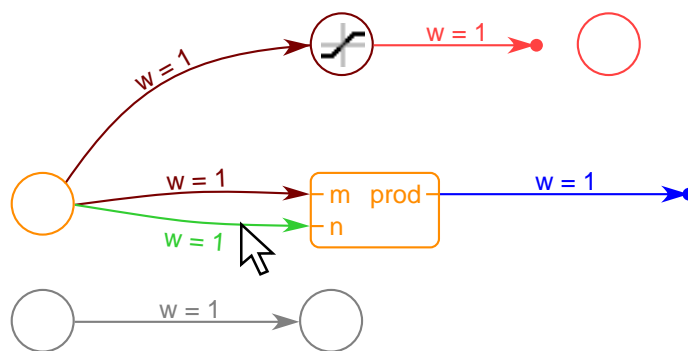


Abbildung 33: Verschiedene Farben der Modulinstanzen eines Netzes während der Bearbeitung im BrainDesigner.

Abbildung 34 zeigt verschiedene Modulinstanzen, wie sie im grafischen Editor dargestellt werden. Units, die genau einen Ein- und Ausgang haben, werden als Kreis dargestellt. Units mit benannten Ein- bzw. Ausgängen (bei mehr als einem Ein- oder Ausgang zwingend nötig) werden als Rechtecke mit abgerundeten Ecken dargestellt – die Ein- und Ausgänge werden in diesem Fall links bzw. rechts angezeigt. Structures werden als Rechtecke angezeigt, ihre Ein- und Ausgänge werden wie bei Units mit benannten Eingängen dargestellt.

Die Systemmodule „Input“, „Output“ und „Bias“ werden als Halbkreis dargestellt.

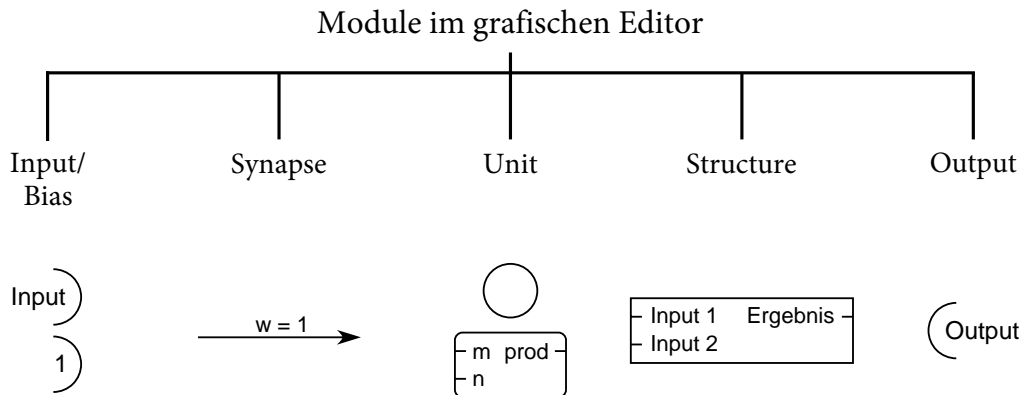


Abbildung 34: Die verschiedenen Modultypen und wie sie im grafischen Editor angezeigt werden.

Synapsen werden als Pfeile dargestellt, die der Datenflussrichtung entsprechen. An unangeschlossenen Enden wird ein kleiner Kreis angezeigt, der es erleichtern soll, solche unangeschlossenen Enden zu finden. Parameter einer Synapse werden am Pfeil dargestellt, die Anzeige kann global oder per Kontextmenü für jeden Parameter jeder Synapse separat gesteuert werden (es gibt Optionen zum dauerhaften Ausblenden beziehungsweise zum Ausblenden, wenn der Parameterwert dem Standard-Wert entspricht).

Synapsen können nicht als Ganzes verschoben werden, vielmehr werden der Start- und der Endpunkt separat verschoben. Schiebt man einen solchen Punkt in die Nähe eines Anschlusses einer Unit oder einer Structure, wird die Synapse verbunden. Wird eine Unit oder Structure mit angeschlossener Synapse verschoben, wird der entsprechende Punkt der Synapse mitverschoben.

Auch die Mitte einer Synapse ist sensitiv, die Synapse wird grün dargestellt, wenn man den Mauszeiger über die Mitte bewegt. Wird dieser mittlere Punkt angeklickt und bei gehaltener Maustaste verschoben, kann die Krümmung der Kurve verändert werden. Die Stärke der Krümmung (siehe Abbildung 35) kann beeinflusst werden, indem die Maus nach links oder rechts bewegt wird (relativ zum Start- und Endpunkt).

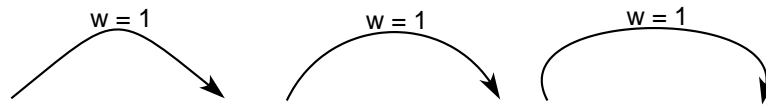


Abbildung 35: Synapse: Unterschiedlich starke Krümmungen.

Ein kleines i an einer Modulinstanz zeigt an, dass für diese ein Kommentar eingegeben wurde. Verweilt man mit dem Mauszeiger über einem solchen Modul wird ein Hinweisfeld mit dem Kommentar angezeigt (siehe Abbildung 36). Die Anzeige der Hinweis-Symbole kann mit der Taste F7 umgeschaltet werden. Mit der Taste F8 kann für alle Units und Structures und wahlweise auch die Synapsen der Name angezeigt werden. Der Name wird zusätzlich immer in der Statusleiste des BrainDesigners angezeigt, wenn sich der Mauszeiger über einem Element befindet.

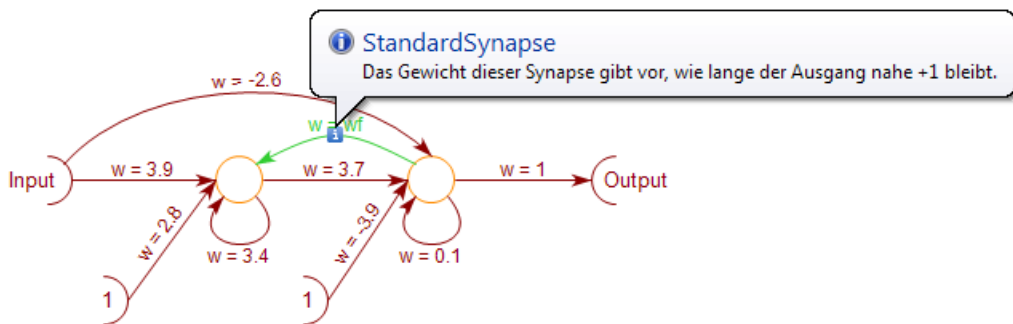


Abbildung 36: Ein kleines i kennzeichnet ein Modul, für das ein Kommentar hinterlegt ist. Verweilt man mit der Maus über dem Modul, wird der Kommentar angezeigt.

Die grafische Darstellung des gesamten Netzes kann als PNG-Rastergrafik oder SVG-Vektorgrafik exportiert werden (Hauptmenü „File“ → „Export“ → „Structure Graphics“). Für den SVG-Export wurde eine eigene einfache SVG-Bibliothek entwickelt.

Jede Modulinstanz kann Parameter haben. Werden im grafischen Editor Modulinstanzen selektiert, werden sämtliche möglichen Parameter in einem Bereich unterhalb der Zeichenfläche dargestellt und können eingestellt werden. Wurden

mehrere Modulinstanzen mit gleichen Parametern selektiert, werden eingegebene Werte für alle Instanzen übernommen.

Um das neuronale Monoflop zu implementieren müssen also folgende Schritte vorgenommen werden:

- Eine neue Projektdatei wird angelegt (Menü „File“ → „New Project...“).
- Mit dem Anlegen des Projekts wurde eine Structure automatisch hinzugefügt. Per Rechtsklick auf die Structure im Bibliotheks- und Projektbereich wird ein Kontextmenü aufgerufen, über das die Structure in „Monoflop“ umbenannt werden kann.
- Per Drag & Drop aus der Standardbibliothek werden zwei Instanzen des Moduls Tanh-Neuron und neun Instanzen des Moduls Standard-Synapse hinzugefügt. Zusätzlich werden ein Systemelement Input, ein Systemelement Output und zwei Systemelemente Bias hinzugefügt.
- Mit der Maus werden die Elemente angeordnet und die Synapsen entsprechend verbunden. Die Synapsen werden nacheinander markiert und jeweils der Parameter w gesetzt, für die parametrisierbare Synapse wird als Parameterwert „wf“ eingetragen.

Abbildung 37 zeigt das vollständige Netz, wie es im BrainDesigner dargestellt wird.

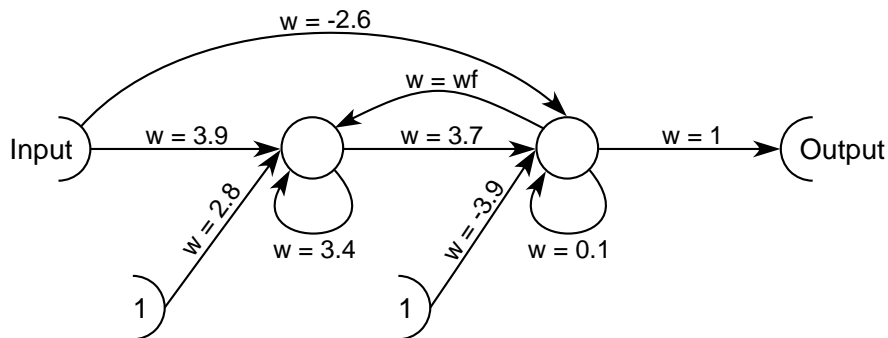


Abbildung 37: Das neuronale Monoflop im BrainDesigner.

Zum Testen des Monoflops wird nun eine weitere Structure im selben Projekt angelegt. In diese wird die soeben erstellte Monoflop-Structure per Drag & Drop aus dem Bibliotheks- und Projektbereich eingefügt. Um Test-Eingabewerte zu

erzeugen wird ein Bias mit einer Synapse an den Eingang des Monoflops angeschlossen. Der Parameter w der Synapse wird auf „ x “ gesetzt. Dieses x muss von $+1$ auf -1 fallen, damit das Monoflop aktiviert wird.

Zum einfachen Ändern von Werten während der Laufzeit können im Cockpit (rechts unten im BrainDesigner-Fenster) sogenannte „Slider“ hinzugefügt werden (siehe Abbildung 38). Jeder Slider hat einen Namen und einen Wert. Wird irgendwo im grafischen Netz für einen Parameter statt eines festen Werts ein Slider-Name eingetragen, wird der dort eingestellte Wert verwendet. Für den Slider werden ein Minimal- und ein Maximalwert eingestellt. Mittels eines Schiebers kann der aktuelle Wert dann leicht mit der Maus verändert werden.

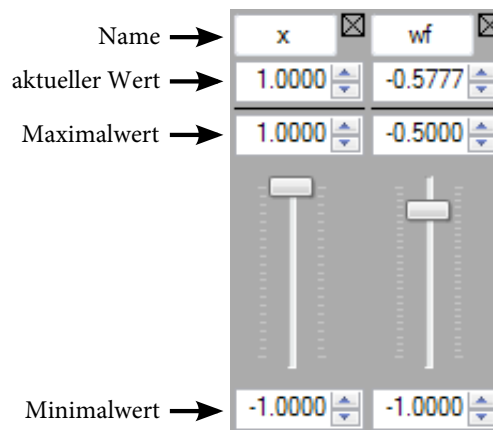


Abbildung 38: Beispiel-Slider im Cockpit für die Slider-Werte x und wf .

Um einzelne Werte eines Netzes über die Zeit inspizieren zu können, kann bei geöffnetem grafischem Editor das sogenannte „CurveDisplay“ eingeblendet werden (die Anzeige kann im Hauptmenü unter „View“ → „Show Curves“ aktiviert und deaktiviert werden). Die Werte werden auf der y -Achse dargestellt, die x -Achse ist die Zeitachse, wobei die neuesten Werte ganz rechts angezeigt werden und sich die Kurve nach links verschiebt. Der aktuelle Zahlenwert wird ganz rechts neben der Kurve ausgegeben. Aus Performancegründen ist das CurveDisplay nicht im Windows-Standard-Grafiksystem GDI+ sondern in DirectX implementiert.

Abbildung 39 zeigt ein Beispiel mit zwei Displays übereinander mit jeweils zwei Kurven. Links unten in jedem Display werden Legenden in Form von Quadraten

in der Farbe der Kurven angezeigt. Bewegt man den Mauszeiger über eine der Legenden wird die entsprechende Modulinstanz im grafischen Editor hervorgehoben. Mit einem Klick auf das Farbquadrat kann man die Farbe der Kurve ändern, mit einem Klick auf das X rechts daneben löscht man die Kurve.

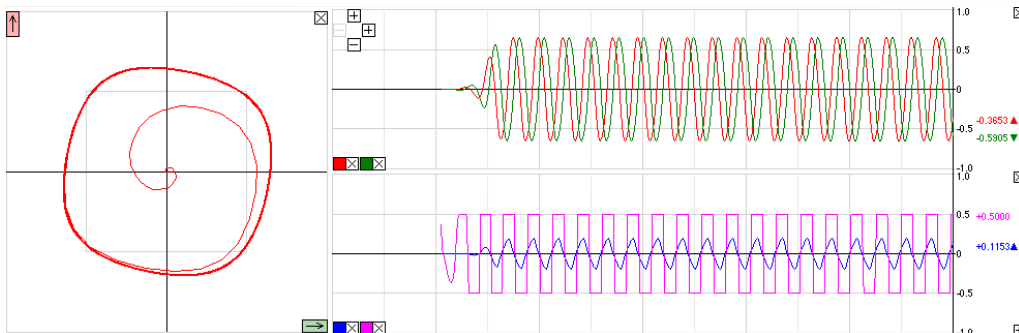


Abbildung 39: Beispiel des CurveDisplays mit zwei Anzeigen übereinander. Ganz rechts wird der aktuelle Wert angezeigt. Ein Pfeil kennzeichnet die Tendenz des Wertes. Eine vertikale graue Linie kennzeichnet 100 Zeitschritte, jeweils bei 50 Zeitschritten gibt es eine gepunktete graue Linie. Links sind die beiden oberen Kurven gegeneinander abgetragen.

Hinzugefügt werden Kurven mittels des Kontextmenüs im grafischen Editor. Wird mit der rechten Maustaste auf eine Modulinstanz geklickt erscheint ein Kontextmenü, in dem es die Möglichkeit gibt, Ausgänge als Kurven hinzuzufügen oder wieder zu entfernen.

Mit den Plus- und Minus-Schaltflächen links oben im CurveDisplay kann sowohl die x- als auch die y-Achse gezoomt werden. Mit der Plus-Schaltfläche ganz rechts unten können weitere Displays hinzugefügt werden, die dann unabhängig voneinander übereinander dargestellt werden. Per Drag & Drop mit dem zugehörigen Farbquadrat der Legende kann eine Kurve von einem Display in ein anderes gezogen werden.

Ganz links oben befindet sich eine weitere Schaltfläche mit einem kleinen Pfeil. Damit lässt sich ein 2D-Display ein- bzw. ausblenden, mit dem zwei Werte gegeneinander abgetragen werden können. Einer der Werte nimmt die x-Achse ein, der andere die y-Achse. Dargestellt werden die letzten 500 Zeitschritte. Per

Drag & Drop des zugehörigen Farbquadrats aus den normalen Displays lassen sich Kurven der x - bzw. y -Achse zuordnen.

Bei der Monoflop-Test-Structure werden nun per Kontextmenü der Ausgang der Synapse (und somit der Eingangswert des Monoflops) und der Ausgang des Monoflops zum CurveDisplay hinzugefügt. Der Cockpit-Slider wf kann mit dem Beispielwert $-0,5777$ belegt werden, x ist zunächst $+1$.

