JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät

Steuerung eines Sechsbeiners mit Linux-Echtzeitbetriebssystem und Einbindung eines Visual-Systems

PROJEKTSEMINAR

im Diplomstudium

Mechatronik

Eingereicht von:
Sebastian Brandstetter, Thomas Schöppl

Angefertigt am:
Institut für Robotik

Betreuung:
Univ.-Ass. Dipl.-Ing. K. Stadlbauer

Linz, November 2009

Inhaltsverzeichnis

Abbildungsverzeichnis

Teil I.

Modellbildung und Hardware

1 Modellbildung

1.1. Projektionsgleichung in Subsystemdarstellung

Die Modellbildung des Sechsbeiners erfolgt zweckmäßigerweise mithilfe der Projektionsgleichung in Subsystemdarstellung (siehe [Bre05]). Dabei handelt es sich um eine synthetische Methode, bei der die Bewegungsgleichung aus den Impuls- und Drallsätzen der Einzelkörper, welche in beliebigen Referenzsystemen dargestellt werden können, zusammengesetzt wird. Die Impuls- und Dralländerungen werden in Richtung der freien Bewegungsrichtungen projeziert und somit die vorherrschenden Bindungen berücksichtigt.

$$\sum_{i=1}^{N}\left[\left(\frac{\partial \mathbf{v}_s}{\partial \dot{\mathbf{s}}}\right)^T\left(\frac{\partial \omega_s}{\partial \dot{\mathbf{s}}}\right)^T\right]_i\left[\begin{matrix}\dot{\mathbf{p}}+\tilde{\omega}_R\mathbf{p}-\mathbf{f}^e\\ \dot{\mathbf{L}}+\tilde{\omega}_R\mathbf{L}-\mathbf{M}^e\end{matrix}\right]_i=0 \tag{1.1}$$

Als nächsten Schritt empfiehlt es sich eine Aufteilung in immer wiederkehrende Baugruppen, sogenannte Subsysteme, vorzunehmen.

$$\sum_{n=1}^{N_{sub}}\left\{\sum_{i=1}^{N_n}\left[\left(\frac{\partial \mathbf{v}_s}{\partial \dot{\mathbf{s}}}\right)^T\left(\frac{\partial \omega_s}{\partial \dot{\mathbf{s}}}\right)^T\right]_i\left[\begin{matrix}\dot{\mathbf{p}}+\tilde{\omega}_R\mathbf{p}-\mathbf{f}^e\\ \dot{\mathbf{L}}+\tilde{\omega}_R\mathbf{L}-\mathbf{M}^e\end{matrix}\right]_i\right\}=0 \tag{1.2}$$

Mit einer entsprechenden Nachdifferenzierung können jetzt den einzelnen Subsystemen eigene Geschwindigkeitsparameter $\dot{\mathbf{y}}_n$ (beschreibende Geschwindigkeiten) zugewiesen werden.

$$\sum_{n=1}^{N_{sub}}\left(\frac{\partial \dot{\mathbf{y}}_n}{\partial \dot{\mathbf{s}}}\right)^T\left\{\sum_{i=1}^{N_n}\left[\left(\frac{\partial \mathbf{v}_s}{\partial \dot{\mathbf{y}}_n}\right)^T\left(\frac{\partial \omega_s}{\partial \dot{\mathbf{y}}_n}\right)^T\right]_i\left[\begin{matrix}\dot{\mathbf{p}}+\tilde{\omega}_R\mathbf{p}-\mathbf{f}^e\\ \dot{\mathbf{L}}+\tilde{\omega}_R\mathbf{L}-\mathbf{M}^e\end{matrix}\right]_i\right\}=0 \tag{1.3}$$

Dadurch können die Subsysteme zunächst unabhängig vom Gesamtsystem aufgestellt werden, in das sie später eingesetzt werden. Die beschreibenden Geschwindigkeiten können mithilfe eindeutiger Funktionalmatrizen auf die Minimalkoordinaten abgebildet werden, und teilen sich auf in einen Führungs- und einen Relativanteil. Die Führungsgeschwindigkeiten werden dabei durch eine Kette von Vorgängersystemen (kinematische Kette), beginnend beim Inertialsystem, bestimmt, während der Relativanteil ausschließlich vom jeweiligen Subsystem selbst bestimmt wird. Die Kombination der beiden Anteile bestimmen wiederum die Führungsgeschwindigkeiten des Nachfolgesystems, falls vorhanden.

1.2. Ermitteln der Subsystemgleichungen

Zu Beginn werden für das jeweilige Subsystem die beschreibenden Geschwindigkeiten $\dot{\mathbf{y}}_n$ definiert, die sich aus den Führungsgeschwindigkeiten $\mathbf{v}_f$, den Führungswinkelgeschwindigkeiten ω_f und den Relativgeschwindigkeiten $\dot{\mathbf{q}}_{rel}$ zusammensetzen. Im nächsten Schritt werden die Drehmatrizen zwischen den körperfesten Koordinatensystemen und dem Führungssystem $\mathbf{A}_{i,f}$, sowie die Drehmatrizen zwischen zwei Körpern $\mathbf{A}_{i,p}$ aufgestellt. Der Index i bezeichnet hierbei

den aktuellen Körper, während p für dessen Vorgänger steht. Im Anschluss werden die Schwerpunktsvektoren der einzelnen Starrkörper, die Verbindungsvektoren zwischen den Körpern, sowie der Ortsvektor zum Anschlusspunkt des nachfolgenden Subsystems aufgestellt. Jetzt können die absoluten Geschwindigkeiten und Winkelgeschwindigkeiten angegeben werden. Diese können im körperfesten Koordinatensystem angeschrieben werden und errechnen sich aus

$$\boldsymbol{\omega}_i = \mathbf{A}_{i,p}\boldsymbol{\omega}_p + \boldsymbol{\omega}_{i,rel} \tag{1.4}$$

$$\mathbf{v}_i = \mathbf{A}_{i,p}\mathbf{v}_p + \tilde{\boldsymbol{\omega}}_i\mathbf{r}_i + \mathbf{v}_{i,rel} \tag{1.5}$$

Analog dazu werden die Absolutgeschwindigkeiten für die Anschlusspunkte von nachfolgenden Körpern und Subsystemen berechnet.
Im nächsten Schritt werden die Funktionalmatrizen $\overline{\overline{\mathbf{F}}}_i$ der einzelnen Körper durch partielles Ableiten der eben ermittelten Absolutgeschwindigkeiten nach den Subsystemkoordinaten $\dot{\mathbf{y}}_n$ ermittelt. Ausserdem werden die Zeitableitungen der Funktionalmatrizen $\dot{\overline{\overline{\mathbf{F}}}}_i$ gebildet. Im Unterschied zu [Bre05] werden als Zwischenvariablen $\dot{\overline{\mathbf{y}}}_n$ die Subsystemkoordinaten gewählt, wodurch die Funktionalmatrizen

$$\overline{\mathbf{F}}_i = \frac{\partial \dot{\overline{\mathbf{y}}}_n}{\partial \dot{\mathbf{y}}_n} \tag{1.6}$$

zu Einheitsmatrizen degenerieren und kein weiteres Nachdifferenzieren notwendig ist.
Anschließend müssen noch die Massenmatrizen $\overline{\overline{\mathbf{M}}}_i$ und die Gyromatritzen $\overline{\overline{\mathbf{G}}}_i$ der Einzelkörper aufgestellt werden. Aufgrund der Wahl von körperfesten Koordinatensystemen sind die Massenmatrizen zeitlich konstant und haben die Form

$$\overline{\overline{\mathbf{M}}}_i = \begin{bmatrix} m_i\mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_i^S \end{bmatrix}, \tag{1.7}$$

wobei m_i der Masse und $\mathbf{J}_i^S$ dem Trägheitstensor des jeweiligen Körpers entspricht. Für die Gyromatrix gilt der Zusammenhang

$$\overline{\overline{\mathbf{G}}}_i = \begin{bmatrix} \tilde{\boldsymbol{\omega}}_i & \mathbf{0} \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_i \end{bmatrix} \overline{\overline{\mathbf{M}}}_i. \tag{1.8}$$

Bevor das Subsystem nun zusammengesetzt werden kann müssen noch die Vektoren der verallgemeinerten eingeprägten Kräfte aufgestellt werden. Diese setzen sich aus den Momenten und den, in den Schwerpunkt reduzierten, angreifenden Kräften nach der Formel

$$\overline{\overline{\mathbf{Q}}}_i^e = \begin{bmatrix} \mathbf{f}_{S,i}^e \\ \mathbf{M}_i^e \end{bmatrix} \tag{1.9}$$

zusammen. Abschließend kann das Subsystem mithilfe der Funktionalmatrizen $\overline{\overline{\mathbf{F}}}_i$ und $\dot{\overline{\overline{\mathbf{F}}}}_i$ aus den Einzelkörpern nach den Gleichungen

$$\mathbf{M}_n = \sum_{i=1}^{N_n} \overline{\overline{\mathbf{F}}}_i^T \overline{\overline{\mathbf{M}}}_i \overline{\overline{\mathbf{F}}}_i \tag{1.10}$$

$$\mathbf{G}_n = \sum_{i=1}^{N_n} \overline{\overline{\mathbf{F}}}_i^T \left(\overline{\overline{\mathbf{M}}}_i \dot{\overline{\overline{\mathbf{F}}}}_i + \overline{\overline{\mathbf{G}}}_i \overline{\overline{\mathbf{F}}}_i\right) \tag{1.11}$$

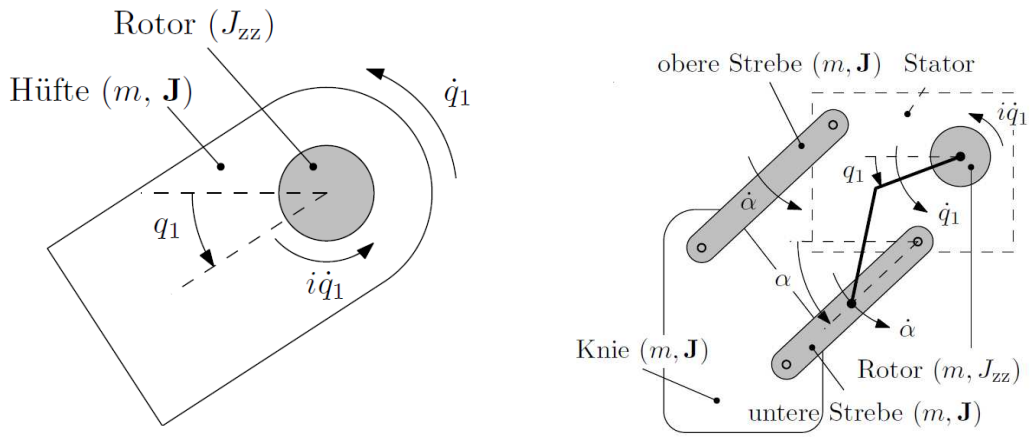
$$\mathbf{Q}_n^e = \sum_{i=1}^{N_n} \overline{\overline{\mathbf{F}}}_i^T \overline{\overline{\mathbf{Q}}}_i^e \tag{1.12}$$

zusammengesetzt werden.