Nun kann die Berechnung des im Bearbeitungsbereich geöffneten grafischen Netzes gestartet werden. Zur Steuerung gibt es im oberen Teil des Cockpits Schaltflächen. Es gibt dabei mindestens die Schaltflächen „play“, „step“ und „stop“. Mit „play“ wird die Berechnung des Netzes gestartet, die Schaltfläche wird zu einer „pause“-Schaltfläche. Mit „step“ wird genau ein Zeitschritt berechnet, mit „stop“ die gesamte Berechnung abgebrochen. Unterhalb der Schaltflächen wird der aktuelle Status angezeigt.

Wird nun während der Berechnung des Monoflop-Testnetzes der Schieberegler x auf -1 gezogen, wird das Monoflop aktiviert und ist eine kurze Zeit nahe $+1$, bis es wieder auf -1 fällt. Dies kann im CurveDisplay beobachtet werden. Der Wert für wf kann nun einfach angepasst und das Verhalten des Monoflops untersucht werden.

In einem zweiten Schritt werden vier weitere Instanzen der Monoflop-Structure in die Test-Structure eingefügt, die alle mit einer Synapse mit dem Parameter x vom Bias-Element an den Eingang verknüpft werden (und somit alle den gleichen Eingangswert haben). Markiert man nun eine der Monoflop-Instanzen, wird im Parameterbereich der Parameter wf des Monoflops angezeigt. Die Vorbelegung des Parameters ist „wf“, weshalb überall der identische Wert des Sliders verwendet wird. Es ist jedoch möglich für jede Instanz einen separaten Wert für den Parameter festzulegen (beispielsweise $-0,9959$, $-0,6087$, $-0,5777$, $-0,5692$ und $-0,5657$). Abbildung 40 zeigt links die Teststruktur und rechts einen Ausschnitt des CurveDisplays mit dem Eingangswert (rot, obere Kurve) und den fünf unterschiedlich langen Reaktionszeiten der Monoflops (untere Kurven).

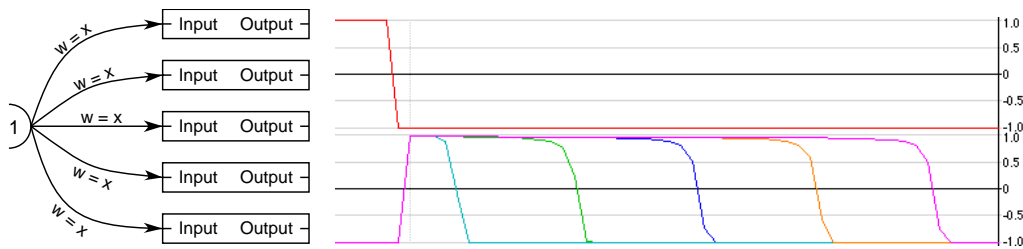


Abbildung 40: Links: Fünf Monoflops in einer Test-Structure. Rechts: CurveDisplay-Ausgabe. In der oberen, roten Kurve ist der Eingang dargestellt. Wenn dieser von +1 auf -1 wechselt, werden alle fünf Monoflop-Ausgänge (untere Kurven) aktiviert, um nach vom Parameter wf abhängigen Zeiten wieder auf -1 zu wechseln.

Die Systemelemente Input und Output wurden hier genutzt, um Ein- und Ausgänge von Structures zu erstellen. Inputs und Outputs können aber auch andere Funktionen haben. Beispielsweise ist es möglich, Werte aus Dateien zu lesen bzw. in Dateien zu schreiben oder Werte von einem Roboter zu lesen bzw. Ansteuerungswerte für einen Roboter zu schreiben. Mittels UDP-Ein- und Ausgängen ist es möglich, per UDP mit anderen Desktop-Programmen zu kommunizieren. Zusätzlich zu UDP-Ein- und Ausgängen ist es auch möglich, den gesamten BrainDesigner per UDP fernzusteuern. Beispielsweise kann eine Evolutionssoftware Slider-Werte setzen, das Netz eine gewisse Anzahl Schritte berechnen lassen und aus den Ausgabewerten eine Fitness berechnen und somit den Slider-Wert optimieren.

4.2 Plugins und Konfigurationen

Die Software BrainDesigner an sich stellt nur die grafische Oberfläche zur Verfügung, mittels derer hierarchische Netze erstellt werden können. Den Elementen der untersten Ebene (Units und Synapsen) ist dabei neuronaler Bytecode hinterlegt (siehe Kapitel 4.3). Die Kompilierung oder Interpretierung dieses Bytecodes wird über Plugins festgelegt. Plugins steuern zudem Ein- und Ausgänge des Netzes. Somit ist es möglich, neue Roboter oder andere Programme an den BrainDesigner anzuschließen. Derzeit existieren Plugins unter anderem für die Roboter A-Serie, Myon und Semni des Labors für Neurorobotik, zum

Lesen und Schreiben von Textdateien (CSV-Format), von Audiodateien (WAV) und zum Kommunizieren mit anderen Programmen (via UDP).

Bevor ein Plugin Berechnungen übernimmt, wird der Netzstruktur-Compiler aufgerufen. Dieser erzeugt aus einem hierarchischen Netz eine flache Liste und bestimmt dabei die Parameter jeder Modulinstanz. Parameter können von einer Structure an eingebettete Structures vererbt werden und auch auf unterster Ebene noch aus Berechnungen bestehen. Beispielsweise kann eine Structure einen Parameter x haben, der an einer Standard-Synapse direkt deren Parameter w zugewiesen wird ($w = x$), an einer anderen aber in eine Berechnung einbezogen wird ($w = 1-x$). Wurde diese Structure in eine andere Structure eingebunden und dabei der Parameter x gesetzt, wird vom Netzstruktur-Compiler nicht nur die Kapselung der Structures aufgelöst sondern es werden auch die Parameter der Modulinstanzen berechnet.

Plugins werden beim Start im Unterordner „plugins“ gesucht und dynamisch geladen. Jedes Plugin stellt einen sogenannten PlayerHandler zur Verfügung. PlayerHandler ist ein Interface, das jedes PlayerHandler-Plugin implementieren muss. Neben einigen Funktionen, um den PlayerHandler zu konfigurieren, gibt es vor allem folgende:

NewTopology	Mit dieser Funktion wird – nach dem Aufruf des Netzstruktur-Compilers – die aktuelle Netztopologie übergeben. Konkret handelt es sich um eine Liste sämtlicher unterschiedlicher Modultypen und ihre Instanzen im Netz. Für jede Instanz ist angegeben, was an ihren Ein- und Ausgängen angeschlossen ist. Hierarchische Netze sind zu diesem Zeitpunkt bereits aufgelöst, alle Modultypen sind also Units oder Synapsen, wobei hier nicht mehr zwischen diesen unterschieden werden muss.
Start	Start der Berechnung.
TimeStep	Ein Zeitschritt der Berechnung.
Stop	Ende der Berechnung.
Pause	Berechnung wird pausiert.
Resume	Berechnung wird fortgesetzt.

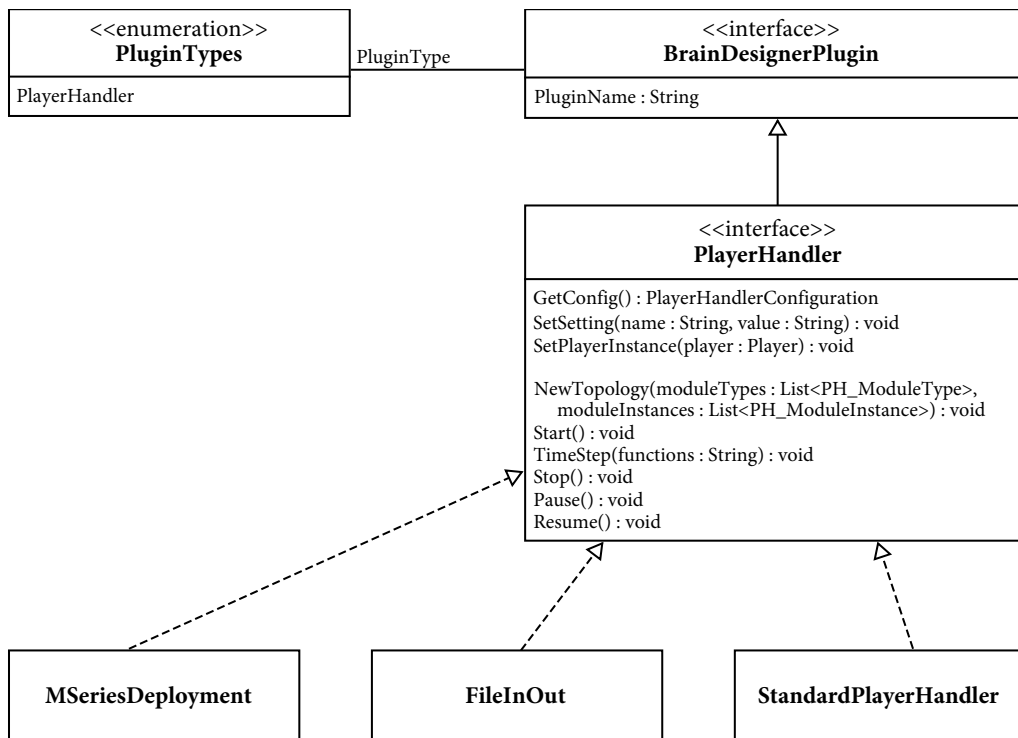


Abbildung 41: UML-Darstellung des Plugin-Typs PlayerHandler. PlayerHandler ist ein Interface, das sich vom allgemeinen Interface BrainDesignerPlugin ableitet. Jedes PlayerHandler-Plugin ist eine Realisierung dieses Interfaces. Als Beispiel sind drei Realisierungen angegeben.

Ein Beispiel ist das Plugin FileInOut, das die Möglichkeit bietet, Werte aus Dateien zu lesen und in Dateien zu schreiben. Über die Konfiguration wird dem BrainDesigner mitgeteilt, dass es den Eingangstyp „File“ gibt, der die Parameter „Filename“ und „Position“ erwartet. Die Funktion NewTopology nimmt die Liste der Modulinstanzen entgegen, die beim Aufruf von Start nach passenden Ein- und Ausgängen durchsucht wird. In TimeStep wird in jedem Zeitschritt eine Zeile jeder Eingabedatei gelesen und die Werte der Netzeingänge werden entsprechend gesetzt. Umgekehrt werden die Werte der Netzausgänge vom Typ „File“ in jedem Zeitschritt in die entsprechenden Dateien geschrieben.

Das Plugin FileInOut behandelt also ausschließlich Ein- und Ausgänge. Im Gegensatz dazu führt das Plugin StandardPlayerHandler ausschließlich Berechnungen durch. Für jede Modulinstanz wird in jedem Zeitschritt der übergebene

neuronalen Bytecode interpretiert. Das Plugin MSeriesDeployment hingegen kompiliert den neuronalen Bytecode für ARM-Prozessoren und speichert das Ergebnis im Programmspeicher der AccelBoard3D-Prozessoren des Roboters Myon.

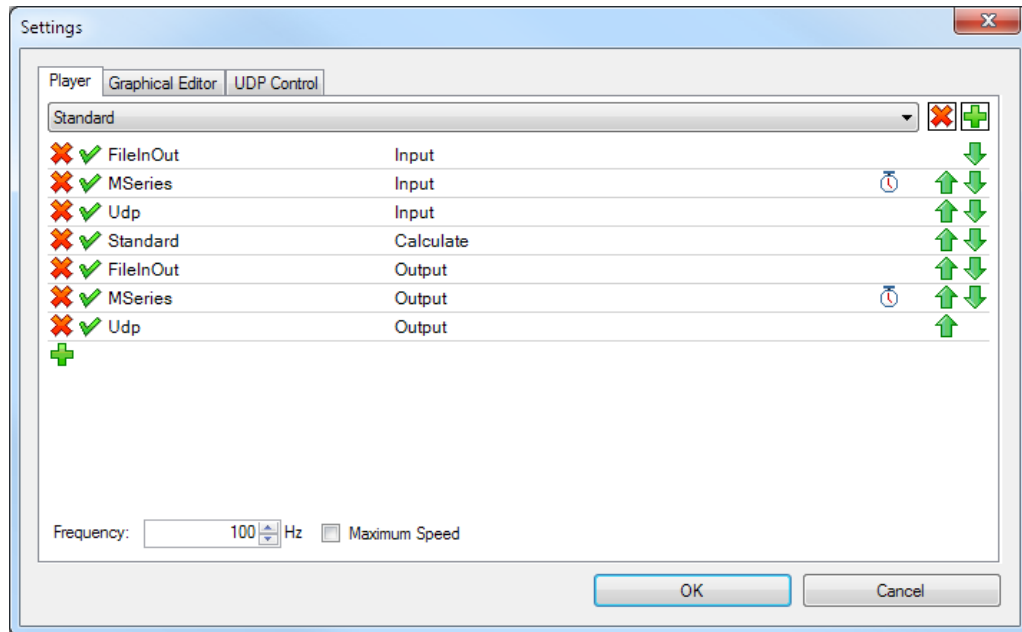


Abbildung 42: Das Konfigurations-Fenster des BrainDesigners. Die genutzten Plugins werden hier eingestellt. Die Reihenfolge kann mittels der grünen Pfeile rechts verändert werden, das rote X links entfernt den Eintrag. Mit Klick auf den grünen Haken kann ein Eintrag deaktiviert werden. Mit dem grünen Plus links unten können neue Einträge hinzugefügt werden.

Es können immer mehrere Plugins aktiv sein, die dann in einer festgelegten Reihenfolge ausgeführt werden. Dies wird *Konfiguration* genannt. Abbildung 42 zeigt das Konfigurationsfenster mit einer beispielhaften Konfiguration. In dieser wird das Netz nicht auf den Roboter geflasht sondern lokal im BrainDesigner ausgerechnet. Es werden jedoch sowohl Werte vom Roboter Myon gelesen („MSeries Input“, vor der Berechnung) als auch Werte an ihn geschrieben

(„MSeries Output“, nach der Berechnung). Bei der gezeigten Konfiguration handelt es sich also um den Transparent Mode. Abbildung 43 zeigt die Abarbeitung im Überblick.

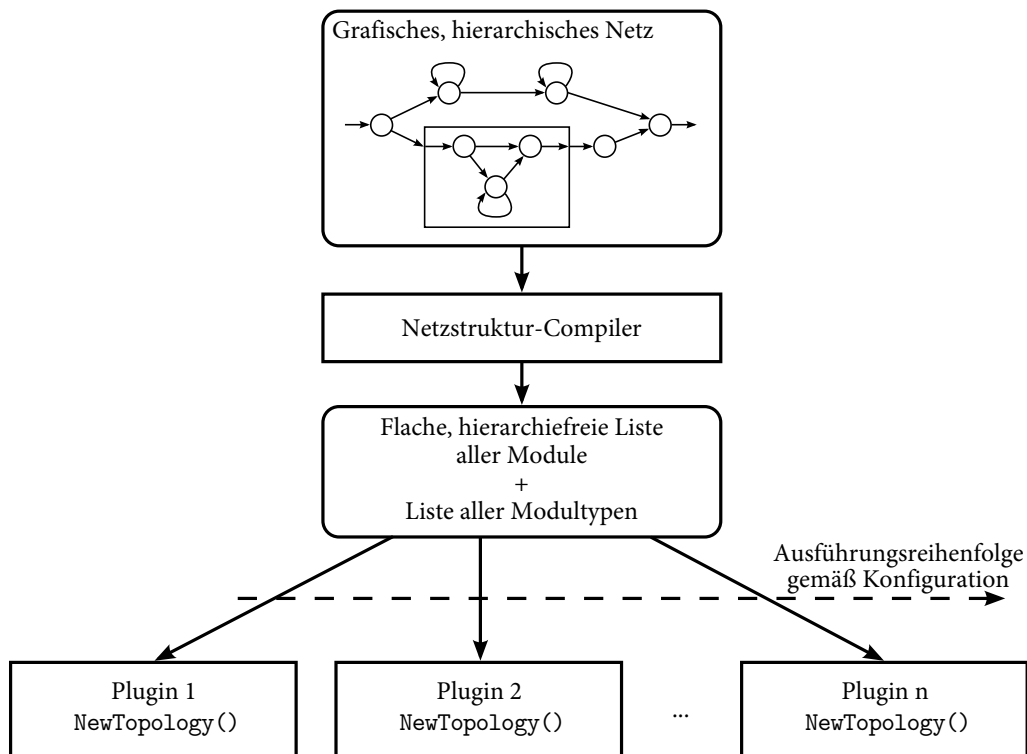


Abbildung 43: Reihenfolge beim Starten der Berechnung eines Netzes. Der Netzstruktur-Compiler erzeugt eine flache Liste aller Module ohne Hierarchien und eine Liste aller Modultypen. Anschließend werden entsprechend der aktuellen Konfiguration alle Plugins aufgerufen.

Die Frequenz der Berechnung (Zeitschritte pro Sekunde) kann von einem Plugin und somit beispielsweise einem angeschlossenen externen Roboter vorgegeben werden. Ist kein Plugin aktiv, das den Takt vorgibt, macht dies der BrainDesigner selbst. Die Frequenz kann im Konfigurations-Fenster festgelegt werden (Standardwert sind 100 Hz) oder mittels eines Hakens bei „Maximum Speed“ die Berechnung schnellstmöglich durchgeführt werden.

4.3 Der neuronale Bytecode

Alle Units und Synapsen enthalten neuronalen Bytecode, der die eigentliche Berechnung der mittels des BrainDesigner erstellten Netze darstellt. Dabei handelt es sich um eine assemblerähnliche Sprache. Sie ist bewusst einfach gehalten, um leicht für eingebettete Prozessoren umsetzbar zu sein.

Die Reihenfolge der Modulinstanzen innerhalb eines BrainDesigner-Netzes ist nicht vorgegeben. Trotzdem ist die Berechnungsreihenfolge in vielen Fällen natürlich wichtig. Durch die Auftrennung eines Neurons des üblichen Neuronenmodells in Synapsen-Module und Neuronen-Module muss sichergestellt sein, dass immer zuerst die Synapsen ihre Eingänge mit dem angegebenen Gewicht multiplizieren und dann erst die Neuronen die nun zur Verfügung stehenden Eingangswerte addieren und die Transferfunktion darauf anwenden. Dies wird sichergestellt, indem im neuronalen Bytecode mittels Zeitstempeln die relative Ausführungsreihenfolge angegeben wird. Der genaue Wert spielt keine Rolle, nur die Reihenfolge wird beachtet. Konkret wird der Bytecode für eine Standard-Synapse bei Zeitstempel 200 ausgeführt, der Bytecode für ein Tanh-Neuron bei Zeitstempel 300.

Der neuronale Bytecode für die Standard-Synapse (Zeitstempel 200) und das Tanh-Neuron (Zeitstempel 300) lautet wie folgt:

```
200:
    mul    V0, Input, w
    write Output, V0

300:
    tanh  V0, Input
    write Output, V0
```

In der Synapse wird mittels eines Multiplikationsbefehls (`mul`) der Input mit dem Parameter `w` multipliziert und in das interne Register `V0` geschrieben. Der Inhalt des Registers `V0` wird mit dem `write`-Befehl in den Output geschrieben. Das Tanh-Neuron hat nur einen Eingang („Input“), alle angeschlossenen Synapsen werden automatisch aufaddiert. Der Befehl `tanh` wendet den Tangens Hyperbolicus als Transferfunktion an und anschließend wird das Ergebnis wiederum mit dem `write`-Befehl an den Ausgang geschrieben.

Auf diese Weise ist sichergestellt, dass alle Synapsen vor den Neuronen berechnet werden. Der Code für einen Zeitstempel wird *Snippet* genannt. Snippets für Zeitstempel kleiner 100 werden nur einmalig bei der Initialisierung des Netzes ausgeführt. Das Tanh-Neuron kann beispielsweise zusätzlich das folgende Snippet haben:

```
50:  
  load V0, Initial  
  write Output, V0
```

Dieses lädt den Parameter Initial in ein internes Register und schreibt diesen Wert in den Output. Somit kann der initiale Ausgabewert eines Tanh-Neurons mit einem Parameter vorgegeben werden.

Als Berechnungsgrundlage dienen neben zwei internen Registern V0 und V1 folgende Elemente:

- Inputs: Bei Units können diese definiert werden und stehen dann unter dem angegebenen Namen zur Verfügung. Wird kein Input angegeben steht implizit ein Eingang „Input“ zur Verfügung. Bei Synapsen gibt es immer nur diesen einen Eingang.
- Parameter: Parameter können während der Bearbeitung des Netzes für jede Instanz eingegeben werden, ändern sich aber während der Laufzeit nicht.
- Internals: Interne Werte, die über die Laufzeit beibehalten werden. In diesen können sich interne Zustände gemerkt werden.
- Outputs: Diese können wie Inputs nur bei Units definiert werden, ansonsten steht implizit ein Output mit dem Namen „Output“ zur Verfügung. Internals und Outputs unterscheiden sich nur dadurch, dass andere Modulinstanzen auf Outputs zugreifen können, auf Internals jedoch nicht.

Abbildung 44 zeigt, wie diese Elemente im BrainDesigner für eine Unit angegeben werden können. In diesem Beispiel gibt es einen benannten Eingang „in“ und einen benannten Ausgang „out“, drei Parameter („dead_zone“, „slope“ und „ramp“) und einen internen Zustand „tmp“.

Inputs		Parameters			Internals		Outputs	
No.	Name	No.	Name	Standard	No.	Name	No.	Name
1	in	1	dead_zone	0	1	tmp	1	out
		2	slope	1				
		3	ramp	0				



Abbildung 44: Eingabemöglichkeit für Inputs, Parameter, Internals und Outputs unterhalb des Quellcode-Editors im BrainDesigner. Ganz unten wird eine Vorschau des Moduls angezeigt (in diesem Fall eine Identitäts-Unit mit Totzone).

Der Bytecode ist vollständig linear, es gibt keine Sprunganweisungen oder Schleifen. Dies beschränkt die maximale Ausführungszeit und ist somit für die Echtzeitfähigkeit der DISTAL-Architektur unvermeidlich. Tabelle 10 zeigt alle Befehle des neuronalen Bytecodes, Anhang 4 listet weitere Informationen zu sämtlichen Befehlen.

Befehl	Beschreibung
abs	Absolutwert
add	Addition
div	Division (wenn Divisor 0: Ergebnis 0)
load	Laden eines Wertes in ein Register (V0 oder V1)
max	Maximalwert von zwei Werten
min	Minimalwert von zwei Werten
mul	Multiplikation
sat	Sättigung im Bereich ± 1 . -1 für alle Werte kleiner -1 , Identität zwischen -1 und $+1$ und $+1$ für alle Werte größer $+1$.
sub	Subtraktion
tanh	Tangens Hyperbolicus
write	Schreiben eines Wertes in einen Output oder ein Internal.

Tabelle 10: Befehle des neuronalen Bytecodes.

4.4 Verwendung des BrainDesigners mit dem Roboter Myon

Für den Roboter Myon (siehe Kapitel 2) gibt es zwei unterschiedliche BrainDesigner-Plugins. Das Plugin „MSeries“ liest den SpinalCord und gibt dessen Werte (siehe Kapitel 3.5.4) mit dem 100-Hz-Takt an Inputs eines Netzes weiter. Nach der Berechnung auf dem PC im BrainDesigner werden Motor-Ausgabewerte per Transparent Mode an den Roboter geschrieben. Das MSeries-Plugin stellt zu diesem Zweck die Input- und Output-Typen „MSeries“ bereit, die mit einem kleinen Roboterkopf im Netz gekennzeichnet werden.

Abbildung 45 zeigt ein sehr einfaches Netz ohne Units. Der Wert eines Beschleunigungssensors im Kopf (SpinalCord-Feld 670) wird mit einem Faktor auf einen Motor gegeben (Motor 42). Der Beschleunigungssensor liefert in dieser Achse Null, wenn der Kopf waagrecht gehalten wird, ansonsten einen von Null verschiedenen Wert je nach Neigungsrichtung. Wird dieser Wert als Ansteuerungsspannung auf den Motor 42 gegeben, wird der Kopf waagrecht ausgerichtet. Es handelt sich um einen P-Regler mit dem Sollwert 0. Durch die Anordnung des Beschleunigungssensors im Vergleich zum Motor ist w positiv.

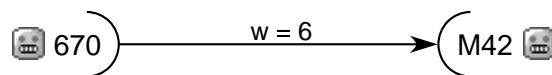


Abbildung 45: Einfaches BrainDesigner-Netz für den Roboter Myon. Ein Beschleunigungswert des Kopfes wird auf einen Motor des Kopfes gegeben, der somit immer waagrecht zur Erde gehalten wird.

Zusätzlich zum Transparent-Mode-Plugin „MSeries“ gibt es das Plugin „MSeriesDeployment“. Es stellt genau dieselben Inputs und Outputs zur Verfügung. Nach der Umstellung der BrainDesigner-Konfiguration kann dasselbe Netz, das vorher im Transparent Mode auf dem PC berechnet wurde, nun für die Prozessoren der AccelBoard3D-Platinen des Roboters Myon (siehe Kapitel 3.6.1) kompiliert werden. Im folgenden Kapitel wird diese Kompilierung erläutert.

4.4.1 Kompilierung für ARM-Cortex-M3-Prozessoren

Die hierarchische Struktur eines Netzes mit geschachtelten Structures wird bereits vom BrainDesigner aufgelöst. Er liefert dem Deployment-Plugin eine Liste unterschiedlicher Modultypen und eine Liste aller Modulinstanzen. Zwischen Units und Synapsen wird an dieser Stelle nicht mehr unterschieden. Abbildung 46 zeigt den internen Ablauf des Deployments für das AccelBoard3D.

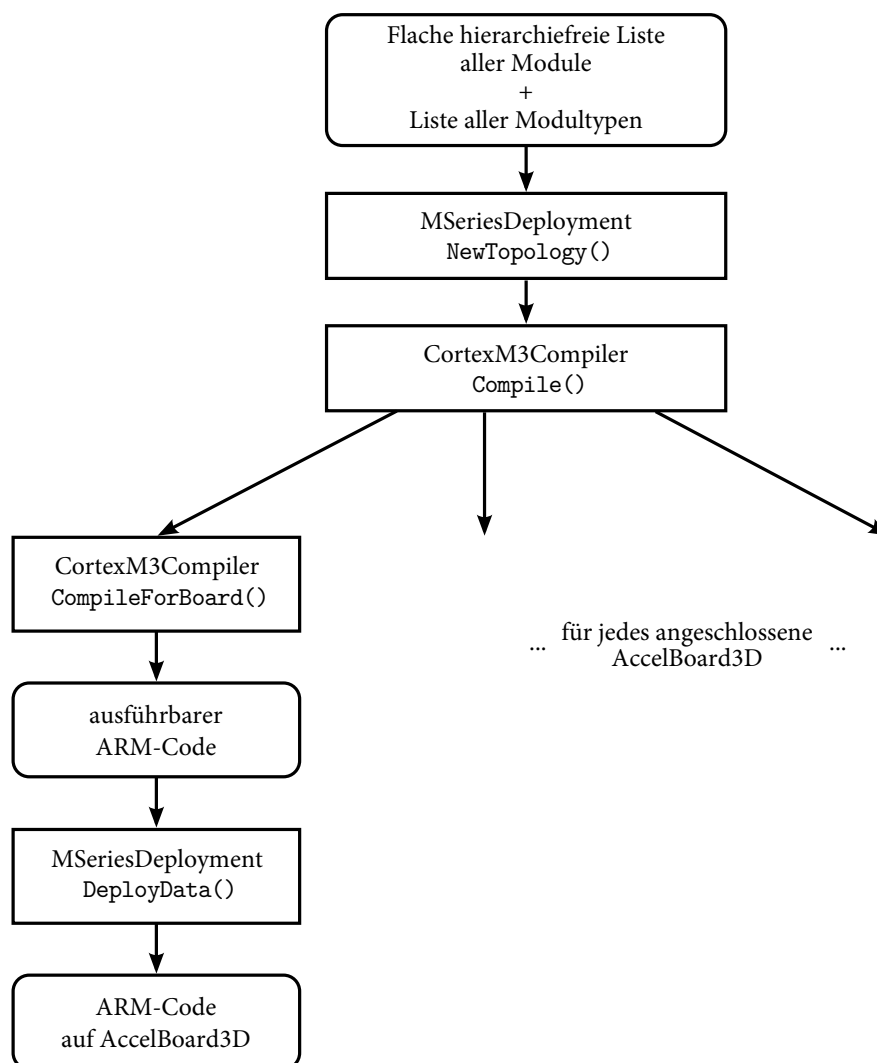


Abbildung 46: Interner Ablauf beim Deployment für das AccelBoard3D.

In einem ersten Schritt bestimmt das Deployment-Plugin nun für jedes AccelBoard3D, welche Modulinstanzen relevant sind. In dem einfachen Beispiel, das den Kopf waagrecht hält, muss Code ausschließlich für das AccelBoard3D mit der ID 30 im Kopf generiert werden. Auf allen anderen Boards muss die Berechnung nicht stattfinden. Ausgehend von allen Outputs, die einem Board zugeordnet sind (eigene SpinalCord-Felder und an dieses Board angeschlossene Motoren), wird rückwärts durch das Netz gegangen, bis alle für das Board relevanten Modulinstanzen ermittelt sind.

Teile des Gesamtnetzes können dabei auf verschiedenen Boards parallel laufen. Soll ein größerer, separater Teil nur auf einem Board berechnet werden, muss das Netz an dessen Ausgabewert aufgeteilt werden. Der Ausgabewert wird in ein freies SpinalCord-Feld des Boards geschrieben, auf dem die Berechnung stattfinden soll. Alle anderen Boards, die diesen Wert benötigen, nutzen dann dieses SpinalCord-Feld als Input. Diese Aufteilung muss manuell erfolgen, eine automatische Aufteilung findet nicht statt.

Aus der Liste relevanter Modulinstanzen und Modultypen wird nun für jedes Board ARM-Code erstellt. Der STM32F103-Prozessor der AccelBoard3D-Platine ist ein Cortex-M3-Prozessor mit ARMv7-M-Architektur. Der Thumb-2-Befehlssatz mischt 16-Bit-Thumb-Befehle mit 32-Bit-ARM-Befehlen. In der weiteren Arbeit werden immer Assembler-Entsprechungen für die verwendeten Befehle angegeben. Es wird jedoch kein Assembler zur Erzeugung des Maschinencodes verwendet sondern eine selbst geschriebene C#-Klasse mit statischen Funktionen für jeden Assembler-Befehl. Eine solche Funktion liefert ein Byte-Array mit dem Maschinencode zurück (2 oder 4 Bytes). Aus diesem wird der Gesamtcode erzeugt, der dann in den Flash-Speicher des Prozessors geschrieben wird.

Im RAM des Prozessors gibt es für alle Modulinstanzen einen definierten Speicherbereich. In diesem werden für die Outputs und internen Werte (Internals) jeder Modulinstanz jeweils 32 Bit reserviert. Diese Werte werden „Locals“ genannt. Jede Instanz hat mindestens einen Local-Wert (einen Output).

Der SpinalCord enthält 16-Bit-Festkommawerte mit 15 Nachkommabits im Intervall $[-1, +1)$. Entsprechend werden auch die 32-Bit-Locals als Festkommawerte mit 15 Nachkommabits interpretiert, sie haben also 17 Vorkommabits und bewegen sich im Intervall $[-65536, +65536)$ mit einer Auflösung von ca. $3 \cdot 10^{-5}$.

Der STM32F103 besitzt keine Floating Point Unit (FPU), das Rechnen mit Festkommawerten ist daher wegen der Performance nötig.