1.3. Verwendete Subsysteme

Der gesamte Roboter wurde in folgende Subsysteme unterteilt (s. u. a. [Kas06]):

- **Starrkörper** zur Modellierung des Chassis
- **Translation**
- **Rotation**
- **Motor-/Armeinheit** zur Modellierung des Hüftgelenks
- **Schenkeleinheit** zur Modellierung von Ober- und Unterschenkel



(a) Motor-/Armeinheit

(b) Schenkeleineinheit

Bild 1.1.: Subsysteme

Aus diesen Subsystemen wird das Gesamtsystem, wie in Abb. 1.2 dargestellt, zusammengesetzt. Der Rest der Modellbildung wurde im Rahmen dieser Projektarbeit nicht weiter behandelt und ist der Diplomarbeit [Kas06] zu entnehmen.

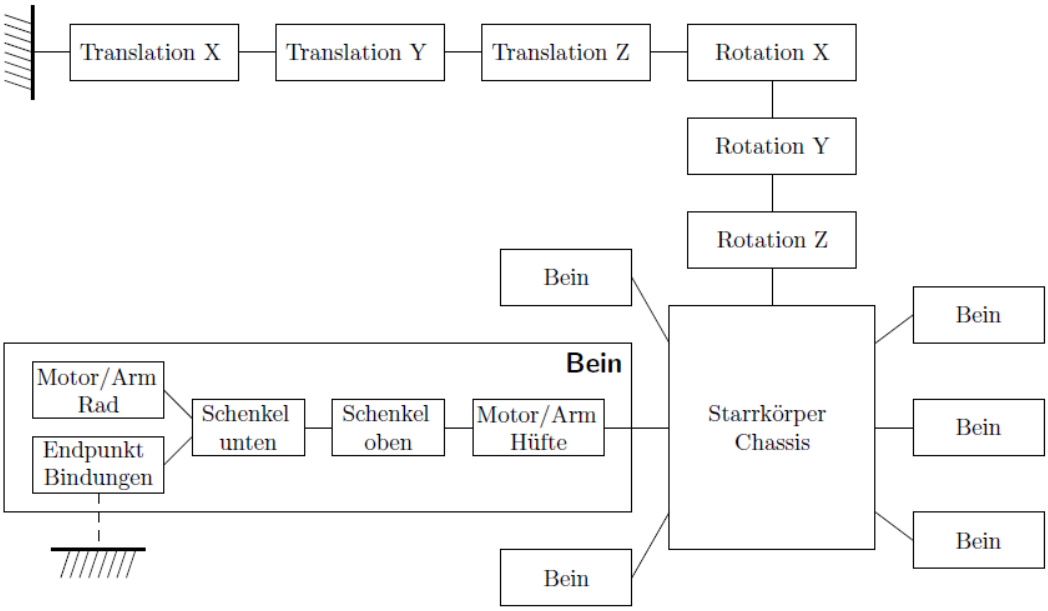


Bild 1.2.: Gesamtsystem des Roboters

2 Übersicht Boardelektronik

Die Boardelektronik ist für die Generierung und Überwachung der benötigten Stellgrößen in Abhängigkeit von verschiedenen Eingangsgrößen verantwortlich. Für diese Aufgabe stehen ihr eine Vielzahl an Komponenten (z.B. Recheneinheit, Aktoren, Sensoren, usw.) zur Verfügung. Eine Aufgabenstellung dabei ist, eine Boardelektronik zu entwerfen, welche spätere Weiterentwicklungen vor allem im Bereich der Software ermöglicht. Es sollte auch möglich sein, neue Hardwarekomponenten, ohne viel Aufwand betreiben zu müssen, nachträglich hinzufügen zu können. Das entwickelte Konzept mit den einzelnen Teilsystemen ist in Abb. 2.1 dargestellt.
Um ausreichend Rechenleistung, auch für softwaretechnische Weiterentwicklungen zur Verfügung zu haben, wird für die Recheneinheit das Embedded Board NANO-945GSE-N270 (EPIC SBC) von IEI Technology (s. Kap. 3 auf S. 14) gewählt. Als Betriebssystem wurde ein Linux Minimal Filesystem (s. Kap. 9.3 auf S. 43) installiert.
Für den drahtlosen Datenverkehr zwischen einem PC und dem Embedded Board stehen zwei Varianten, Bluetooth und WLAN, welche gängige Laptops integriert haben, zur Auswahl. In dieser Arbeit wird die WLAN-Verbindung wegen der höheren Reichweite und höheren Datenübertragungsrate der Bluetooth-Verbindung vorgezogen.
Für die Kommunikation zwischen dem Embedded Bord und den einzelnen Beineinheiten wird das BIOE-System (s. Kap. 4 auf S. 16) herangezogen. An diesem BIOE-Bus können bis zu 16 BIO-Elemente angeschlossen werden. Da zur Zeit nur 6 BIO-Elemente benötigt werden, ist auch hier eine Erweiterung leicht möglich.
Die Beineinheit (s. Kap. 5 auf S. 20) hat die Aufgabe die Positionssignale von den Servos so aufzubereiten, damit sie dann über einen AD-Wandler auf den BIO-Elementen weiterverarbeitet werden können. Ebenfalls auf der Beineinheit ist die Ansteuerung für einen Gleichstrommotor integriert. Für die Feststellung der Position relativ zu einem fixen Koordinatenpunkt im Raum wird die Kamera Logitech 9000 Pro verwendet. Diese Kamera wurde am Institut schon mehrmals erfolgreich eingesetzt.
Für die Energieversorgung werden zwei LI-FE Akkumulatoren (s. Kap. 6.1 auf S. 24) gewählt. Diese zeichnen sich durch ihre kompakte Bauart und hohe Energiedichte aus.

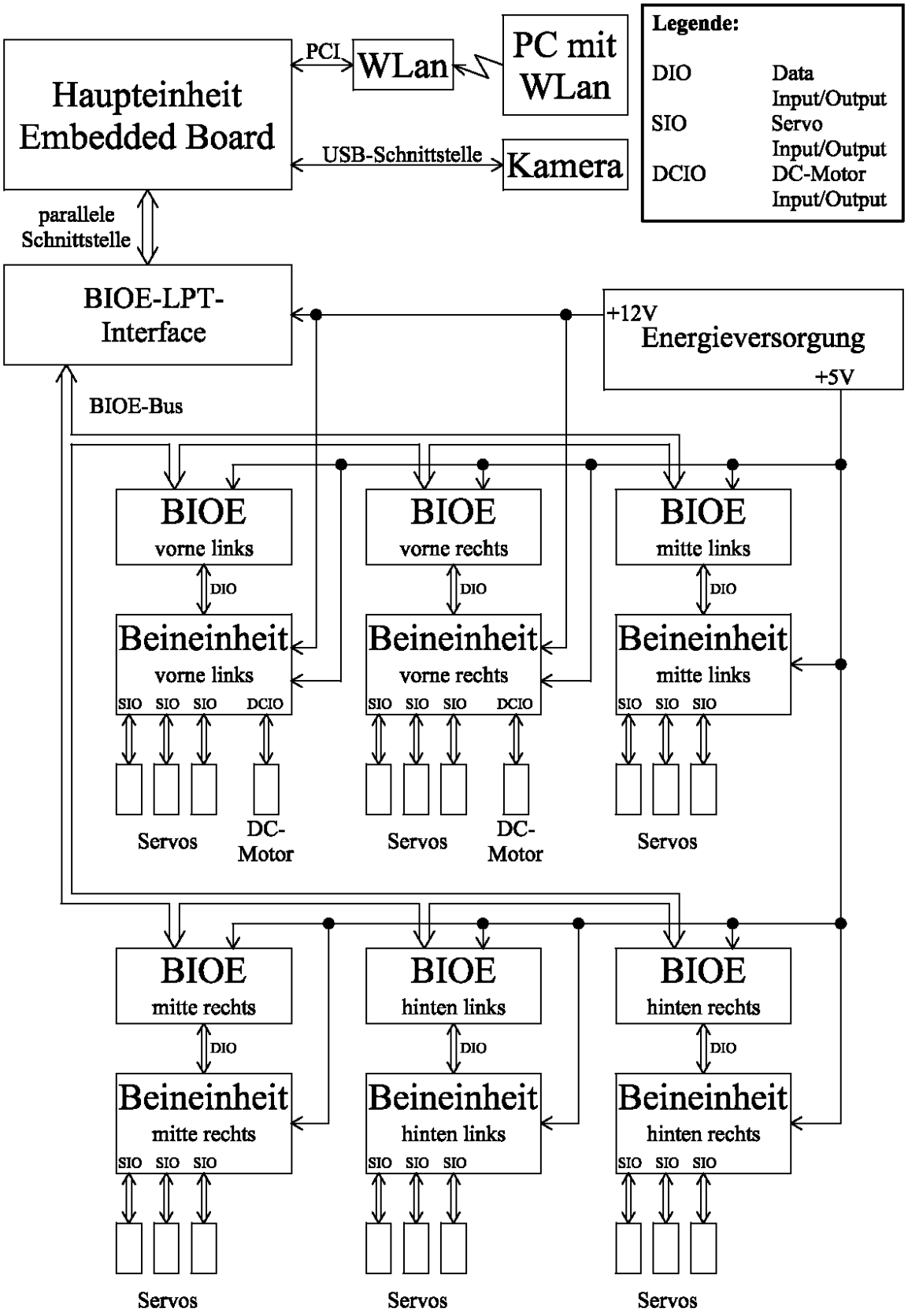


Bild 2.1.: Übersicht der Boardelektronik

3 Embedded Board

Als Embedded Board wird das NANO-945GSE-N270 (EPIC SBC) von IEI Technology gewählt. Dieses Board basiert auf dem stromsparenden, aber leistungsstarken Intel Atom N270 1.6 GHz Prozessor. Dieser Prozessor bietet genug Rechenleistung für das Steuerprogramm des Sechsbeiners und bietet noch zusätzliche Reserven für eventuelle Weiterentwicklungen. Ausserdem zeichnet er sich durch eine geringe Verlustleistung von nur 2,5W auf. Das Embedded Board ist ebenfalls für geringen Stromverbrauch ausgelegt und benötigt keinen aktiven Kühler für den Betrieb. Das Modell NANO-ATOM benötigt eine 12V Gleichstromversorgung und besitzt dadurch ideale Voraussetzungen für unser autonomes System.

Als Speicher für das Betriebssystem und für das Steuerprogramm wird eine 4 GB CF-Karte verwendet. Durch die Größe der Speicherkarte ist es möglich, eine komplette Linux-Distribution auf dem Embedded Board laufen zu lassen. Aus persönlichem Interesse fällt die Wahl des Betriebssystem auf ein Linux Minimal Filesystem. Um dieses von der CF-Karte beim Bootvorgang laden zu können, muss diese als Master definiert werden. Dazu muss der Jumper JCF1 (siehe Abb. 3.1) gesetzt werden. Bei offenem Jumper (Standardeinstellung) ist die CF-Karte als Slave konfiguriert.

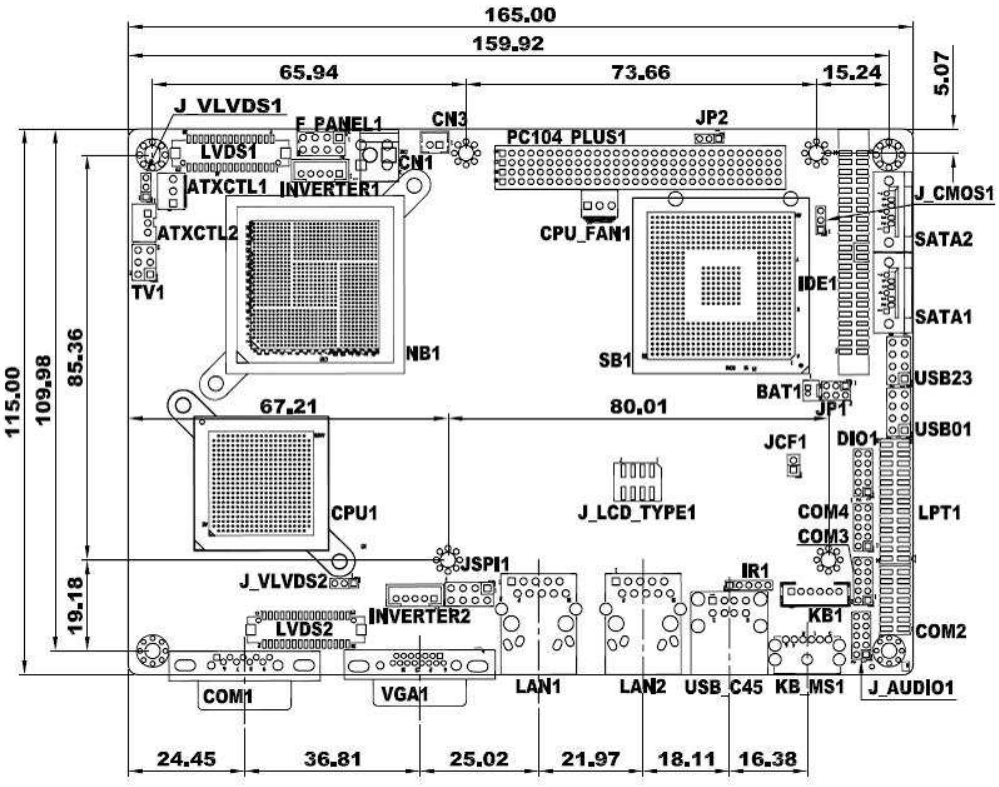


Bild 3.1.: Embedded Board

Um eine Verbindung zwischen dem BIOE-LPT-Interface, welches einen Standard LPT-Anschluss besitzt, und dem Parallelport des Embedded Board (siehe Abb. 3.2) herstellen zu können, muss ein Adapter entsprechender gebaut werden.

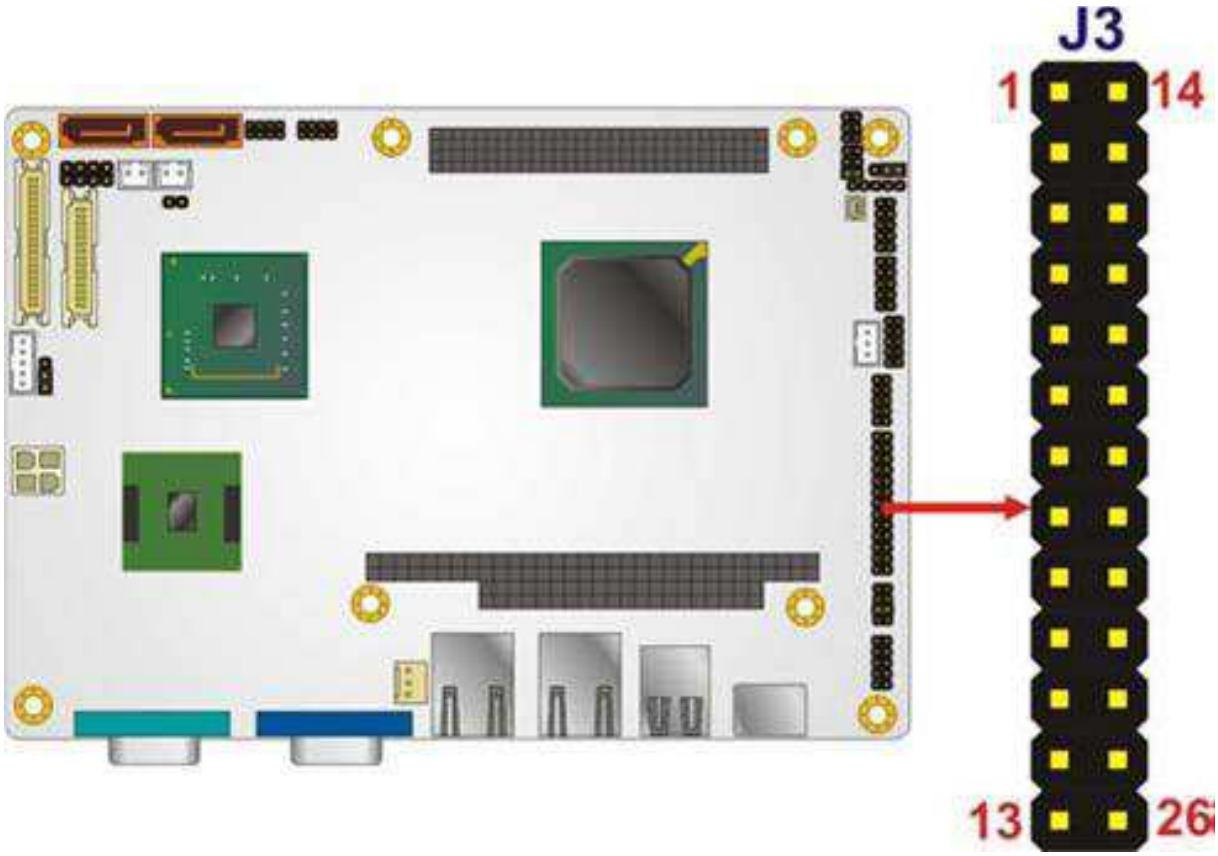


Bild 3.2.: Parallelport am Embedded Board

4 BIOE-System

Die Kommunikation zwischen dem Embedded Board und den einzelnen Beineinheiten erfolgt durch das Basic Input/Output Elements-System, kurz BIOE-System. Es besteht aus dem BIOE-LPT-Interface, welches an der parallelen Druckerschnittstelle angeschlossen wird. Dieses ist über ein 4bit paralleles Interface mit den einzelnen BIO-Elementen verbunden. Hier können bis zu 16 Einheiten angeschlossen werden, welche seriell angesprochen werden. Dies kann bei einer hohen Anzahl von BIO-Elementen und sehr kurzer Abtastzeit zu Problemen führen.
Das fertige BIOE-System besitzt bereits einige implementierte Funktionen. Diese können über den DTN-Parameter ausgewählt werden. Hier werden zum Beispiel bereits AD-Wandler oder Ansteuerungen für Servomotoren zur Verfügung gestellt. Weitere Informationen zu diesen Konfigurationen oder über das BIOE-System sind [Wei08] zu entnehmen.