Bei Cortex-M3-Mikroprozessoren gibt es neben speziellen Program Status Registern sechzehn 32-Bit-Register. Die Register R0 bis R12 sind General-Purpose-Register, R13 der Stack Pointer (SP), R14 das Link-Register (LR) und R15 der Program Counter (PC) [ARM10]. Die 13 General-Purpose-Register werden für die Kompilierung eines BrainDesigner-Netzes verwendet.

Die Kompilierung teilt sich in zwei Bereiche. Zum einen wird für jedes Snippet jedes Modultyps Code erstellt. Dabei handelt es sich um ein Unterprogramm, an dessen Ende ein Rücksprung steht. Der zweite Bereich enthält für jede Modulinstanz einen Sprung zum entsprechenden Snippet-Unterprogramm. Vorher werden in die 13 Register alle für das Snippet-Unterprogramm relevanten Daten abgelegt. Der Code zum Füllen der Register wird Präambel der Instanz genannt.

Register	Verwendung
R0–R7	Eingabewerte für Snippets
R8–R9	Interne Snippet-Register V0 und V1
R10	RAM-Startadresse SpinalCord-Speicherbereich
R11	RAM-Startadresse Locals-Speicherbereich
R12	RAM-Startadresse Locals der aktuellen Modulinstanz

Tabelle 11: Verwendung der 13 General-Purpose-Register bei der Cortex-M3-Kompilierung.

Die ersten acht Register (R0 bis R7) werden mit Eingabewerten für das Snippet gefüllt (Inputs und Parameter). R8 und R9 sind freie Register, die für die Bytecode-Zwischenwerte V0 und V1 verwendet werden. Das Register R10 enthält die RAM-Startadresse des SpinalCord-Speicherbereichs, Register R11 die RAM-Startadresse des Locals-Speicherbereichs und R12 die RAM-Startadresse der Locals für die aktuelle Modulinstanz. Tabelle 11 zeigt die Register im Überblick.

Ist die Berechnung auf dem AccelBoard3D aktiviert, springt die Systemsoftware in jedem Zeitschritt in den Speicherbereich der Anwendungssoftware. Dort wird

zunächst der Inhalt des Registers R0 überprüft. Ist R0 = 0 wird einmaliger Initialisierungscode ausgeführt, ist R0 = 1 wird der Schleifencode ausgeführt. Es wird also zum einen Code für alle Snippets mit Zeitstempel kleiner 100 und alle Instanzen, die diese nutzen, erstellt, zum anderen Code für alle Snippets mit Zeitstempel größer 100 und alle Instanzen, die diese nutzen. Im Schleifenfall wird der Initialisierungscode übersprungen und nur der Schleifencode ausgeführt. Die Kompilierung ist für den Initialisierungscode und den Schleifencode identisch, im Folgenden wird der Schleifencode erläutert.

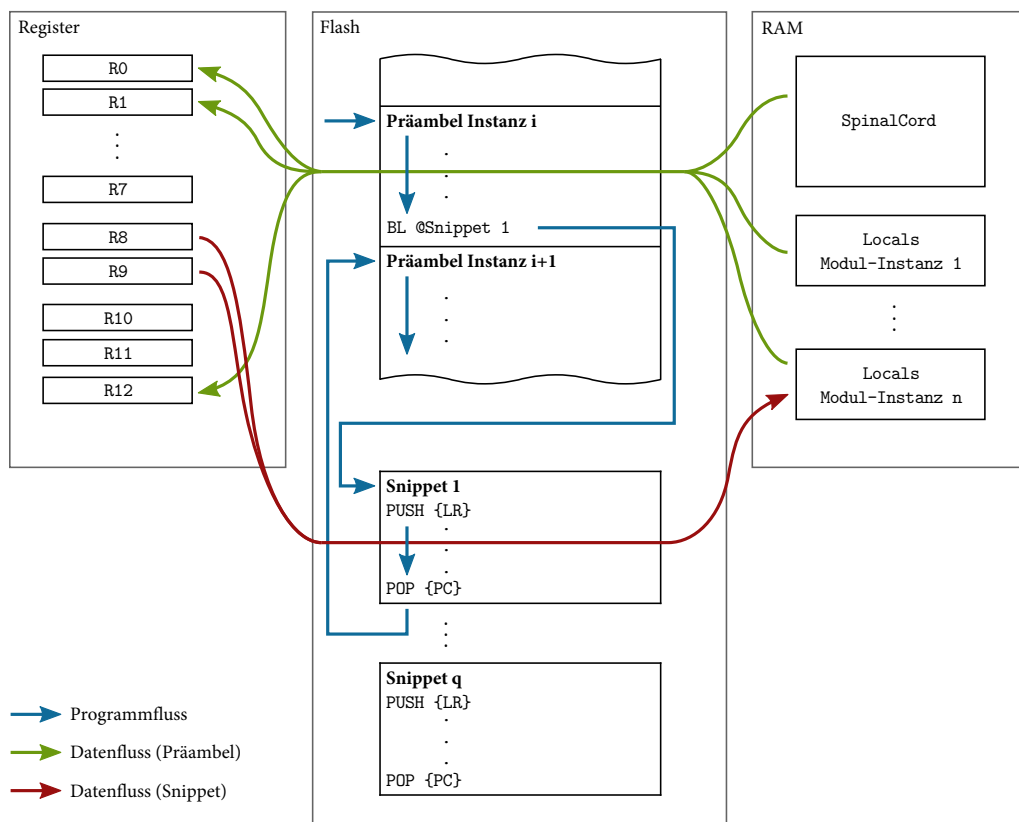


Abbildung 47: Speicher sowie Programm- und Datenfluss bei der Berechnung des neuronalen Bytecodes auf einem ARM-Cortex-M3-Prozessor.

Zunächst werden die beiden konstanten Register R11 (SpinalCord-RAM-Startadresse) und R12 (Locals-RAM-Startadresse) gefüllt. Anschließend werden alle Präambeln in der Reihenfolge ihrer Snippet-Zeitstempel erstellt und zuletzt die

Ausgänge des Netzes geschrieben (Motor-Outputs oder SpinalCord-Felder). In den Speicherbereich hinter dem Rücksprung in die Systemsoftware sind die Snippet-Codes abgelegt, die jeweils mit einem Rücksprung zur Präambel der nächsten Modulinstanz enden.

Abbildung 47 zeigt den Ablauf schematisch. Bei der Berechnung für Modulinstanz i werden in der zugehörigen Präambel zunächst alle nötigen Werte aus dem RAM in die Register geladen. Auch Parameter werden in die Register abgelegt. Dann wird ein Sprung zum zugehörigen Snippet durchgeführt (hier: Snippet 1). Dort wird mit den Werten in den Registern gearbeitet und am Ende die Ergebnisse in die Locals der Instanz abgelegt. Nun wird zurückgesprungen und die Präambel der Instanz $i+1$ beginnt.

Die Belegung der Register R0 bis R7 ist abhängig vom Modultyp. Die ersten Register sind die Inputs des Moduls, gefolgt von den Parametern. Daraus ergibt sich, dass ein Modul nie mehr als zusammen acht Inputs und Parameter haben darf.

Zum Füllen der Input-Register werden alle Ausgänge angeschlossener Modulinstanzen addiert. Der erste Wert wird direkt in das entsprechende Register geladen, alle weiteren zunächst in das während der Präambel ungenutzte Register R8 und anschließend addiert.

Das folgende Beispiel zeigt die Präambel für den ersten Input einer Modulinstanz:

```
LDRSH.W R0, [R10, 1400]
```

LDRSH ist der Befehl zum Laden eines „Signed halfword“, also eines vorzeichenbehafteten 16-Bit-Wertes, wie er im SpinalCord abgelegt ist. R10 enthält die Startadresse des SpinalCords, 1400 ist ein Offset in Bytes. Es wird also der Wert des SpinalCord-Feldes 700 in Register R0 geladen.

Das nächste Beispiel füllt den zweiten Input mit der Summe von zwei Locals anderer Modulinstanzen:

```
LDR.W R1, [R11, 12]  
LDR.W R8, [R11, 28]  
ADD.N R1, R8
```

In der ersten Zeile wird der Local-Wert geladen, der sich 12 Byte nach dem Anfang des Speicherbereichs befindet. Da ein Local-Wert immer 32 Bit groß ist,

handelt es sich um den insgesamt vierten Local-Wert. In der zweiten Zeile wird ein zweiter Wert zunächst in Register R8 geladen und in der letzten Zeile zum ersten Wert in Register R1 addiert.

Parameter werden mit zwei Befehlen direkt in die Register geschrieben (MOVW: 16-Bit-Immediate; MOVT: 16-Bit-Immediate in die oberen 16 Bit).

Im Snippet-Code wird auf die in der Präambel gefüllten Register zugegriffen. Bei einer Standard-Synapse befindet sich in R0 der Eingangswert und in R1 der Parameterwert w. Mit dem ersten Bytecode-Befehl werden beide Werte multipliziert und in V0 (Register R8) geschrieben. In ARM-Code übersetzt sieht dieser Befehl wie folgt aus:

```
PUSH    {R2}
SMULL   R2, R8, R0, R1
LSR     R8, R8, #15
LSL     R2, R2, #17
ORR     R8, R8, R2
POP     {R2}
```

Die Multiplikation zweier 32-Bit-Werte ist ein 64-Bit-Wert. Zusätzlich zum Zielregister R8 wird also noch ein temporäres Register benötigt, hier R2. Der dort vorhandene Wert wird am Anfang auf den Stack gelegt und am Ende von dort zurückgelesen. Der ARM-Befehl SMULL führt die Multiplikation von R0 und R1 durch, das 64-Bit-Ergebnis liegt in R2 und R8. Multipliziert man zwei Festkommawerte mit je 15 Nachkommastellen hat das Ergebnis 30 Nachkommastellen. Um wieder ein 32-Bit-Ergebnis mit 15 Nachkommastellen zu erhalten, werden die 15 unteren Bits und die 17 oberen Bits abgeschnitten. Mit einem logischen Oder (ORR) können die beiden Register dann wieder zusammengeführt werden.

Mit der zweiten Zeile des neuronalen Bytecodes der Standard-Synapse wird nun das Ergebnis, das sich in Register R8 befindet, an den Ausgang der Synapse geschrieben. In ARM-Code ist dies auch nur ein Befehl:

```
STR     R8, [R12, #0]
```

R12 enthält die Adresse, an der die Locals der aktuellen Instanz liegen. Bei der Standard-Synapse mit einem Ausgang und ohne Internals ist der Offset 0. STR ist der ARM-Befehl zum Speichern eines 32-Bit-Wertes aus einem Register in den RAM.

Das gesamte Snippet wird eingerahmt vom Sichern des Link-Registers auf dem Stack (push) und dem Rückschreiben desselben vom Stack in den Program Counter (pop).

Wie die Befehle des neuronalen Bytecodes in ARM-Code übersetzt werden, ist für jeden Befehl in der Befehlsreferenz in Anhang 4 dargelegt.

4.4.2 Maximale Netzgrößen beim AccelBoard3D

Für die Abarbeitung großer Netze gibt es drei Beschränkungen: die Rechenzeit, der vorhandene Flash-Speicher für den Programmcode und der RAM zum Vorhalten der Locals. In diesem Kapitel sollen alle drei Ressourcen untersucht werden. Dabei wird sich auf Netze beschränkt, die aus Tanh-Neuronen (ohne Bias und Initialisierungswerte) und Standard-Synapsen bestehen.

Die vorhandene Rechenzeit pro Zeitschritt beträgt beim Roboter Myon 5,44 ms. Da der Prozessor mit 72 MHz getaktet ist, entspricht dies 391.680 Zyklen. Bei den folgenden Berechnungen wird immer vom ungünstigsten Fall ausgegangen. Ein Sprung dauert beispielsweise abhängig vom Sprungziel zwei bis vier Zyklen, hier wird immer von vier Zyklen ausgegangen. Die Division beim Cortex-M3 unterstützt „early termination“, bricht also bei bestimmten Eingabewerten früher ab. Auch hier wird immer von der ungünstigsten Situation ausgegangen.

Für die Anwendungssoftware stehen auf dem AccelBoard3D 80 kB zur Verfügung, also 81.920 Byte.

Unabhängig von der Größe des Netzes gibt es Code, der immer ausgeführt wird und beispielsweise prüft, ob es sich um Initialisierungscode oder Schleifencode handelt. Dieser Code ist 52 Byte lang und benötigt 52 Zyklen. Dazu kommen pro Output des Netzes (Schreiben in den SpinalCord) 12 Byte und 5 Zyklen.

Das Snippet der Standard-Synapse ist 28 Byte groß und benötigt zur Ausführung 21 Zyklen. Das Tanh-Neuron benutzt eine tanh-Funktion, die die Systemsoftware zur Verfügung stellt. Diese arbeitet mit einer Look-Up-Tabelle mit Interpolation. Insgesamt benötigt das Tanh-Neuron im Speicherbereich der Anwendungssoftware 18 Byte. Die Ausführung dauert 65 Zyklen.

Für jede Instanz wird eine Präambel angelegt und abgearbeitet. Bei der Standard-Synapse ist jede Präambel 24 Byte lang und benötigt zur Abarbeitung 10 Zyklen. Beim Tanh-Neuron ist dies abhängig von der Anzahl angeschlossener Synapsen.

Das Minimum bei einer Synapse beträgt 16 Byte und 8 Zyklen. Für jede weitere angeschlossene Synapse kommen 6 Byte und 3 Zyklen hinzu.

Tabelle 12 gibt einen Überblick über den benötigten Flash-Speicher und die benötigten Rechenzeiten.

Codefragment	Flash-Größe (Bytes)	Zeit (Zyklen)
Fester Code	52	52
Code pro Output	12	5
Standard-Synapse: Snippet	28	21
Standard-Synapse: Präambel	24	10
Tanh-Neuron: Snippet	18	65
Tanh-Neuron: Präambel mit n angeschlossenen Synapsen	$10+6n$	$5+3n$

Tabelle 12: Übersicht über den benötigten Flash-Speicher und die benötigte Rechenzeit für verschiedene Codefragmente.

Für die weitere Berechnung wird angenommen, dass es N Tanh-Neuronen gibt und S Standard-Synapsen, wobei $S \geq N$. Die Anzahl durchschnittlich an ein Neuron angeschlossener Synapsen beträgt somit S/N . Die Anzahl der Outputs ist im Vergleich vernachlässigbar und wird hier mit fünf angenommen.

Der feste Speicherbedarf beträgt somit 158 Byte. Dazu kommen $24 \cdot S$ Bytes für die Synapsen-Präambeln und $(10 + 6 S/N) \cdot N = 10 \cdot N + 6 \cdot S$ Bytes für die Neuronen.

Der feste Zeitbedarf beträgt 77 Zyklen. Dazu kommen $31 \cdot S$ Zyklen für die Synapsen und $(70 + 3 S/N) \cdot N = 70 \cdot N + 3 \cdot S$ Zyklen für die Neuronen.

Tabelle 13 zeigt für verschiedene Verhältnisse von Synapsen zu Neuronen, welche maximale Anzahl Neuronen und Synapsen möglich sind bezogen auf den vorhandenen Flash-Speicher und die vorhandene Rechenzeit. In dieser Konstellation ist also immer der Flash-Speicher die beschränkende Ressource. Ist der Bytecode in den verwendeten Modulen umfangreicher, erhöht sich der Speicherbedarf zwar kaum, die benötigte Rechenzeit unter Umständen jedoch merklich.

Synapsen pro Neuron	Speicherplatzbeschränkt		Ausführungszeitbeschränkt	
	Neuronen	Synapsen	Neuronen	Synapsen
1	2.044	2.044	3.765	3.765
2	1.168	2.336	2.837	5.674
3	817	2.451	2.276	6.828
4	628	2.512	1.900	7.600
5	511	2.555	1.631	8.155
6	430	2.580	1.429	8.574
7	371	2.597	1.271	8.897

Tabelle 13: Maximal mögliche Neuronen bzw. Synapsen für verschiedene Verhältnisse von Synapsen zu Neuronen jeweils mit beschränktem Flash-Speicher und beschränkter Rechenzeit. Engpass ist hier immer der Speicherplatz.