4.1. Funktion der BIO-Elemente

Pro Beineinheit müssen drei Servomotoren angesteuert werden und die von ihnen eingehenden analogen Positionssignale in ein digitales Signal umgewandelt werden. Für eine ausreichende Auflösung des analogen Signals wurde ein 10bit AD-Wandler verwendet. Außerdem wird ein PWM-Signal und 2 Signalleitungen, die zwischen $0V$ und $5V$ geschalten werden können, für die Ansteuerung der H-Brücke, benötigt. Um die Signale vom Impulsgeber des Gleichstrommotor verarbeiten zu können, muss zusätzlich ein Positionsdecoder auf dem BIO-Element implementiert werden. Diese aufgezählten Funktionen sind bereits jedes für sich implementiert und können durch den dazugehörigen DTN-Parameter aufgerufen werden. Um jedoch ein effektives System zu bekommen und die Anzahl der BIO-Elemente so gering als möglich zu halten, ist es notwendig alle diese Funktionen auf ein solches zu implementieren. Diese Aufgabenstellung wurde mit Herrn DI Weichinger Klaus besprochen. Er stellte dann eine Funktion für die BIO-Elemente zusammen, welches die benötigten Anforderungen erfüllte.

Im Listing 4.1 ist die fertige Konfiguration eines BIO-Elements dargestellt, welche für den Sechsbeiner verwendet werden. Diese wird mit der DTN-Nummer 106 aktiviert.

Listing 4.1: Belegung der BIO-Elemente

```
1  ;=============================================
2  ; DTN106 - Universalansteuerung
3  ;          4x10bit ADC
4  ;          4xModellbauservo
5  ;          1xMotoransteuerung für H-Brücke
6  ;          1xInkrementaldekoder
7  ;=============================================
8  ;
9  ;  +-------------------------------------+
10 ;  |DTN 106                        AREF+-
11 ;  |      IN           OUT         P15 +- ADC4 In
```

```
12 ;   | EP2                SERVO1      P14 +- ADC3 In
13 ;   | EP3                SERVO2      P13 +- ADC2 In
14 ;   | EP4                SERVO3      P12 +- ADC1 In
15 ;   | EP5                SERVO4      P11 +- RESET
16 ;   | EP6    ADC1                    P10 +- B
17 ;   | EP7    ADC2                    P9  +- A
18 ;   | EP8    ADC3                    GND +-
19 ;   | EP9    ADC4                    P8  +- OUT1-B
20 ;   | EP10               MOT         P7  +- OUT1-A
21 ;   | EP11   POS                     RES +-
22 ;   | EP12   SER0                    P6  +- LED (ADC-Sample Impuls Output
       )
23 ;   | EP13   SER1                    VCC +-
24 ;   | EP14                           P5  +- SERIAL-IN (in Planung)
25 ;   |                                P4  +- SERVO4
26 ;   |                                P3  +- SERVO3
27 ;   |                                P2  +- SERVO2
28 ;   |                                P1  +- SERVO1
29 ;   |                                P0  +- PWM1
30 ;   +----------------------------------------+
```

Ansteuerung der Servomotoren

Jede Beineinheit besteht aus drei Servomotoren, jedoch wurde in dieser Funktion, um Reserven für Erweiterungen zu haben, die Ansteuerung und Digitalisierung für vier Servomotoren implementiert. Modellbauservos, wie sie beim Sechsbeiner verwendet werden, benötigen ein Impulssignal mit einer Periodendauer von $20ms$ mit einer High-Phase zwischen $1ms$ bis $2ms$. Dabei bestimmt die Dauer der High-Phase direkt die Position des Servomotor. An den ServoX-Ausgängen wird ein solches Signal generiert. Um die Dauer der High-Phase zu beeinflussen werden Zahlenwerte zwischen 0 und 39936 an das BIO-Element übermittelt. Dabei entspricht die Zahl 0 einer High-Phase von $0ms$ und die Zahl 39936 einer High-Phase von $2ms$.

AD-Wandler

Die analogen Positionssignale, welche einem Spannungswert zwischen $0V$ und $5V$ entsprechen, werden von den 10bit AD-Wandlern in einen Zahlenwerte zwischen 0 und 1023 konvertiert. Dabei entspricht $0V$ dem digitalen Wert 0 und $5V$ dem digitalen Wert 1023.

Ansteuerung der Gleichstrommotoren

Um die H-Brücke für die Gleichstrommotoren steuern zu können, benötigt man drei Ausgangssignale. Diese sind das PWM-Siganl und die beiden Ausgänge OUT1-A und OUT1-B. Dabei werden die beiden Ausgänge OUT1-A und OUT1-B zwischen den technischen Werten 0 und 1 geschalten. Diese drei Ausgänge werden mit dem Eingang MOT angesteuert. Der Wert für diesen Eingang setzt sich aus der Formel

$$MOT = PWM1 + (OUT1-A) * 256 + (OUT1-B) * 512 \tag{4.1}$$

zusammen. Die Variablen in der Formel 4.1 können dabei die Werte

$$PWM1 = [0, ..., 255] \tag{4.2}$$
$$OUT1 - A = [0, 1] \tag{4.3}$$
$$OUT1 - B = [0, 1] \tag{4.4}$$

annehmen. Dabei wird mit der Variable PWM1 die Geschwindigkeit festgelegt und mit den beiden anderen Variablen die Drehrichtung des Gleichstrommotors. In der Abb. 5.2 in Kap. 5 auf S. 21 ist die Belegung der einzelnen Variablen für die verschiedenen Ansteuerungsmöglichkeiten der H-Brücke genau beschrieben.

Positionsdekoder

Um die Position bzw. die Geschwindigkeit der Gleichstrommotoren messen zu können, müssen die beiden Signale vom Impulsgeber ausgewertet werden. An den beiden Eingängen A und B werden die beiden Rechtecksignale des Impulsgeber angeschlossen. Dabei wird auf jede steigende und fallende Flanke auf B gezählt. Mittels des Signals von A wird die Zählrichtung, also die Drehrichtung des Gleichstrommotors, festgestellt. Im BIO-Element ist ein 16bit Zähler implementiert. Der Startpunkt des Zählers liegt bei 32768 und kann mit einem HIGH-Signal auf dem RESET Eingang wieder auf diesen Startpunkt zurückgesetzt werden. Der Impulsgeber sendet pro voller Umdrehung der Abtriebswelle 3936 Impulse. Da bei dem implementierten Inkrementaldecoder auf die fallende und steigende Flanke gezählt wird, ergeben sich nun 7872 Punkte pro Umdrehung. Dadurch stellt sich das Problem, dass der Zähler schnell an seine grenzen gelangt und ein Überlauf stattfindet. Um jedoch die Geschwindigkeit des Gleichstrommotor berechnen zu können benötigt man ein kontinuierliches stetiges Signal. Dieses Problem wurde softwaretechnisch durch Mitzählen der vollen Umdrehungen und anschließendes Addieren der entsprechenden Impulszahl gelöst.

4.2. Programmierung der BIO-Elemente

Um die BIO-Elemente verwenden zu können, müssen diese programmiert werden. Es werden im Internet viele Programmierer zum Selbstbau angeboten. Für diese Aufgabenstellung wurde ein ISP-Programmierer wie er z.B. im Internet[1] zu finden ist, verwendet. Als Nächstes benötigt man noch eine Software zum Programmieren des Mikrocontrollers. Auch hier hat man eine große Auswahl, wobei die meisten als Freeware im Internet zum Herunterladen sind. Die Programmierung wird mit dem Programm *AVRDUDE*, welches am Institut vorhanden ist und auf der beiliegenden DVD[2] zu finden ist, durchgeführt. Die Installation des Programms ist in der Datei *Readme.txt* im Ordner AVR-Programm genau beschrieben.
Um nun die gewünschte Datei auf das BIO-Element überspielen zu können, muss der ISP-Programmierer mit dem BIO-Element wie in Abb. 4.1 dargestellt ist, verbunden werden. Dabei ist zu beachten, dass das BIO-Element beim Programmiervorgang mit $5V$ Spannung versorgt werden muss.