Im RAM muss Speicher für die Locals vorgesehen werden. Jedes Tanh-Neuron und jede Standard-Synapse benötigt 32 Bit Speicherplatz. Der verwendete Prozessor hat insgesamt 20 kB RAM (20.479 Byte). Die Systemsoftware benötigt davon insgesamt 3.704 Byte, es verbleiben 16.775 Byte. Dies ist ausreichend für 4.193 Locals.

Die maximale Anzahl Modulinstanzen, die im Flash abgelegt werden kann (beim Verhältnis $S/N = 1$) beträgt 4.088 und ist somit kleiner als die Anzahl der möglichen Locals im RAM. Der Arbeitsspeicher ist also nicht die begrenzende Resource.

Bei Netzen, die nur aus einfachen Tanh-Neuronen und Standard-Synapsen bestehen sind pro AccelBoard3D bis zu 4.000 Modulinstanzen möglich. Bei 21 AccelBoard3D im Roboter Myon sind somit bis zu 84.000 Modulinstanzen denkbar. Bei einer mittleren Synapsen-Neuronen-Quote von 4,0 sind rund 625 Neuronen bei 2.500 Synapsen pro Board möglich, insgesamt also über 13.000 Neuronen und 52.000 Synapsen.

Seit der Entwicklung der AccelBoard3D-Platine wurden neue Embedded-ARM-Prozessoren mit mehr Leistung vorgestellt. Für eine Weiterentwicklung der Rechenknoten würde sich ein Cortex-M4-Prozessor anbieten, beispielsweise ein STM32F4 von ST Microelectronics. Der STM32F405 ist in einer kleinen Bauform erhältlich (LQFP-64, 10×10 mm), die trotzdem noch einfach zu bestücken ist. Dieser Prozessor ist bereits einzeln für unter 10 Euro erhältlich. Er kann mit bis zu 168 MHz getaktet werden, hat 192 kB RAM und 1 MB Flash-Speicher.

Synapsen pro Neuron	Speicherplatzbeschränkt Neuronen	Synapsen	Ausführungszeitbeschränkt Neuronen	Synapsen
1	22.933	22.933	8.786	8.786
2	13.104	26.208	6.622	13.244
3	9.173	27.519	5.313	15.939
4	7.056	28.224	4.436	17.744
5	5.733	28.665	3.807	19.035
6	4.828	28.968	3.335	20.010
7	4.169	29.183	2.967	20.769

Tabelle 14: Maximal mögliche Neuronen bzw. Synapsen beim STM32F405. Im Unterschied zum Prozessor der AccelBoard3D-Platine ist hier immer die Ausführungszeit die Beschränkung.

Tabelle 14 zeigt die speicherplatz- bzw. rechenzeitbeschränkten maximalen Neuronen- bzw. Synapsenzahlen für verschiedene Verhältnisse von Synapsen zu Neuronen beim STM32F405. Für die Berechnung wurden wiederum 5,44 ms angenommen. Da der Flash-Speicher beim STM32F405 nicht kilobyteweise gelöscht werden kann, werden 128 kB für DISTAL-Bootloader, Konfigurationsbereich und Systemsoftware angesetzt. Es bleiben also 896 kB für die Anwendungssoftware. Da die Systemsoftware nicht mehr RAM benötigen wird, können weit über 40.000 Modulinstanzen vorgehalten werden.

Im Vergleich zum Cortex-M3-Prozessor auf dem AccelBoard3D ist nun also die Rechenzeit der beschränkende Faktor. Bei durchschnittlich vier Synapsen pro

Neuron sind somit pro Prozessorknoten ca. 4.400 Neuronen und 17.600 Synapsen möglich. Dies stellt eine Versiebenfachung im Vergleich zum AccelBoard3D dar.

5 Fazit und Ausblick

Autonome Robotik ist ein interdisziplinäres Fachgebiet. Um erfolgreich Forschung in diesem Bereich betreiben zu können, müssen Hardware, Systemarchitektur und Werkzeuge eng miteinander verzahnt sein. Man kann dies als die drei Säulen der Robotik-Forschung bezeichnen. Von zentraler Bedeutung ist hierbei die Systemarchitektur.

Im Rahmen dieser Arbeit wurde die Echtzeit-Systemarchitektur DISTAL vorgestellt (Kapitel 3). Aus der Analyse der Anforderungen an Forschungsroboter wurden wesentliche Konzepte für das Design der Architektur abgeleitet. Die Besonderheit entsteht aus dem Zusammenspiel der Aspekte Modularität und Autonomie. Da es möglich sein sollte, einzelne Teile eines Roboters autonom zu betreiben, wurde DISTAL so konzipiert, dass kein zentraler Rechenknoten notwendig ist. Mehrere autonome Knoten können – auch im Betrieb – miteinander verbunden werden und so Daten über das gesamte System austauschen. Ebenso ermöglicht die Laufzeit-Flexibilität, Teile eines Roboters abzunehmen, ohne dass es zu einer Unterbrechung der Regelungsprogramme kommt.

Trotz der Komplexität eines solchen verteilten Systems sollte es möglich sein, dass sich unerfahrene Anwender in kürzester Zeit in das System einarbeiten können. Beim nano-Camp von 3sat haben zwölf Jugendliche innerhalb von nur 90 Minuten Regelschleifen auf einem Roboter in Betrieb nehmen können und dabei eindrucksvoll bewiesen, dass dieses Ziel erreicht wurde.

Realisierbar war diese schnelle Einarbeitung durch die enge Verbindung der Systemarchitektur mit dem Werkzeug BrainDesigner. Mit dieser Software, die ebenfalls in der Arbeit vorgestellt wurde (Kapitel 4), ist es möglich, sensomotorische Regelschleifen grafisch zu erstellen. Damit auch große Netze übersichtlich bearbeitet werden können, ist die hierarchische Kapselung von Teilnetzen möglich. Die erforderliche Flexibilität wurde erreicht, indem Modulen der untersten Ebene ein neuronales Bytecode hinterlegt ist (Kapitel 4.3). Er legt fest, wie die Daten verarbeitet werden. Die erstellten Netze können direkt im BrainDesigner berechnet und ihre Werte visualisiert werden. Ein Roboter kann dabei an den

PC angeschlossen direkt betrieben werden. Gleichzeitig ist es mit nur einer einfachen Konfigurationsänderung möglich, aus dem Netz per Knopfdruck Code für Embedded-Prozessoren zu erzeugen und in deren Programmspeicher abzulegen. In diesem Modus kann ein Roboter autonom betrieben werden, wobei es jederzeit möglich ist, ihn an einen PC anzuschließen und den BrainDesigner zur Visualisierung der Daten zu verwenden.

Ein Roboter, der die vorgestellte Architektur nutzt und bei dessen Entwicklung vor allem die Aspekte Modularität, Autonomie, Einfachheit und Robustheit große Bedeutung hatten, ist Myon (Kapitel 2). An ihm wurden in dieser Arbeit die Systemarchitektur und die Kompilierung der Netze des BrainDesigners beispielhaft erläutert.

Auf dem Roboter Myon wurden mithilfe der DISTAL-Systemarchitektur inzwischen erfolgreich etliche Regelungen umgesetzt. Das „aufstehende Bein“ ist ein Einzelbein, das mittels des CSL-Paradigmas [Hil13] aufsteht und auch auf unebenen Untergründen steht. Mit diesem Experiment wurde gezeigt, dass auch mit einzelnen Körperteilen gearbeitet werden kann. Das „stabile Stehen“ überträgt das Paradigma auf den gesamten Roboter. Auch bei kleinen Stößen und geneigten Böden steht der Roboter stabil. Darüber hinaus konnten auch Laufbewegungen erfolgreich umgesetzt werden.

In dieser Arbeit wurde gezeigt, dass auch anspruchsvolle Regelungen für humanoide Roboter auf einer vollständig dezentralen Architektur möglich sind. Die Aussage von [UBL06] aus dem Jahr 2006, dass solche Architekturen für humanoide Roboter nicht geeignet seien, da die Kinematik und Dynamik eine zentrale Regelungsinstanz erfordern⁷, ist inzwischen also überholt.

5.1 Ausblick

Die DISTAL-Architektur ermöglicht die einfache Adaption für neue Systeme. Darüber hinaus sind auch bei bestehenden Implementierungen, wie der des Ro-

⁷ „While fully decentral controller architectures have been implemented successfully for many multi-legged walking machines, they are not suitable for humanoid robots—the highly coupled kinematics and dynamics demand a central controller instance” [UBL06].

boters Myon, Erweiterungen möglich. Für Module, die die Batterien überwachen und deren Ladestand übermitteln, wurden bereits bei der Planung Übertragungsslots vorgesehen. Sie sind also ohne großen Aufwand einzubinden. Der Austausch bestehender Komponenten ist ebenfalls einfach möglich, derzeit wird beispielsweise das BrainModule überarbeitet. Die Elektronik der Motoreinheiten Dynamixel RX-28 könnte durch eigene ersetzt werden, die die Aufgaben der Regelungsplatinen (AccelBoard3D) mit übernehmen. Dies ist zwar bei 44 Motoren pro Myon-Roboter ein erheblicher Aufwand und die Datenkommunikation müsste für mehr Teilnehmer angepasst werden. Auf die AccelBoard3D-Platinen und die Kabel zu den Motoreinheiten könnte dann aber verzichtet werden. Dies würde nicht nur Gewicht sparen, durch die wegfallende Kommunikation zwischen AccelBoard3D und Dynamixel wäre auch eine potentielle Fehlerquelle eliminiert. Dies würde die Zuverlässigkeit des Systems nochmals erhöhen.

Auch bei den Werkzeugen sind Anpassungen möglich. Für andere DISTAL-Systeme muss der Quellcode des BrainDesigners dank seiner Erweiterbarkeit durch Plugins nicht verändert werden. Mit der Bereitstellung einer leicht zu entwickelnden DLL können sofort Netze für einen neuen Roboter erstellt werden. Auch für Myon sind Änderungen denkbar, die ebenfalls nur die jeweiligen Plugins betreffen. Beispielsweise wird derzeit für jedes AccelBoard3D bestimmt, welche Teile eines neuronalen Netzes lokal notwendig sind. Es kann daher vorkommen, dass große Teilnetze parallel auf mehreren Boards berechnet werden. Das ist aufgrund der großen Anzahl möglicher Neuronen pro Board bisher keine Einschränkung und erhöht die Robustheit beim Ausfall eines Boards. Sollten die Netze zukünftig aber größer werden, könnte der BrainDesigner solche mehrfach vorkommenden Teilnetze automatisch erkennen und dem Nutzer Handlungsempfehlungen zur Bündelung auf einem AccelBoard3D geben. Erweiterungen sind ebenso beim neuronalen Bytecode denkbar. Mit den derzeitigen Befehlen ist es möglich, unzählige lokale Lernverfahren zu implementieren. Mit geringen Änderungen kann diese Flexibilität auch für globale Lernverfahren erzielt werden. Sowohl DISTAL als auch der BrainDesigner sind so anpassungsfähig, dass sie auch zukünftig für anspruchsvolle Aufgaben und neue Robotiksysteme eingesetzt werden können.

Anhang

Anhang 1: SpinalCord-Datenfelder beim Roboter Myon

Auf den folgenden Seiten sind alle 720 SpinalCord-Datenfelder des Roboters Myon dargestellt. Sync- und ID-Feld sind türkis dargestellt, das Peephole rosa, Sensordaten gelb, Ansteuerungsdaten für Motoren grün. Felder, die von sensor-motorischen Programmteilen frei genutzt werden können sind blau eingefärbt.

Bei den Sensordaten gelten folgende Bezeichnungen:

- **Voltage:** Versorgungsspannung
- **Temp:** Temperatur
- **AccelX/AccelY/AccelZ:** Beschleunigungswerte in drei Achsen. ± 1 entspricht dabei dem Messbereich des Sensors $\pm 8g$.
- **Current DS A/Current DS B:** Am DoubleStrand A bzw. DoubleStrand B gemessener Strom. Welche Motoren an DoubleStrand A bzw. DoubleStrand B angeschlossen sind kann an den vier „Position“-Sensorfeldern abgelesen werden.
- **Position:** Winkelposition des internen Potentiometers der Dynamixel RX-28.
- **Angle:** Winkelposition von extern an Gelenke angebrachten Potentiometern.
- **Force:** Kraftsensoren an der Fußsohle.

0	Head	1	Left Leg	2	Right Leg	3	Left Arm	4	Right Arm
0	SYNC	27	SYNC	30	SYNC	33	SYNC	36	SYNC
1	ID	28	ID	31	ID	34	ID	37	ID
2	Peephole	29	Peephole	32	Peephole	35	Peephole	38	Peephole
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									
26									

5	Body	6	Head	7	Left Leg Lower Bottom	8	Right Leg Lower Bottom	9	Left Leg Lower Top
39	SYNC	42	SYNC	45	SYNC	72	SYNC	99	SYNC
40	ID	43	ID	46	ID	73	ID	100	ID
41	Peephole	44	Peephole	47	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M1 5 6 7	74	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M2 5 6 7	101	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M3 5 Temp M5 6 Temp M7 7 Temp M9
				48	AccelX	75	AccelX	102	AccelX
				49	AccelY	76	AccelY	103	AccelY
				50	AccelZ	77	AccelZ	104	AccelZ
				51	Current DS A M1	78	Current DS A M2	105	Current DS A M3+M5
				52		79		106	Current DS B M7+M9
				53	DS A, Position 1 Position 1	80	DS A, Position 1 Position 2	107	DS A, Position 1 Position 3
				54		81		108	DS A, Position 2 Position 5
				55		82		109	DS B, Position 1 Position 7
				56		83		110	DS B, Position 2 Position 9
				57	Ankle LToes	84	Ankle RToes	111	Ankle LAnklePitch
				58	Ankle LAnkleRoll	85	Ankle RAnkleRoll	112	
				59	Force LLeftBack	86	Force RLeftBack	113	
				60	Force LLeftFront	87	Force RLeftFront	114	
				61	Force LRightBack	88	Force RRightBack	115	
				62	Force LRightFront	89	Force RRightFront	116	
				63		90		117	
				64		91		118	
				65		92		119	
				66		93		120	
				67		94		121	
				68	DS A, Torque 1 Motor 1	95	DS A, Torque 1 Motor 2	122	DS A, Torque 1 Motor 3
				69		96		123	DS A, Torque 2 Motor 5
				70		97		124	DS B, Torque 1 Motor 7
				71		98		125	DS B, Torque 2 Motor 9