[1] http://8051help.mcselec.com/index.html?stk200300ispprogrammer.htm
[2] DVD:/AVR-Programm/

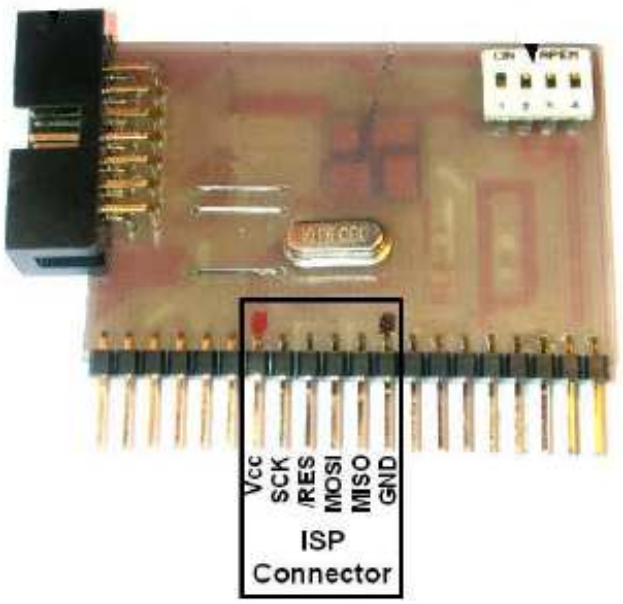


Bild 4.1.: Anschluss des Programmierers an das BIO-Element

Die Eingabe der Befehle zum Programmieren des Mikrocontrollers erfolgt über die Kommandozeile. Als erstes müssen die Fuse-Bytes richtig gesetzt werden, welche die Eigenschaften des Controllers bestimmen. Sie werden mit dem Befehl

```
avrdude -p m16 -c stk200 -P lpt1 -U lfuse:w:0xFF:m
        -U hfuse:w:0xC9:m
```

programmiert. Diese Einstellung für das *lfuse* und *hfuse* bewirken, dass der Mikrocontroller einen externen Takt, der von einem Quarz erzeugt wird, verwendet. Im nächsten Schritt wird die Firmware auf den Mikrocontroller übertragen werden. Dies wird mit den Befehl

```
avrdude -p m16 -c stk200 -P lpt1 -U flash:w:BIOE01.hex
```

durchgeführt. Dabei muss die Firmware *BIOE01.hex* in dem selben Ordner abgelegt sein, in dem sich der Befehl *AVRDUDE* befindet.
Bei beiden Befehlen benötigt *AVRDUDE* einige Parameter. Diese beginnen immer mit einem Minuszeichen gefolgt von einem Buchstaben. Dabei ist auf die Groß- und Kleinschreibung der Parameter zu achten. Die bei diesen Befehlen verwendeten Parameter haben folgende Bedeutung bzw. Aufgaben:

-p ... legt den Mikrocontroller fest, z.B.: der ATmega16 heißt m16

-c ... gibt an welcher Programmierer verwendet wird

-P ... bestimmt den Port über den der Programmierer am PC angeschlossen ist

-U ... Parameter zur Durchführung von Speicheroperationen

Eine vollständige Beschreibung und Hilfe zu dem Programm *AVRDUDE* bzw. zu den verschiedenen Parametern ist auf der Internetseite[3] zu finden.

[3] http://www.wiki.elektronik-projekt.de/mikrocontroller/avr/avrdude

5 Beineinheit

Die Beineinheit ist das Bindeglied zwischen den BIOEs und den Servos mit Sensor und den Gleichstrommotoren. Die Steuerungssignale für die Servos werden direkt vom BIOE (s. Kap. 4 auf S. 16) generiert und von der Beineinheit nur direkt weitergeleitet.

5.1. Aufbereitung der Positionssignale

Um bei den Servomotoren die aktuelle Position messen zu können, werden diese nach [Kas06] modifiziert. Um das analoge Positionssignal von den Servomotoren in ein digitales Signal wandeln zu können, muss es noch speziell aufbereitet werden. Die dafür benötigte Schaltung (s. Abb. 5.1) wurde bereits bei der Diplomarbeit [Kas06] entwickelt und wird in die Beineinheit integriert.

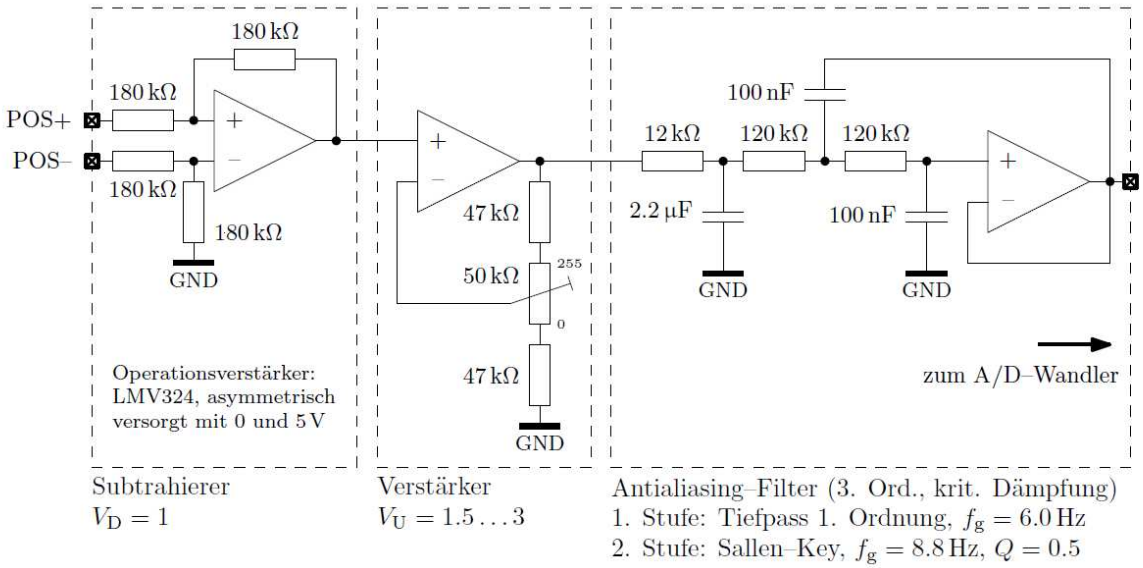


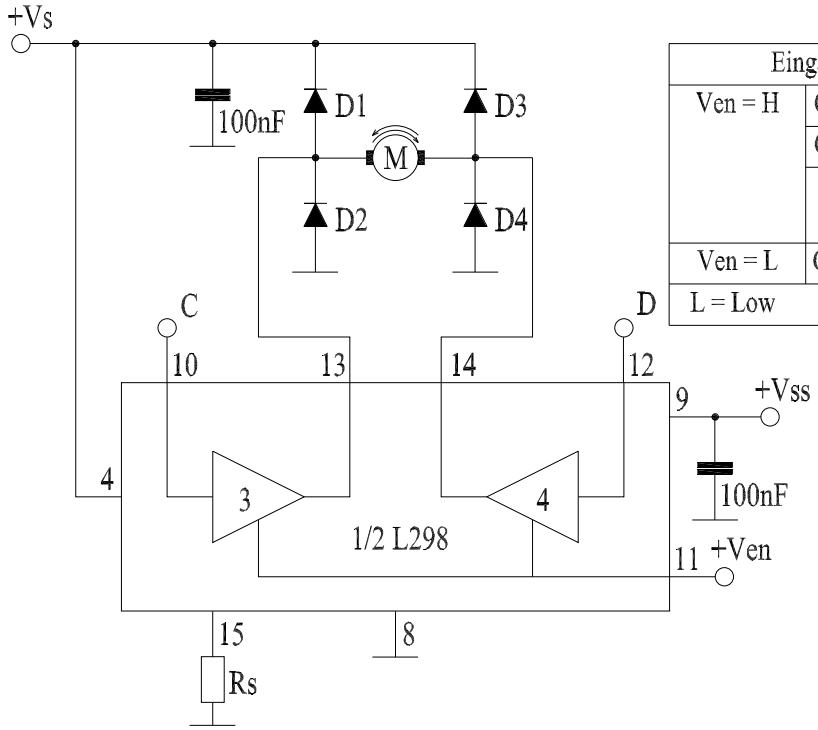
Bild 5.1.: Aufbereitung des Positionssignals für die A/D-Wandlung

Bei der ersten Stufe in Abb. 5.1 handelt es sich um eine Subtrahiererstufe mit einer Verstärkung von 1. Dieser Differenzverstärker wird dafür benötigt, die Positionsspannung auf das Massepotential zu beziehen, um es weiterverarbeiten zu können. Außerdem wird durch einen hohen Eingangswiderstand, der durch die externe Beschaltung bestimmt wird, eine zu starke Belastung der Servomotoren verhindert. Dies könnte sonst zu einer Verfälschung der internen Regelung der Servomotoren führen.
Als nächste Stufe folgt ein nichtinvertierender Verstärker. Mit diesem Schaltungsteil wird das Positionssignal so verstärkt, dass der gesamte Wertebereich des A/D-Wandler ausgenutzt werden kann und dadurch eine höhere Auflösung erzielt wird.
Die letzte Stufe besteht aus einem Tiefpassfilter welches Störsignale unterdrücken soll. Dabei

handelt es sich um ein Filter mit kritischer Dämpfung und einer Grenzfrequenz von ca. 8Hz. Nach dieser Aufbereitung des Signals kann es vom A/D-Wandler, der auf dem BIOE (s. Kap. 4 auf S. 16) implementiert wurde, verarbeitet werden.

5.2. Ansteuerung der Gleichstrommotoren

Die zweite Aufgabe der Beineinheit ist es, die Gleichstrommotoren für den Fahrantrieb anzusteuern. Dazu wird eine H-Brücke verwendet wie sie in Abb. 5.2 dargestellt ist.



Eingang		Funktion
Ven = H	C = H; D = L	Vorwärtslauf
	C = L; D = H	Rückwärtslauf
	C = D	schneller Motor Stop
Ven = L	C = X; D = X	Freilauf
L = Low	H = High	X = egal

Bild 5.2.: H-Brücke für die Ansteuerung des Fahrantriebes

Als Treiber für die H-Brücke wird der Baustein *L298* verwendet. Dieser Baustein enthält zwei Treiberstufen die jeweils mit 2*A* belastet werden können. Die V_{EN}-Leitung wird an den PWM-Anschluss gelegt und dadurch kann die Geschwindigkeit der Motoren gesteuert werden. Das dafür benötigte PWM-Signal wird von der BIOE-Einheit (s. Kap.4 auf S. 16) generiert. Mit der unterschiedlichen Polarität der beiden Anschlüsse C und D wird die Drehrichtung des Gleichstrommotor festgelegt. Liegen beide Eingänge auf gleichem Potential werden die Moteranschlüsse kurzgeschlossen und dadurch elektrisch gebremst. Der Widerstand R_S wird nicht verwendet, statt dessen wird der Anschluss 15 direkt auf Masse gelegt. Über diesen Anschluss fließt der gesamte Strom. Wird hier ein Hochlastwiderstand angeschlossen, so kann aus der abfallenden Spannung der Strom berechnet werden und dieser für eine Stromregelung verwendet werden. Dies ist jedoch bei der Anwendung für den Fahrantrieb des Sechsbeiner nicht notwendig.
Um eine Überbelastung des Bausteins *L298* zu vermeiden, werden zwei Treiber für die H-Brücken parallel geschaltet. Diese Parallelschaltung ist in Abb. 5.3 dargestellt.