10	Right Leg Lower Top	11	Left Leg Upper Bottom	12	Right Leg Upper Bottom	13	Left Leg Upper Middle	14	Right Leg Upper Middle
126	SYNC	153	SYNC	180	SYNC	207	SYNC	234	SYNC
127	ID	154	ID	181	ID	208	ID	235	ID
128	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M4 5 Temp M6 6 Temp M8 7 Temp M10	155	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M13 5 Temp M15 6 Temp M11 7	182	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M14 5 Temp M16 6 Temp M12 7	209	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M17 5 Temp M19 6 Temp M21 7	236	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M18 5 Temp M20 6 Temp M22 7
129	AccelX	156	AccelX	183	AccelX	210	AccelX	237	AccelX
130	AccelY	157	AccelY	184	AccelY	211	AccelY	238	AccelY
131	AccelZ	158	AccelZ	185	AccelZ	212	AccelZ	239	AccelZ
132	Current DS A M4+M6	159	Current DS A M13+M15	186	Current DS A M14+M16	213	Current DS A M17+M19	240	Current DS A M18+M20
133	Current DS B M8+M10	160	Current DS B M11	187	Current DS B M12	214	Current DS B M21	241	Current DS B M22
134	DS A, Position 1 Position 4	161	DS A, Position 1 Position 13	188	DS A, Position 1 Position 14	215	DS A, Position 1 Position 17	242	DS A, Position 1 Position 18
135	DS A, Position 2 Position 6	162	DS A, Position 2 Position 15	189	DS A, Position 2 Position 16	216	DS A, Position 2 Position 19	243	DS A, Position 2 Position 20
136	DS B, Position 1 Position 8	163	DS B, Position 1 Position 11	190	DS B, Position 1 Position 12	217	DS B, Position 1 Position 21	244	DS B, Position 1 Position 22
137	DS B, Position 2 Position 10	164		191		218		245	
138	Anale RAnklePitch	165	Anale LKnee	192	Anale RKnee	219	Anale LHipPitch	246	Anale RHipPitch
139		166		193		220		247	
140		167		194		221		248	
141		168		195		222		249	
142		169		196		223		250	
143		170		197		224		251	
144		171		198		225		252	
145		172		199		226		253	
146		173		200		227		254	
147		174		201		228		255	
148		175		202		229		256	
149	DS A, Torque 1 Motor 4	176	DS A, Torque 1 Motor 13	203	DS A, Torque 1 Motor 14	230	DS A, Torque 1 Motor 17	257	DS A, Torque 1 Motor 18
150	DS A, Torque 2 Motor 6	177	DS A, Torque 2 Motor 15	204	DS A, Torque 2 Motor 16	231	DS A, Torque 2 Motor 19	258	DS A, Torque 2 Motor 20
151	DS B, Torque 1 Motor 8	178	DS B, Torque 1 Motor 11	205	DS B, Torque 1 Motor 12	232	DS B, Torque 1 Motor 21	259	DS B, Torque 1 Motor 22
152	DS B, Torque 2 Motor 10	179		206		233		260	

15	Left Leg Upper Top	16	Right Leg Upper Top	17	Left Arm Lower	18	Right Arm Lower	19	Left Arm Upper
261	SYNC	288	SYNC	315	SYNC	342	SYNC	369	SYNC
262	ID	289	ID	316	ID	343	ID	370	ID
263	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M23 5 6 Temp M25 7	290	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M24 5 6 Temp M26 7	317	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M27L 5 6 Temp M29 7	344	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M27R 5 6 Temp M30 7	371	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M31 5 6 Temp M33 7
264	AccelX	291	AccelX	318	AccelX	345	AccelX	372	AccelX
265	AccelY	292	AccelY	319	AccelY	346	AccelY	373	AccelY
266	AccelZ	293	AccelZ	320	AccelZ	347	AccelZ	374	AccelZ
267	Current DS A M23	294	Current DS A M24	321	Current DS A M27	348	Current DS A M28	375	Current DS A M31
268	Current DS B M25	295	Current DS B M26	322	Current DS B M29	349	Current DS B M30	376	Current DS B M33
269	DS A, Position 1 Position 23	296	DS A, Position 1 Position 24	323	DS A, Position 1 Position 27 (Greifer)	350	DS A, Position 1 Position 27 (Greifer)	377	DS A, Position 1 Position 31
270		297		324		351		378	
271	DS B, Position 1 Position 25	298	DS B, Position 1 Position 26	325	DS B, Position 1 Position 29	352	DS B, Position 1 Position 30	379	DS B, Position 1 Position 33
272		299		326		353		380	
273	Anale LHipRoll	300	Anale RHipRoll	327	Anale (Ver.2) LElbow	354	Anale (Ver.2) RElbow	381	Anale (Ver.2) LShoulderRoll
274		301		328		355		382	
275		302		329		356		383	
276		303		330		357		384	
277		304		331		358		385	
278		305		332		359		386	
279		306		333		360		387	
280		307		334		361		388	
281		308		335		362		389	
282		309		336		363		390	
283		310		337		364		391	
284	DS A, Torque 1 Motor 23	311	DS A, Torque 1 Motor 24	338	DS A, Torque 1 Motor 27 (Greifer)	365	DS A, Torque 1 Motor 27 (Greifer)	392	DS A, Torque 1 Motor 31
285		312		339		366		393	
286	DS B, Torque 1 Motor 25	313	DS B, Torque 1 Motor 26	340	DS B, Torque 1 Motor 29	367	DS B, Torque 1 Motor 30	394	DS B, Torque 1 Motor 33
287		314		341		368		395	

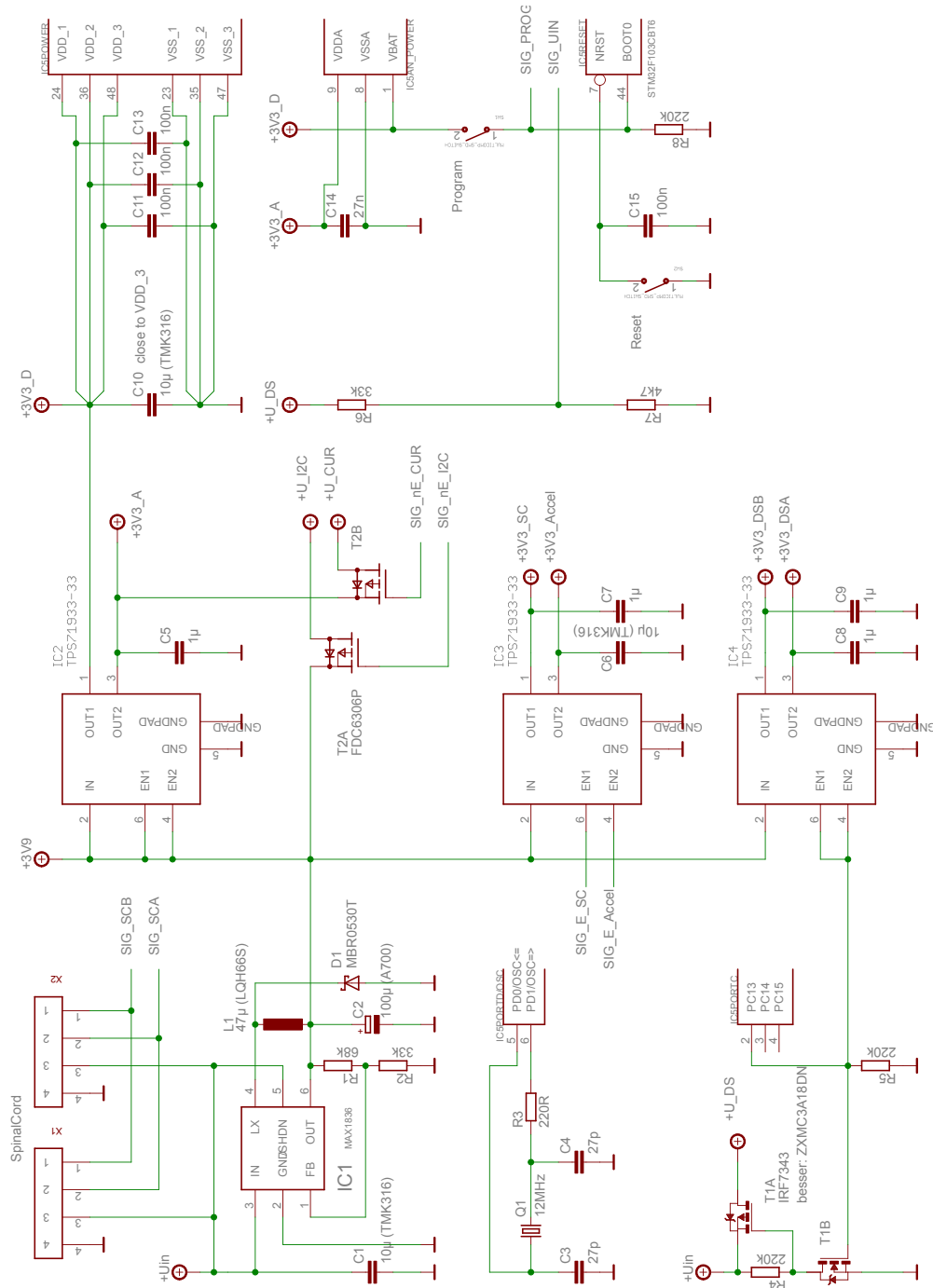
20	Right Arm Upper	21	Body Right Upper	22	Body Upper Center	23	Body Lower Center	24	Body Bottom Front
396	SYNC	423	SYNC	450	SYNC	477	SYNC	504	SYNC
397	ID	424	ID	451	ID	478	ID	505	ID
398	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M32 5 6 Temp M34 7	425	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M36 5 6 Temp M38 7	452	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M35 5 6 Temp M37 7	479	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M39 5 6 7	506	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 5 6 7
399	AccelX	426	AccelX	453	AccelX	480	AccelX	507	AccelX
400	AccelY	427	AccelY	454	AccelY	481	AccelY	508	AccelY
401	AccelZ	428	AccelZ	455	AccelZ	482	AccelZ	509	AccelZ
402	Current DS A M32	429	Current DS A M36	456	Current DS A M35	483	Current DS A M39	510	
403	Current DS B M34	430	Current DS B M38	457	Current DS B M37	484		511	
404	DS A, Position 1 Position 32	431	DS A, Position 1 Position 36	458	DS A, Position 1 Position 35	485	DS A, Position 1 Position 39	512	
405		432		459		486		513	
406	DS B, Position 1 Position 34	433	DS B, Position 1 Position 38	460	DS B, Position 1 Position 37	487		514	
407		434		461		488		515	
408	Anale (Ver.2) RShoulderRoll	435	Anale (Ver.2) RShoulderPitch	462	Anale (Ver.2) LShoulderPitch	489	Anale WaistRoll	516	
409		436		463		490		517	
410		437		464		491		518	
411		438		465		492		519	
412		439		466		493		520	
413		440		467		494		521	
414		441		468		495		522	
415		442		469		496		523	
416		443		470		497		524	
417		444		471		498		525	
418		445		472		499		526	
419	DS A, Torque 1 Motor 32	446	DS A, Torque 1 Motor 36	473	DS A, Torque 1 Motor 35	500	DS A, Torque 1 Motor 39	527	
420		447		474		501		528	
421	DS B, Torque 1 Motor 34	448	DS B, Torque 1 Motor 38	475	DS B, Torque 1 Motor 37	502		529	
422		449		476		503		530	

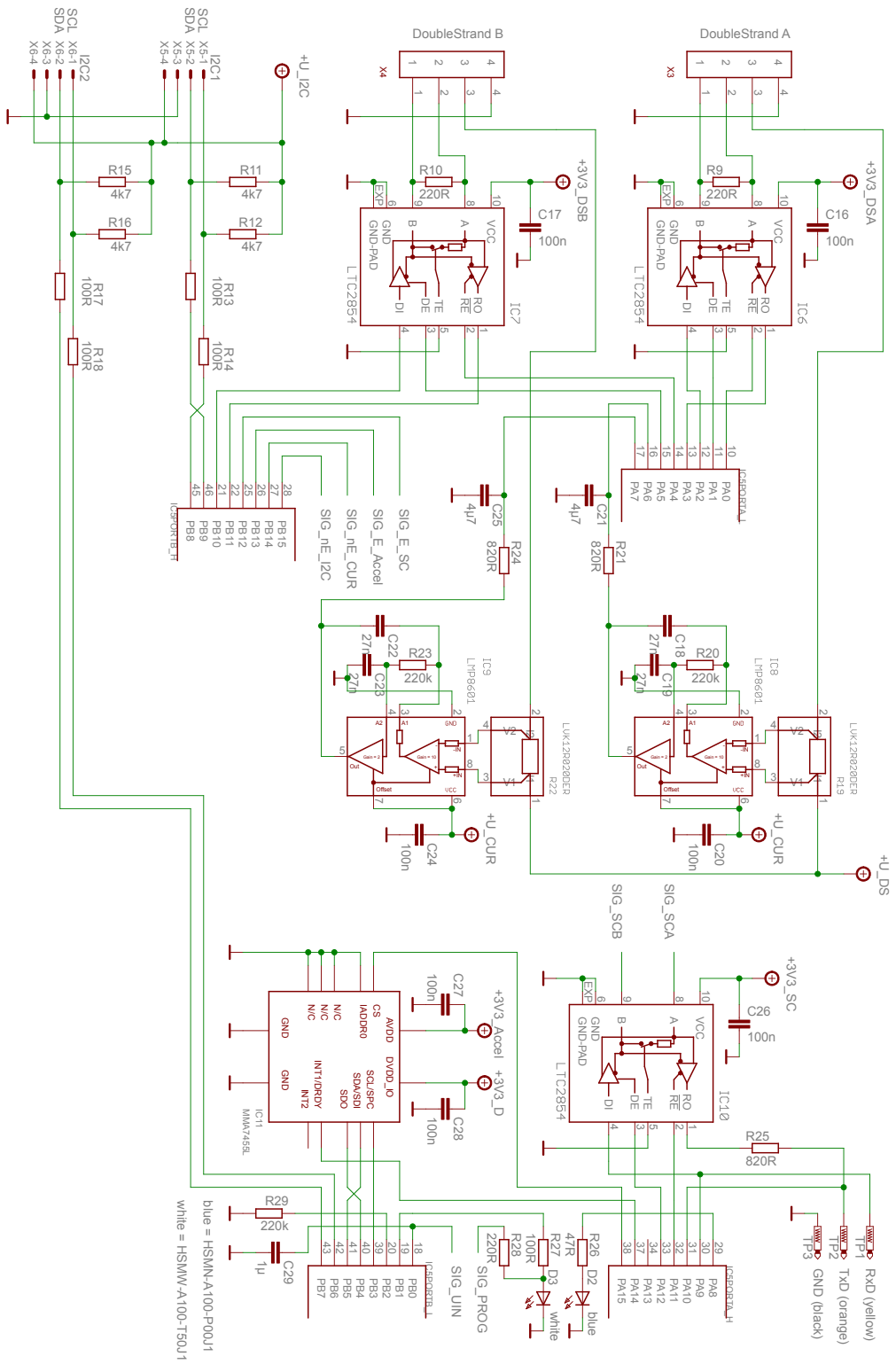
Anhang

25	Body Bottom Back	26	Body Reserved 1	27	Body Reserved 2	28	Body Reserved 3	29	Body Reserved 4
531	SYNC	558	SYNC	585	SYNC	612	SYNC	639	SYNC
532	ID	559	ID	586	ID	613	ID	640	ID
533	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M40 5 6 Temp M41 7	560	Peephole	587	Peephole	614	Peephole	641	Peephole
534	AccelX	561		588		615		642	
535	AccelY	562		589		616		643	
536	AccelZ	563		590		617		644	
537	Current DS A M40	564		591		618		645	
538	Current DS B M41	565		592		619		646	
539	DS A, Position 1 Position 40	566		593		620		647	
540		567		594		621		648	
541	DS B, Position 1 Position 41	568		595		622		649	
542		569		596		623		650	
543	Anale LHipYaw	570		597		624		651	
544	Anale RHipYaw	571		598		625		652	
545		572		599		626		653	
546		573		600		627		654	
547		574		601		628		655	
548		575		602		629		656	
549		576		603		630		657	
550		577		604		631		658	
551		578		605		632		659	
552		579		606		633		660	
553		580		607		634		661	
554	DS A, Torque 1 Motor 40	581		608		635		662	
555		582		609		636		663	
556	DS B, Torque 1 Motor 41	583		610		637		664	
557		584		611		638		665	

30	Head Right	31	Head Left
666	SYNC	693	SYNC
667	ID	694	ID
668	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 Temp M43 5 Temp M44 6 Temp M42 7	695	Peephole 0 Uptime Low 1 Uptime High 2 Voltage DS 3 Firmware Version 4 5 6 7
669	AccelX	696	AccelX
670	AccelY	697	AccelY
671	AccelZ	698	AccelZ
672	Current DS A M43+M44	699	Current DS A LEDs + Servos
673	Current DS B M42	700	Random Value
674	DS A, Position 1 Position 43	701	Brightness LED 2, 1
675	DS A, Position 2 Position 44	702	Brightness LED 4, 3
676	DS B, Position 1 Position 42	703	Brightness LED 6, 5
677		704	Brightness LED 8, 7
678	Ankle HeadPitch	705	Brightness LED 10, 9
679		706	Brightness LED 12, 11
680		707	
681		708	
682		709	
683		710	
684		711	
685		712	
686		713	
687		714	
688		715	
689	DS A, Torque 1 Motor 43	716	Servo 1 Eye left / right
690	DS A, Torque 2 Motor 44	717	Servo 2 Eye up / down
691	DS B, Torque 1 Motor 42	718	Servo 3 Eyelid upper
692		719	Servo 4 Eyelid lower

Anhang 2: Schaltplan AccelBoard3D





Anhang 3: Kommunikationsprotokoll Bootloader AccelBoard3D

In diesem Anhang sind die Befehle verzeichnet, die der entwickelte Bootloader des AccelBoard3D-Prozessors entgegennimmt. Wie man die Systemsoftware verlässt, um mit dem DISTAL-Bootloader zu kommunizieren bzw. wie man den Bootloader beim Start des AccelBoards erreicht, ist in Kapitel 3.6.2 („Speicherbereiche und Bootloader“) erklärt.