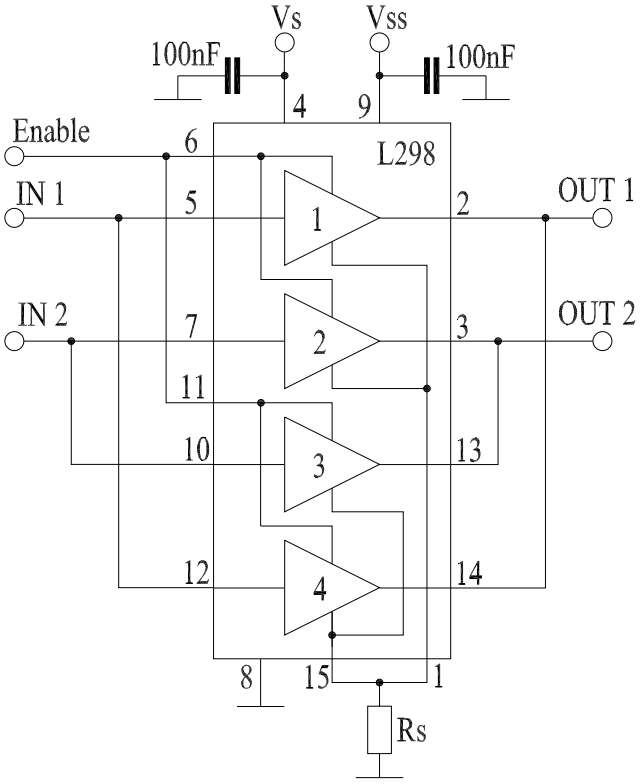


Bild 5.3.: Parallelschaltung von zwei Brückentreiber

Für die aktuelle Positionsbestimmung wird ein am Gleichstrommotor montierter Impulsgeber (s. Kap. 8 auf S. 34) verwendet. Die beiden Signalleitungen werden von der Beineinheit weiter an die BIOE-Einheit geleitet, wo die Auswertung der Signale durchgeführt wird.

6 Energieversorgung

Für die Stromversorgung muss eine Lösung erarbeitet werden, die mehrere Bedingungen zu erfüllen hat. Die wichtigste Bedingung ist eine autonome Versorgung herzustellen.
Eine weitere ist eine konstante Spannungsversorgung für alle Komponenten, die aber auch genügend Strom für die Servomotoren liefern kann. Für die Abschätzung des Stromverbrauches wird für die Servomotoren ein Spitzenstrom von 0,5A angenommen. Das Embedded Board hat laut Datenblatt einen Stromverbrauch von 1A, und für die sechs Platinen der Beinkontrolle und den sechs BIOEs wird ebenfalls 1A Stromverbrauch angenommen. Daraus ergibt sich ein Gesamtverbrauch von 11A.
Für das Embedded Board und für den Fahrantrieb muss eine Versorgungsspannung von 12V und für die restlichen Komponenten eine Versorgungsspannung von 5V bereitgestellt werden. Die Vorgabe dabei besteht darin, dies durch eine zentrale Stromversorgung zu ermöglichen. Außerdem muss gewährleistet werden, dass bei Inbetriebnahme des Sechsbeiners zuerst das Embedded Board mit den BIOEs hochgefahren wird. Die Servomotoren dürfen erst ab dem Zeitpunkt mit Strom versorgt werden, sobald das Steuerungsprogramm die korrekten Stellgrößen bereitstellt. Ist dies nicht der Fall kann eine Schädigung der Servomotoren durch undefinierte Signale auftreten.

Das erarbeitete Konzept, unter Einhaltung der angeführten Einschränkungen, ist in Abb. 6.1 dargestellt.

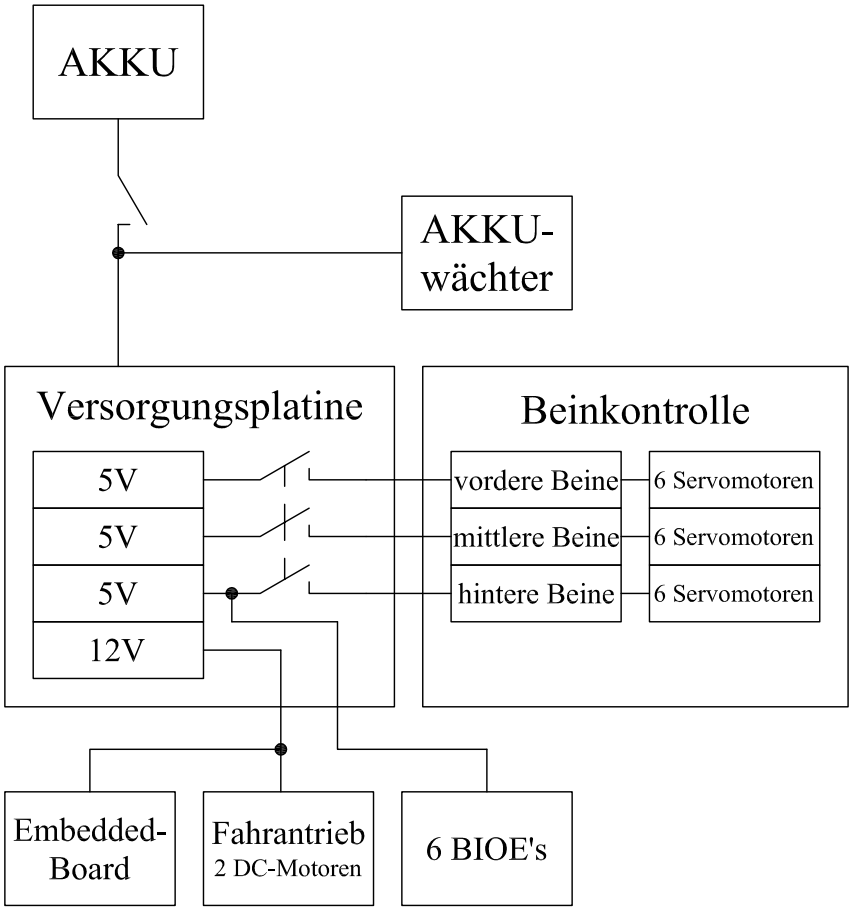


Bild 6.1.: Übersicht Stromversorgung Krabbler

Für die autonome Stromversorgung werden Akkus (s. Kap. 6.1 auf S. 24) gewählt, die eine Mindestspannung von $20V$ zu Verfügung stellen. Damit können die geforderten konstanten Spannungsquellen realisiert werden. Um eine zu starke Entladung der Akkus zu verhindern, wird die Platine Akkuwächter (s. Kap. 6.2 auf S. 25) hinzugefügt. Um die verschiedenen konstanten Spannungsquelle realisieren zu können, wird die Versorgungsplatine (s. Kap. 6.3 auf S. 26) entwickelt. Auf dieser Platine wird auch die getrennte Versorgung von der Beinkontrolle realisiert.

6.1. Akkumulatoren

Für die Stromversorgung werden Akkus des Typs LI-FE der Firma *A123 Systems* verwendet. Durch diese Art der Stromversorgung wird ein autonomer Betrieb von etwa $2h$ erreicht. Dabei werden 2 Pakete aus jeweils 4 in Serie geschalteten Zellen verwendet. Diese beiden Pakete besitzen im geladenen Zustand jeweils eine Spannung von $13,2V$. Um eine stabile Stromversorgung zu gewährleisten werden die beiden Pakete in Serie geschalten, was somit eine Versorgungsspannung von $26,4V$ im voll geladenen Zustand ergibt. Die genauen technischen Eigenschaften der Akkumulatoren sind in den Datenblätter[1] zu finden.

[1] http://www.lindinger.at/shopexpand.php?artikelnr=67765

Beim Entladen der Akkus ist darauf zu achten, dass die Spannung einer einzelnen Zelle nicht unter $2,5V$ fällt, da dies eine Schädigung der Akkus zur Folge hätte. Daraus ergibt sich eine Mindestspannung bei den 8 in Serie geschalteten Zellen von $20V$. Auf diese Spannung wird die untere Spannungsgrenze des Akkuwächters (s. Kap. 6.2 auf S. 25) eingestellt.

Zum Laden der beiden Akkupakete wird der Schnelllader *EOS 606 I 12V* mit Balancer verwendet. Dieses Ladegerät benötigt eine Versorgungsspannung von $12V$ Gleichstrom. Um die verschiedenen Ladegeschwindigkeiten der einzelnen Zellen ausgleichen zu können, und somit ein vollständiges Laden jeder Zelle eines Paketes zu erreichen, wird der Balancer benötigt. Dieser sorgt für ein gleichbleibendes Spannungsniveau zwischen den einzelnen Zellen beim Ladevorgang.

6.2. Akkuwächter

Um ein zu starkes Entladen der Akkus zu verhindern, wird das Grundkonzept der Schaltung aus [Roh08] übernommen und für den Sechsbeiner modifiziert. Sie dient zur Überwachung der momentanen Spannung, die von den Akkus abgegeben wird. Außerdem ist es erforderlich, das Embedded Board vor Abstürzen zu sichern, welches ebenfalls durch eine zu geringe Spannungsversorgung hervorgerufen werden kann. Die auf dieser Platine eingebauten Leuchtdioden zeigen das Spannungsniveau der Akkus. Sobald nur mehr die roten LEDs leuchten, müssen die beiden Akkus unbedingt aufgeladen werden, um eine Schädigung dieser zu vermeiden.

Das obere und untere Spannungsniveau des Akkuwächters wird mit den Drehpotentiometern $R2$ und $R3$ eingestellt. Dabei wird zuerst das obere Spannungsniveau mit $R2$ eingestellt und dann das untere mit $R3$. Für die von uns verwendeten Akkus wird als obere Grenze eine Spannung von $26,4V$ und als unter Grenze eine Spannung von $20V$ gewählt (s. Kap. 6.1 auf S. 24).

Mit dem Jumper JP1 (s. Abb. 6.2) kann die Anzeige entweder im Punktmodus (nur eine LED leuchtet) oder Balkenmodus (alle LEDs bis zum aktuellen Spannungsniveau leuchten) gewählt werden. Ist der Jumper nicht gesetzt, befindet sich der Akkuwächter im Punktmodus.

Der Schaltplan des Akkuwächter ist in Abb. 6.2 dargestellt.
Layout der Platine, Bauteilliste und Bestückungsplan zu dem in Abb. 6.2 dargestellten Schaltplan sind in Anhang A zu finden.

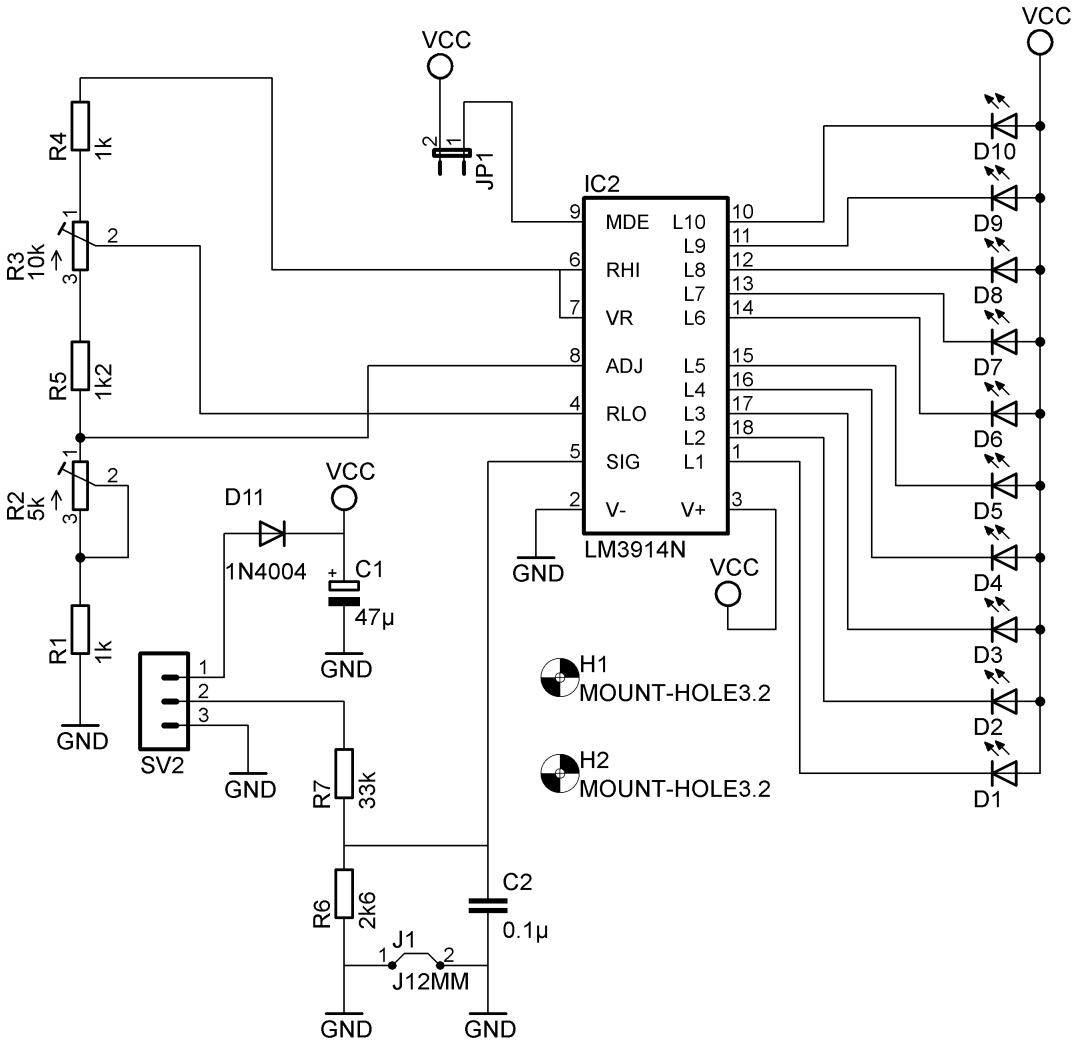


Bild 6.2.: Schaltplan Akkuwächter

6.3. Versorgungsplatine

Die Versorgungsplatine muss die schwankende Spannung der Akkus, die zwischen 20*V* und 26*V* liegt, mittels Spannungswandler auf konstante Spannungen von 5*V* und 12*V* bringen. Durch den großen Stromverbrauch der Servomotoren, mi einem Spitzenstrom von ca. 9*A*, muss deren Versorgung auf drei parallel verlaufende Zweige aufgeteilt werden. Ansonsten ist eine Realisierung mit diesen Spannungswandlern nicht möglich. Als vierten Zweig wird die 12*V* Versorgung für das Embedded Board und den Fahrantrieb implementiert. Mittels Relais wird die benötigte Trennung der Servomotoren von der Stromversorgung während der Inbetriebnahme des Sechsbeiners vorgenommen. Diese Relais werden bei dieser Lösung durch einen externen Schalter betätigt, jedoch ist ein Schalten der Relais mittels Transistoren auf dieser Platine bereits vorgesehen. Diese Transistoren können dann vom Embedded Board aus gesteuert werden und somit der Schaltvorgang vom Steuerungsprogramm durchgeführt werden.

Die Spannungswandler werden mit der Schaltung aus Abb. 6.3 realisiert.

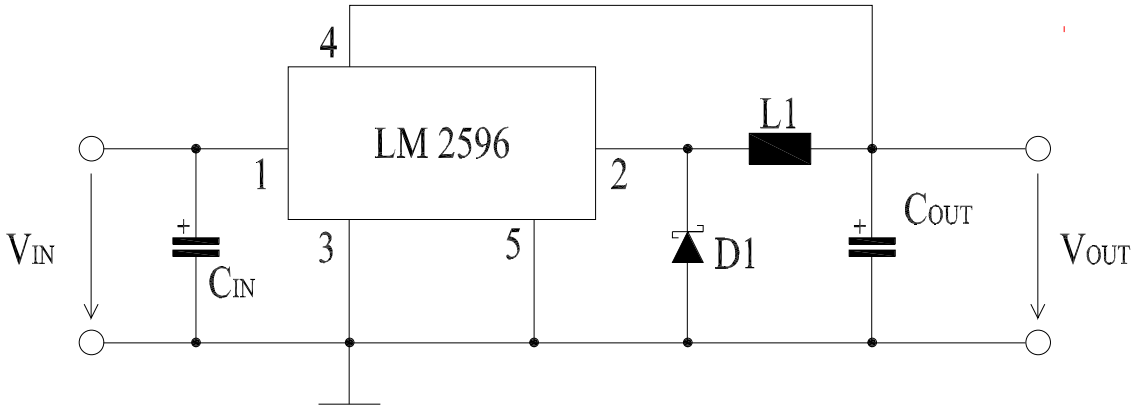


Bild 6.3.: Schaltplan Spannungswandler

Als Spannungswandler wurde der Baustein *LM2596* verwendet. Im Datenblatt[2] sind typische Verschaltungsmöglichkeiten dieses Bausteins vorgegeben. Für die Berechnung und Bestimmung der dazugehörigen Bauteile sind ebenfalls im Datenblatt Formeln und Tabellen zu finden. Somit ist es relativ einfach die benötigten Schaltungen, für die beiden Spannungspegel $5V$ und $12V$, zu bestimmen.

Layout der Platine, Bauteilliste und Bestückungsplan der gesamten Versorgungsplatine sind in Anhang A zu finden.

[2] http://www.national.com/ds/LM/LM2596.pdf

7 Servomotoren

7.1. Allgemeines zu den Servomotoren

Der Antrieb für die Beineinheiten besteht aus digitalen Modellbauservomotoren. Diese Servomotoren wurden bereits in der Diplomarbeit [Kas06] ausgewählt und modifiziert. Sie können ohne Probleme in diese Projektarbeit übernommen werden.
Es werden zwei verschiedene Typen von Servomotoren verwendet. Diese werden von der Firma *Conrad* gekauft, jedoch handelt es sich dabei um umbenannte Modellbauservos der Firma *Hitec*. Die Bezeichnungen der beiden Servotypen sind *HS-5645MG* und *HS-5245MG*. Genauere technische Angaben zu den Servomotoren sind in [Kas06] zu finden.
Diese Modellbauservomotoren können mit einer Spannung von $4.8V$ bis $6V$ laut Datenblatt betrieben werden. Um eine einheitliche Versorgung mit den BIOEs herstellen zu können, wird für sie eine Versorgungsspannung von $5V$ gewählt. Bei der Diplomarbeit [Kas06] wird auf Grund der vorhandenen Akkumulatoren eine Versorgungsspannung von $6V$ verwendet.