Antworten sind normalerweise bei Erfolg 0x06 (ASCII: ACK; Acknowledge) bzw. bei Misserfolg 0x15 (ASCII: NAK; Negative Acknowledge). Das Protokoll ist sehr einfach gehalten, ausschließlich beim Befehl „Schreibe Daten“ wird eine Prüfsumme verwendet.

Jeder Befehl startet mit einem Kommando-Byte, das dem ASCII-Code eines Großbuchstabens entspricht, der zu dem Befehl passt. Der Befehl zum Schreiben von Daten lautet beispielsweise 0x57, ASCII W wie englisch „Write“.

Befehl: Sende eigene ID (0x47; ASCII: „G“)

Veranlasst das Board, seine eigene ID zu senden. Dieser Befehl kann nur ausgeführt werden, wenn ausschließlich ein Board angeschlossen ist, da sonst mehrere Boards antworten würden und es zu Kollisionen auf dem Bus käme.

Anfragepaket: 0x47

Antwortpaket: ID

Befehl: ID vorhanden? (0x49; ASCII „I“)

Mit diesem Befehl kann abgefragt werden, ob ein Teilnehmer mit der entsprechenden ID angeschlossen ist. Dieser antwortet.

Anfragepaket: 0x49, ID

Antwortpaket: 0x06

Befehl: Lösche Block (0x43; ASCII „C“)

Veranlasst eine bestimmte ID, einen Flash-Speicherblock zu löschen. Das Accel-Board3D umfasst 128 kB Flash-Speicher, jeder Block ist 1 kB groß, ein gültiger Bereich für den Block ist also 0 bis 127.

Anfragepaket: 0x43, ID, Block

Antwortpaket: 0x06 (ACK) oder 0x15 (NAK)

Befehl: Schreibe Daten (0x57; ASCII „W“)

Schreibt 64 Byte Daten an die angegebene Adresse, die als 32-Bit-Wert mitgesendet wird. Von den vier Bytes der Adresse wird das niederwertigste Byte zuerst gesendet (Little Endian).

Es wird eine 1-Byte-Prüfsumme mitgesendet. Diese wird ermittelt, indem alle Bytes (außer das Kommando-Byte 0x57) addiert werden und das Ergebnis bitweise invertiert wird. Überläufe über 8 Bit werden ignoriert.

Anfragepaket: 0x57, ID, 4 Byte Adresse, 64 Byte Daten, Prüfsumme

Antwortpaket: 0x06 (ACK) oder 0x15 (NAK)

Befehl: Lese Daten (0x52; ASCII „R“)

Liest 64 Byte Daten von der angegebenen Adresse, die als 32-Bit-Wert mitgesendet wird. Von den vier Bytes der Adresse wird das niederwertigste Byte zuerst gesendet (Little Endian).

Anfragepaket: 0x52, ID, 4 Byte Adresse

Antwortpaket: 64 Byte Daten

Befehl: Ende der Kommunikation (0x53; ASCII „S“)

Mit diesem Befehl beenden alle Boards den Bootloader-Modus und starten die Systemsoftware. Bei diesem Befehl gibt es keine Antwort.

Anfragepaket: 0x53

Anhang 4: Befehle des neuronalen Bytecodes

Alle Befehle des neuronalen Bytecodes folgen dem Muster

Befehl Ziel, Quelle1, Quelle2, ...

Ein solcher Befehl führt eine Berechnung aus, das Ergebnis wird immer im ersten Parameter (Ziel) abgelegt. Eventuelle Quellen-Parameter werden dahinter angegeben. Als Quellen und Ziele stehen dabei je nach Befehl zur Verfügung:

- Inputs/Parameter des Moduls
- Internals/Outputs des Moduls
- interne Register V0 und V1

Gibt es nur einen In- und Output (beispielsweise bei Synapsen), werden diese mit „Input“ und „Output“ angesprochen.

ABS (Absolutwert)

Dieser Befehl liefert den Absolutwert seines Eingabewerts.

Format

```
ABS <src>
```

Erlaubte Werte

```
<src>:  V0, V1
```

Beispiel

```
ABS V0
```

Weitere Hinweise

Das Zielregister entspricht dem Quellregister.

Kompilierung ARM-Prozessor

Der Wert wird mit 0 verglichen. Ist er kleiner, wird der Wert negiert.

Mittels des CMP-Befehls wird das Register mit 0 verglichen. Ist der Wert 0 oder größer wird mit BPL der folgende Befehl übersprungen, der sonst die Negierung vornimmt (RSB zieht R8 von 0 ab).

Beispiel:

```
CMP  R8, #0      ; R8 = Source
BPL  #1          ; PL = Plus or Zero
RSB  R8, R8, #0
```

Ausführungszeit: Bis zu 4 Zyklen (je 1 Zyklus pro Befehl und im Fall des Sprungs ein Zyklus zum Pipeline-Reload des Prozessors).

ADD (Addition)

Dieser Befehl addiert zwei Werte.

Format

```
ADD <dest>, <src1>, <src2>
```

Erlaubte Werte

<dest>: V0, V1

<src1>: Inputs/Parameter, V0, V1

<src2>: Inputs/Parameter, V0, V1

Beispiel

```
ADD V0, Input, x
```

Kompilierung ARM-Prozessor

Die Addition wird mittels eines einzelnen ADD-Befehls durchgeführt.

Beispiel:

```
ADD R8, R8, R9
```

Ausführungszeit: 1 Zyklus.

DIV (Division)

Dividiert den ersten Quellwert (Dividend) durch den zweiten Quellwert (Divisor).

Format

DIV <dest>, <src1>, <src2>

Erlaubte Werte

<dest>: V0, V1

<src1>: Inputs/Parameter, V0, V1

<src2>: Inputs/Parameter, V0, V1

Beispiel

DIV V0, Input1, Input2

Weitere Hinweise

Die Division durch Null wird abgefangen und das Ergebnis ist dann immer gleich Null.

Auf dem ARM-Prozessor gibt es weitere Einschränkungen für den Wertebereich des Dividenten: dieser muss im Intervall [-64, +64) liegen. Das Ergebnis hat eine geringere Nachkommagenauigkeit von 10 Binärstellen ($\sim \pm 0,001$).

Kompilierung ARM-Prozessor

Der ARM-Prozessor stellt einen Signed-DIV-Befehl SDIV zur Verfügung. Um eine Division durch Null zu vermeiden, wird geprüft, ob der Divisor gleich Null ist und in diesem Fall das Ergebnis auf Null gesetzt. Sollte dies der Fall sein, wird der eigentliche Divisions-Abschnitt übersprungen.

Der SDIV-Befehl liefert ein 32-Bit-Ergebnis. Dieses oder vorher der Divident muss bei zwei Festkommawerten mit 15 Nachkommastellen um 15 Binärstellen nach links geschiftet werden. Dadurch geht Genauigkeit verloren: entweder würden beim Dividenten nur zwei Vorkommastellen bleiben oder beim Ergebnis würde es keine Nachkommastellen geben.

Konkret wurde eine Mischung benutzt: Der Dividend wird um 10 Stellen nach links geschiftet, was den Wertebereich auf das Intervall [-64, +64) einschränkt. Das Ergebnis wird zusätzlich um die fehlenden 5 Bit geschiftet, sodass die Nachkommagenauigkeit auf 10 Stellen reduziert wird.

Beispiel:

```
CMP  R1, #0      ; R1 = src2 (Divisor)
BNE  #2
MOVW R8, #0      ; R8 = dest
B    #5          ; Überspringe Division
LSL  R0, R0, #10 ; R0 = src1 (Dividend)
SDIV R8, R0, R1
LSL  R8, R8, #5
```

Ausführungszeit: Bis zu 18 Zyklen (die eigentliche Division benötigt abhängig von Dividend und Divisor bis zu 12 Zyklen).

LOAD (Laden)

Lädt einen Wert in ein internes Register.

Format

```
LOAD <dest>, <value>
```

Erlaubte Werte

<dest>: V0, V1

<value>: Float-Wert, Internals/Outputs, Inputs/Parameter

Beispiel

```
LOAD V0, 0.25
```

```
LOAD V0, Internal1
```

```
LOAD V0, Input1
```

Weitere Hinweise

Es werden Festkommazahlen mit 15 Nachkommastellen verwendet. Die darstellbare Genauigkeit liegt also bei $1/2^{15} \approx 3 \cdot 10^{-5}$.

Kompilierung ARM-Prozessor

Der in einen 32-Bit-Wert umgewandelte Festkommawert (17 Vorkommastellen, 15 Nachkommastellen) wird mittels MOVW und MOVT in das gewünschte Register geladen.

Beispiel:

```
MOVW R8, #0x4000
```

```
MOVT R8, #0x0000
```

Ausführungszeit: 2 Zyklen.

Für Internals/Outputs wird der LDR-Befehl verwendet:

```
LDR R8, [R12, #1]
```

Ausführungszeit: 1 Zyklus.

Für Inputs/Parameter wird der MOV-Befehl verwendet:

```
MOV R8, R2
```

Ausführungszeit: 1 Zyklus.

MAX (Maximalwert)

Übernimmt den größeren von zwei Werten.

Format

```
MAX <dest>, <src1>, <src2>
```

Erlaubte Werte

<dest>: V0, V1

<src1>: Inputs/Parameter, V0, V1

<src2>: Inputs/Parameter, V0, V1

Beispiel

```
MAX V0, Input1, Input2
```

Kompilierung ARM-Prozessor

Es werden src1 und src2 verglichen: ist src2 größer, wird src2 ins Zielregister geschrieben und der letzte Befehl übersprungen, ansonsten wird mit diesem letzten Befehl src1 ins Zielregister geschrieben.

Beispiel:

```
CMP  R0, R1      ; R0 = src1, R1 = src2
B    10, #1      ; Condition 10: Greater
                        ; than or equal
MOV  R8, R1      ; R8 = dest
B    #0          ; Unconditional Branch
MOV  R8, R0
```

Ausführungszeit: Bis zu 5 Zyklen.

MIN (Minimalwert)

Übernimmt den kleineren von zwei Werten.

Format

```
MIN <dest>, <src1>, <src2>
```

Erlaubte Werte

<dest>: V0, V1

<src1>: Inputs/Parameter, V0, V1

<src2>: Inputs/Parameter, V0, V1

Beispiel

```
MIN V0, Input1, Input2
```

Kompilierung ARM-Prozessor

Es werden src1 und src2 verglichen: ist src2 kleiner, wird src2 ins Zielregister geschrieben und der letzte Befehl übersprungen, ansonsten wird mit diesem letzten Befehl src1 ins Zielregister geschrieben.

Beispiel:

```
CMP  R0, R1      ; R0 = src1, R1 = src2
B    11, #1      ; Condition 11: Less than
MOV  R8, R1      ; R8 = dest
B    #0          ; Unconditional Branch
MOV  R8, R0
```

Ausführungszeit: Bis zu 5 Zyklen.

MUL (Multiplikation)

Multipliziert zwei Werte.

Format

```
MUL <dest>, <src1>, <src2>
```

Erlaubte Werte

<dest>: V0, V1

<src1>: Inputs/Parameter, V0, V1

<src2>: Inputs/Parameter, V0, V1

Beispiel

```
MUL V0, Input, w
```

Kompilierung ARM-Prozessor

Die Multiplikation wird mittels des Befehls SMULL durchgeführt, der zwei 32-Bit-Werte multipliziert und ein 64-Bit-Wert als Ergebnis liefert. Dieses wird dann mittels LSR, LSL und ORR entsprechend des Festkommaformats (15 Nachkommastellen) wieder auf 32 Bit gebracht. Neben dem Zielregister (R8 für V0, R9 für V1) wird ein weiteres Zielregister für die oberen 32 Bit benötigt. Hierfür wird ein für diesen Befehl ungenutztes Register verwendet und dessen Inhalt vorher auf den Stack gelegt und nach der Ausführung wieder von dort zurückgeholt.

Beispiel:

```
PUSH R2
SMULL R2, R8, R0, R1 ; R0: src1, R1: src2
LSR R8, R8, #15
LSL R2, R2, #17
ORR R8, R8, R2
POP R2
```

Ausführungszeit: Bis zu 10 Zyklen.

SAT (Sättigung)

Sättigt den angegebenen Parameter im Bereich ± 1 .

Format

```
SAT <src>
```

Erlaubte Werte

```
<src>: V0, V1
```

Beispiel

```
SAT V0
```

Kompilierung ARM-Prozessor

Es wird der ARM-eigene Sättigungsbefehl SSAT genutzt, der Immediate-Wert ist dabei 15, sodass den 15 Nachkommastellen des Festkommaformats entsprechend gesättigt wird.

Beispiel:

```
SSAT R8, R8, #15
```

Ausführungszeit: 1 Zyklus.

SUB (Subtraktion)

Dieser Befehl subtrahiert den zweiten Source-Wert vom ersten.

Format

```
SUB <dest>, <src1>, <src2>
```

Erlaubte Werte

<dest>: V0, V1

<src1>: Inputs/Parameter, V0, V1

<src2>: Inputs/Parameter, V0, V1

Beispiel

```
SUB V0, Input, x
```

Kompilierung ARM-Prozessor

Die Subtraktion wird mittels eines einzelnen SUB-Befehls durchgeführt.

Beispiel:

```
SUB R8, R8, R9
```

Ausführungszeit: 1 Zyklus.

TANH (Tangens Hyperbolicus)

Berechnet den Tangens Hyperbolicus. Es wird eine Funktion genutzt, die die Systemsoftware der DISTAL-Architektur bereitstellt. Diese arbeitet mittels einer Lookup-Tabelle.

Format

```
TANH <dest>, <src>
```

Erlaubte Werte

```
<dest>: V0, V1
```

```
<src>: Inputs/Parameter, V0, V1
```

Beispiel

```
TANH V0, Input
```

Kompilierung ARM-Prozessor

Es wird eine vorbereitete Tanh-Routine mit Lookup-Tabelle verwendet. Der kompilierte Code besteht im Grunde aus einem Sprung zu dieser Funktion. Der Wert muss in Register R0 bereitgestellt werden, weshalb dieses Register zunächst auf den Stack gelegt wird (und am Ende von dort zurückgeholt wird) und der gewünschte Eingabewert nach R0 kopiert wird (wenn es nicht sowieso R0 ist). Nach der Funktion wird der Wert zum gewünschten Output kopiert. Ist der Output sowieso R0, muss nicht kopiert werden, R0 wird in diesem Fall des Weiteren nicht auf dem Stack abgelegt.

Beispiel:

```
PUSH R2
SMULL R2, R8, R0, R1 ; R0: src1, R1: src2
LSR R8, R8, #15
LSL R2, R2, #17
ORR R8, R8, R2
POP R2
```

Ausführungszeit: Bis zu 7 Zyklen (inkl. Sprung) + Ausführungszeit des Tanh (26 Zyklen).

WRITE (Ausgabe schreiben)

Schreibt einen Wert in einen Output oder ein Internal.

Format

```
WRITE <dest>, <src>
```

Erlaubte Werte

<dest>: Outputs/Internals

<src>: Inputs/Parameter, V0, V1, Internals

Beispiel

```
WRITE Output, V0
```

Kompilierung ARM-Prozessor

Es ist zu unterscheiden, ob die Quelle sich bereits in einem Register befindet, oder erst in ein Register zwischengespeichert werden muss (bei Internals).

Befindet sich der zu schreibende Wert bereits in einem Register und die Zieladresse ist ein Offset zu Register R12 (Start des eigenen Datenbereichs im RAM), dann wird der STR-Befehl genutzt.

Beispiel:

```
STR R8, [R12, #0]
```

Ausführungszeit: 1 Zyklus.

Ist dies nicht der Fall, wird zunächst der Wert gelesen und in Register R0 zwischengespeichert, das vorher auf den Stack gelegt wurde. Dieser wird dann mittels des STR-Befehls geschrieben und R0 wieder vom Stack gelesen.

Beispiel:

```
PUSH R0
LDR R0, [R12, #1]
STR R0, [R12, #0]
POP R0
```

Ausführungszeit: 4 Zyklen.

Literatur

- [AKK+05] Kazuhiko Akachi, Kenji Kaneko, Noriyuki Kanehira, Shigehiko Ota, Go Miyamori, Masaru Hirata, Shuuji Kajita, Fumio Kanehiro: *Development of Humanoid Robot HRP-3P*. 5th IEEE-RAS International Conference on Humanoid Robots, 2005.
- [ARA+06] Tamim Asfour, Kristian Regenstein, Pedram Azad, Joachim Schröder, Alexander Bierbaum, Niko Vahrenkamp, Rüdiger Dillmann: *ARMAR III: An Integrated Humanoid Platform for Sensory-Motor Control*. 6th IEEE-RAS International Conference on Humanoid Robots, 2006.
- [ARM10] *ARM Cortex-M3 Technical Reference Manual*. Revision r2p1. ARM Limited, 2005–2008, 2010.
- [AS10] Md. Akhtaruzzaman, A. A. Shafie: *Evolution of Humanoid Robot and contribution of various countries in advancing the research and development of the platform*. IEEE International Conference on Control, Automation and Systems (ICCAS), 2010.
- [Bai05] Jean-Christophe Baillie: *URBI: Towards a Universal Robotic Low-Level Programming Language*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2005.
- [Ben10] Christian Benckendorff: *Technische Realisierung multimodaler Sensorik für humanoide Roboter*. Diplomarbeit, Institut für Informatik, Humboldt-Universität zu Berlin, 2010.
- [BES+13] Thomas Buschmann, Alexander Ewald, Markus Schwienbacher, Valerio Favot, Heinz Ulbrich: *Humanoide Laufmaschinen*. at – Automatisierungstechnik. Methoden und Anwendungen der Steuerungs-, Regelungs- und Informationstechnik. Band 61, Heft 4, Seiten 217–232, 2013.
- [BM13] Jason Blackman, Scott Monroe: *Overview of 3.3V CAN (Controller Area Network) Transceivers*. Application Report, Texas Instruments, 2013.

- [BY08] George Bekey, Junku Yuh: *The status of robotics*. IEEE Robotics & Automation Magazine. Band 15, Ausgabe 1, Seiten 80–86, März 2008.
- [CS00] Ève Coste-Manière, Reid Simmons: *Architecture, the Backbone of Robotics Systems*. IEEE International Conference on Robotics and Automation (ICRA), 2000.
- [Dil05] Rüdiger Dillmann: *Rechnerarchitektur, Sensorik und adaptive Steuerung einer vierbeinigen Laufmaschine mit dynamisch stabilem Gang*. In: *Autonomes Laufen*. Seiten 175–189, Springer, 2005.
- [Duf03] Brian R. Duffy: *Anthropomorphism and the social robot*. Robotics and Autonomous Systems. Band 42, Seiten 177–190, 2003.
- [Eng02] Horst Engels: *CAN-Bus: Felddbusse im Überblick, CAN-Bus-Protokolle, CAN-Bus-Meßtechnik, Anwendungen*. Franzis Verlag, 2002.
- [Far31] Henry George Farmer: *The Organ of the Ancients from Eastern Sources*. William Reeves Bookseller Ltd., London, 1931.
- [Fel05] Max Felser: *Real-time ethernet—industry prospective*. Proceedings of the IEEE. Band 93, Ausgabe 6, Seiten 1118–1129, 2005.
- [Hil07] Manfred Hild: *Neurodynamische Module zur Bewegungssteuerung autonomer mobiler Roboter*. Dissertation, Institut für Informatik, Humboldt-Universität zu Berlin, 2007.
- [Hil13] Manfred Hild: *Defying Gravity – A Minimal Cognitive Sensorimotor Loop Which Makes Robots With Arbitrary Morphologies Stand Up*. 11th International Conference on Accomplishments in Electrical and Mechanical Engineering and Information Technology (DEMI), 2013.
- [HJS06] Manfred Hild, Matthias Jüngel, Michael Spranger: *Humanoid Team Humboldt Team Description 2006*. RoboCup 2006: Robot Soccer World Cup X Preproceedings, RoboCup Federation, 2006.
- [HKK14] Michael ten Hompel, Christopher Kirsch, Thomas Kirks: *Zukunftspfade der Logistik – Technologien, Prozesse und Visionen zur vierten industriellen Revolution*. In: *Enterprise-Integration*. Seiten 203–213, Springer, 2014.

- [HMS07] Manfred Hild, Robin Meißner, Michael Spranger: *Humanoid Team Humboldt Team Description 2007*. RoboCup 2007: Robot Soccer World Cup XI Preproceedings, RoboCup Federation, 2007.
- [HSB+11] Manfred Hild, Torsten Siedel, Christian Benckendorff, Matthias Kubisch, Christian Thiele: *Myon: Concepts and Design of a Modular Humanoid Robot Which Can Be Reassembled During Runtime*. 14th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR), 2011.
- [HSB+12] Manfred Hild, Torsten Siedel, Christian Benckendorff, Christian Thiele, Michael Spranger: *Myon, a New Humanoid*. In: *Language Grounding in Robots*. Springer, New York, 2012.
- [Hur39] J. B. Hursh: *Conduction Velocity and Diameter of Nerve Fibers*. American Journal of Physiology. Ausgabe 127, Seiten 131–139, 1939.
- [IFR13] International Federation of Robotics: *World Robotics 2013 – Industrial Robots*. Executive Summary, 2013.
- [KBH11] Matthias Kubisch, Christian Benckendorff, Manfred Hild: *Balance Recovery of a Humanoid Robot Using Cognitive Sensorimotor Loops (CSLs)*. 14th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR), 2011.
- [KHK+08] Kenji Kaneko, Kensuke Harada, Fumio Kanehiro, Go Miyamori, Kazuhiko Akachi: *Humanoid Robot HRP-3*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2008.
- [Kin04] Clark Kinnaird: *Operating RS-485 Transceivers at Fast Signaling Rates*. Application Report, Texas Instruments, 2004.
- [Koe01] Teun Koetsier: *On the prehistory of programmable machines: musical automata, looms, calculators*. In: *Mechanism and Machine theory*. Band 36, Ausgabe 5, Seiten 589–603, 2001.
- [Koh73] Friedrich Kohl: *Geschichte der Jacquard-Maschine und der sich ihr anschließenden Abänderungen und Verbesserungen nebst der Biographie Jacquard's*. Nicolaische Verlags-Buchhandlung, Berlin, 1873.

- [KOK+74] Ichiro Kato, Sadamu Ohteru, Hiroshi Kobayashi, Katsuhiko Shirai, Akihiko Uchiyama: *Information-power machine with senses and limbs (Wabot 1)*. First CISM-IFTToMM Symposium on Theory and Practice of Robots and Manipulators. Springer, S. 11–24, 1974.
- [Kon02] Kurt Konolige: *Saphira Robot Control Architecture*. Technical Report, SRI International, Menlo Park, 2002.
- [KPO04] Jung-Yup Kim, Ill-Woo Park, Jun-Ho Oh: *Design and Walking Control of the Humanoid Robot, KHR-2 (KAIST Humanoid Robot 2)*. IEEE International Conference on Control, Automation and Systems (ICCAS), 2004.
- [Kug08] Thomas Kugelstadt: *The RS-485 Design Guide*. Application Report, Texas Instruments, 2008.
- [LH02] Gabriel Leen, Donal Heffernan: *TTCAN: a new time-triggered controller area network*. Microprocessors and Microsystems. Band 26, Ausgabe 2, Seiten 77–94, März 2002.
- [LLG+04] Sebastian Lohmeier, Klaus Löffler, Michael Gienger, Heinz Ulbrich, Friedrich Pfeiffer: *Computer System and Control of Biped Johnnie*. Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA), Vol. 4, 2004.
- [LT07] Hun-ok Lim, Atsuo Takanishi: *Biped walking robots created at Waseda University: WL and WABIAN family*. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 365.1850, Seiten 49–64, 2007.
- [MM05] Vladimír Mařík, Duncan McFarlane: *Industrial Adoption of Agent-Based Technologies*. IEEE Intelligent Systems. Band 20, Ausgabe 1, Seiten 27–35, Januar/Februar 2005.
- [MN10] Richard Michaelis, Gerhard W. Niemann: *Entwicklungsneurologie und Neuropädiatrie: Grundlagen und diagnostische Strategien*. Georg Thieme Verlag, 2010.
- [Mor07] Michael E. Moran: *Evolution of robotic arms*. Journal of Robotic Surgery. Band 1, Ausgabe 2, Juli 2007.

- [MP43] Warren S. McCulloch, Walter Pitts: *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics. Band 5, Ausgabe 4, Seiten 115–133, 1943.
- [MSV+08] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, Francesco Nori: *The iCub humanoid robot: an open platform for research in embodied cognition*. Proceedings of the 8th workshop on performance metrics for intelligent systems. Seiten 50–56, 2008.
- [Neu07] Peter Neumann: *Communication in industrial automation – What is going on?* Control Engineering Practice. Band 15, Ausgabe 11, Seiten 1332–1347, November 2007.
- [NKO+13] Keiji Nagatani, Seiga Kiribayashi, Yoshito Okada, Kazuki Otake, Kazuya Yoshida, Satoshi Tadokoro, Takeshi Nishimura, Tomoaki Yoshida, Eiji Koyanagi, Mineo Fukushima, Shinji Kawatsuma: *Emergency Response to the Nuclear Accident at the Fukushima Daiichi Nuclear Power Plants using Mobile Rescue Robots*. Journal of Field Robotics. Band 30, Ausgabe 1, Seiten 44–63, Januar/Februar 2013.
- [OAS+06] Yu Ogura, Hiroyuki Aikawa, Kazushi Shimomura, Hideki Kondo, Akitoshi Morishima, Hun-ok Lim, Atsuo Takanishi: *Development of a New Humanoid Robot WABIAN-2*. IEEE International Conference on Robotics and Automation (ICRA), 2006.
- [OC03] Anders Orebäck, Henrik I. Christensen: *Evaluation of Architectures for Mobile Robotics*. Autonomous Robots. Band 14, Seiten 33–49, 2003.
- [OG74] B. W. Ongerboer de Visser, C. Goor: *Electromyographic and reflex study in idiopathic and symptomatic trigeminal neuralgias: latency of the jaw and blink reflexes*. Journal of Neurology, Neurosurgery & Psychiatry. Band 37, Ausgabe 11, Seiten 1225–1230, 1974.
- [Pfe05] Friedrich Pfeiffer: *Entwurf und Realisierung einer zweibeinigen Laufmaschine*. In: *Autonomes Laufen*. Springer Berlin Heidelberg. Seiten 121–145, 2005.

- [PMV+06] Francesco De Pellegrini, Daniele Miorandi, Stefano Vitturi, Andrea Zanella: *On the Use of Wireless Networks at Low Level of Factory Automation Systems*. IEEE Transactions on Industrial Informatics. Band 2, Ausgabe 2, Mai 2006.
- [Poa02] Josh Poage: *The Development of the Jacquard Loom: Early History of Computer Data Storage*. 2002.
- [Pry08] Gunnar Prytz: *A performance analysis of EtherCAT and PROFINET IRT*. IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2008.
- [QGC+09] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng: *ROS: an open-source Robot Operating System*. ICRA Workshop on Open Source Software. Band 3, Ausgabe 3.2, 2009.
- [Reg10] Kristian Regenstien: *Modulare, verteilte Hardware-Software-Architektur für humanoide Roboter*. Dissertation, Fakultät für Informatik, Karlsruher Institut für Technologie, 2010.
- [RFY+90] T. Ryushi, T. Fukunaga, K. Yuasa, H. Nakajima: *The influence of motor unit composition and stature on fractionated patellar reflex times in untrained men*. European Journal of Applied Physiology and Occupational Physiology. Band 60, Ausgabe 1, Seiten 44–48, 1990.
- [Rob08] Robotis Co. Ltd.: *Dynamixel RX-28*. Datenblatt. Rev. 1.10, 2008.
- [SKB07] Heribert Stolzenberg, Heidrun Kahl, Karl E. Bergmann: *Körpermaße bei Kindern und Jugendlichen in Deutschland*. Bundesgesundheitsblatt – Gesundheitsforschung – Gesundheitsschutz. Band 50, Ausgabe 5–6, Seiten 686–700, Mai 2007.
- [STH10] Michael Spranger, Christian Thiele, Manfred Hild: *Integrating high level cognitive systems with sensorimotor control*. Advanced Engineering Informatics. Band 24, Ausgabe 1, Seiten 76–83, Januar 2010.
- [STM08] ST Microelectronics: *STM32F103x8, STM32F103xB Datasheet*. Datenblatt, 2008.

- [Thi07] Christian Thiele: *Integrierte Entwicklungsumgebung zur Bewegungssteuerung humanoider Roboter*. Studienarbeit, Institut für Informatik, Humboldt-Universität zu Berlin, 2007.
- [THS+12] Arndt von Twickel, Manfred Hild, Torsten Siedel, Vishal Patel, Frank Pasemann: *Neural control of a modular multi-legged walking machine: Simulation and hardware*. Robotics and Autonomous Systems. Band 60, Ausgabe 2, Seiten 227–241, Februar 2012.
- [UBL06] Heinz Ulbrich, Thomas Buschmann, Sebastian Lohmeier: *Development of the humanoid robot LOLA*. Applied Mechanics and Materials. Band 5, Seiten 529–540, Oktober 2006.
- [UZ07] Klaus Uhl, Marco Ziegenmeyer: *MCA2 – an extensible modular framework for robot control applications*. 10th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR), 2007.

Danksagung

Ich möchte allen danken, die mich bei der Erstellung dieser Arbeit unterstützt haben. Vor allem danke ich Herrn Dr. Manfred Hild für die Betreuung dieser Arbeit und die Tatsache, dass ich jederzeit mit Fragen und Problemen zu ihm kommen konnte. Auch meine beiden Gutachter Prof. Dr. Hans-Dieter Burkhard und Prof. Dr. Holger Schlingloff haben mir hilfreiche Hinweise zur Verbesserung der Arbeit gegeben.

Mein Dank gilt auch allen anderen, die mir inhaltliche Anregungen gegeben haben und denen, die diese Arbeit Korrektur gelesen haben, namentlich Katrin Stickel, Alexander Löwer, Marcus Janz, Christine Wunsch und Matthias Kubisch.

Dem gesamten Labor für Neurorobotik möchte ich für die großartige Arbeitsatmosphäre danken und dafür, dass ich bei der Entwicklung eines so wunderbaren Roboters wie dem Myon beteiligt sein durfte. Mein Dank gilt dabei neben dem Leiter des Labors, Dr. Manfred Hild, vor allem Torsten Siedel, Matthias Kubisch, Mario Weidner und Christian Benckendorff.

Einen ganz besonderen Dank schulde ich meinen Eltern, die mich immer unterstützt haben und eine unglaubliche Geduld mit mir hatten. Ihnen möchte ich diese Arbeit widmen.

Erklärungen

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Diplomarbeit in diesem Studiengang erstmalig einzureichen.

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 13. Juni 2014