7.2. Kalibrieren der Servomotoren

Im Steuerungsprogramm für den Sechsbeiner werden als Ausgangswerte für die Servomotoren die Motorwinkel in Radiant berechnet. Als Rückgabewerte von den Servomotoren erwartet das Programm ebenfalls den Motorwinkel in Radiant. Die BIOE-Einheit benötigt für die Steuerung der Servomotoren einen Zahlenwerte zwischen 0 und 39936 (Beschreibung s. Kap. 4 auf S. 16) und als Rückgabewerte liefern die A/D-Wandler Werte zwischen 0 und 1023. Nun muss hier, wenn möglich, ein linearer Zusammenhang zwischen diesen Zahlenwerten und den Motorwinkeln gefunden werden. Da die gleichen Servotypen untereinander Abweichungen in der Nullposition und in den Extrempositionen aufweisen, muss dieser Zusammenhang für jeden der 18 Motoren separat bestimmt werden. Somit ergibt sich für jeden eine Geradengleichung für die Umrechnung des Motorwinkel in den PWM-Zahlenwert und eine für die Umrechnung des ADC-Wertes in den Motorwinkel.

Um möglichst gute Ergebnisse zu erzielen und diesen Kalibrierungsvorgang gegebenenfalls schnell wiederholen zu können, werden dafür einige Programme entwickelt. Diese haben sich bereits als sehr nützlich erwiesen, da beim Wechseln von defekten Servomotoren ebenfalls wieder eine Kalibration vorgenommen werden muss.

Als erstes wird beim Abstimmen der Motoren der Nullpunkt und die beiden Endpunkte festgelegt und dabei gleichzeitig der dazugehörige Motorwinkel durch Messen ermittelt. Dazu wird das entwickelte Matlab-Modell *messtest*, welches in Abb. 7.1 zu sehen ist, verwendet.

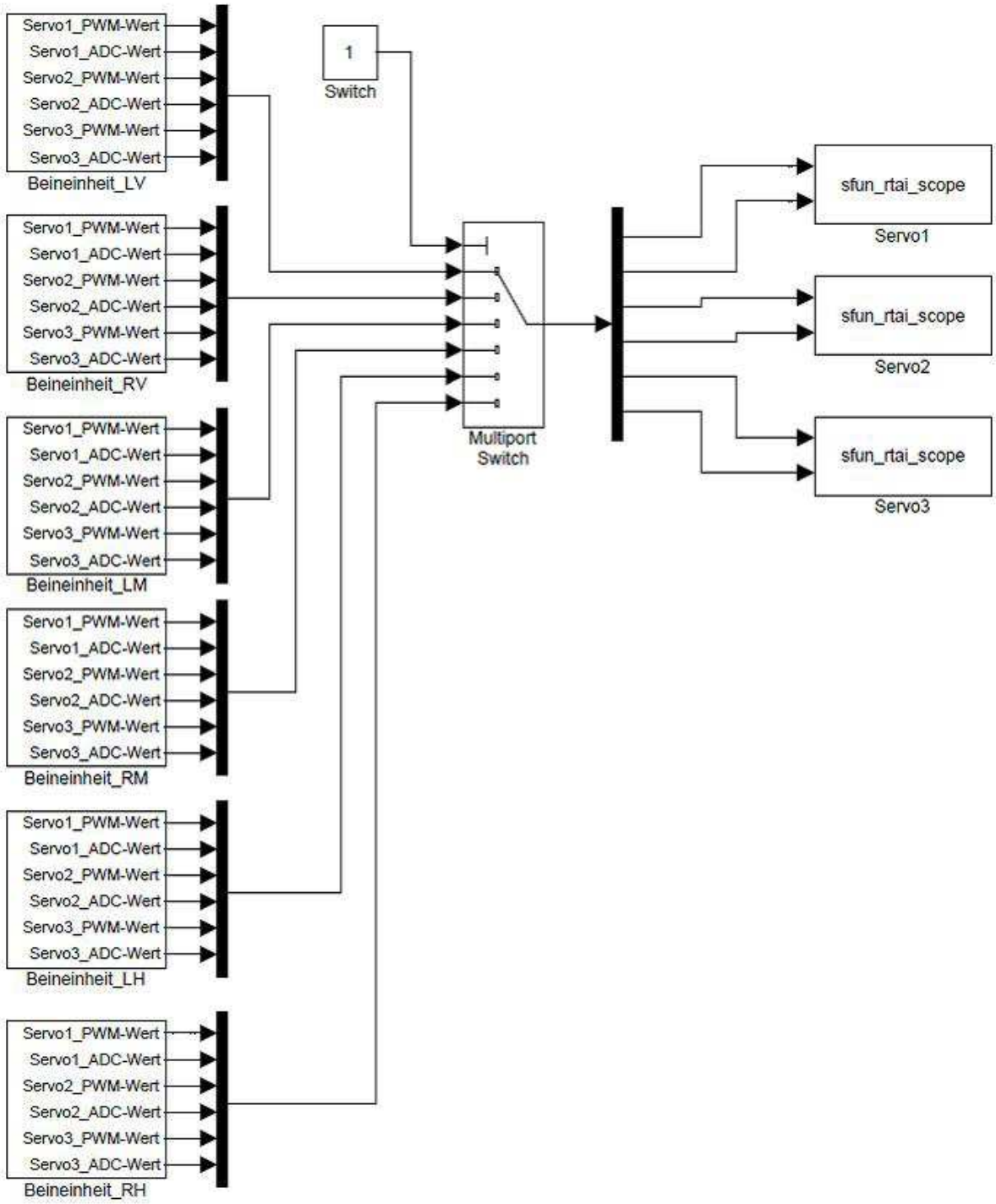


Bild 7.1.: Matlab-Modell für die Aufnahme von Servodaten

Dieses Modell besteht aus 6 Subsystemen, wobei jedes Subsystem einer Beineinheit zugeordnet ist. Durch die Konstante *Switch* kann zwischen den einzelnen Beineinheiten durchgeschalten werden, und die PWM- und ADC-Werte, der drei Servomotoren pro Beineinheit, auf Scopes dargestellt werden. Die Subsysteme, von welchen eines in Abb. 7.2 dargestellt ist, erzeugen PWM-Siganle, welche an die Servomotoren gesendet werden. Diese können im laufenden Programm in Linux beliebig verändert werden, was für die Bestimmung des Nullpunktes und der beiden Endpunkte notwendig ist. Gleichzeitig werden die zu den PWM-Werten dazugehörigen

ADC-Werte an das Programm zurückgegeben. Dies ermöglicht es dann einen Zusammenhang zwischen diesen beiden Werten zu berechnen. Kern dieser Subsysteme ist der Block BIOE-Device welcher in Kapitel 13.1 auf Seite 60 genauer beschrieben wird.

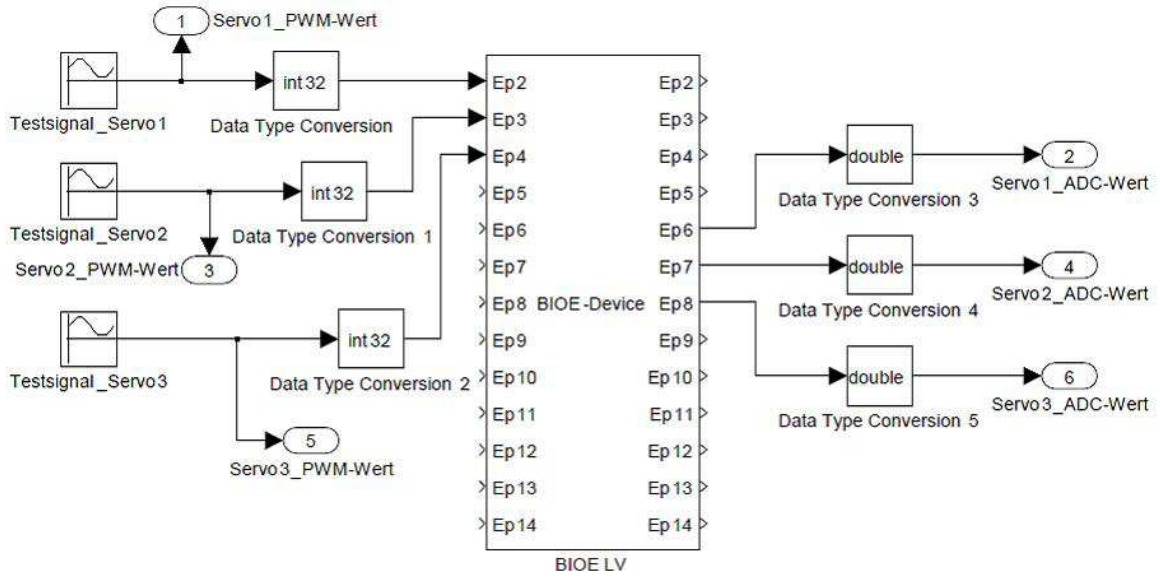


Bild 7.2.: Subsytem pro Beineinheit

Aus diesem Modell wird nun ein Echtzeitprogramm generiert, welches auf den Sechsbeiner übertragen und gestartet wird (s. Kap. 13.4 auf S. 13.4). Um die PWM- und ADC-Daten darstellen und speichern zu können, muss eine Verbindung zu einem PC mit dem Programm QRtaiLab (s. Kap. 10 auf S. 48) hergestellt werden.
Als erstes werden die Nullpunkte und die Endpunkte der einzelnen Servomotoren und die dazugehörigen Winkel in den Endpunkten ermittelt. Dafür wird die Amplitude des jeweiligen Sinusgenerators auf Null gesetzt und der Offset solange verändert, bis sich der gewünschte Winkel am Servomotor einstellt. Mit den drei Scopes in diesem Programm können die ADC-Werte zu den dazugehörigen Motorwinkeln und PWM-Werten bestimmt werden. Diese Werte müssen für alle 18 Servomotoren ermittelt werden.
In Abb. 7.3 ist ein Screenshot von dem Programm QRtaiLab, welches mit dem Programm *mestest* am Embedded Board verbunden ist, abgebildet.

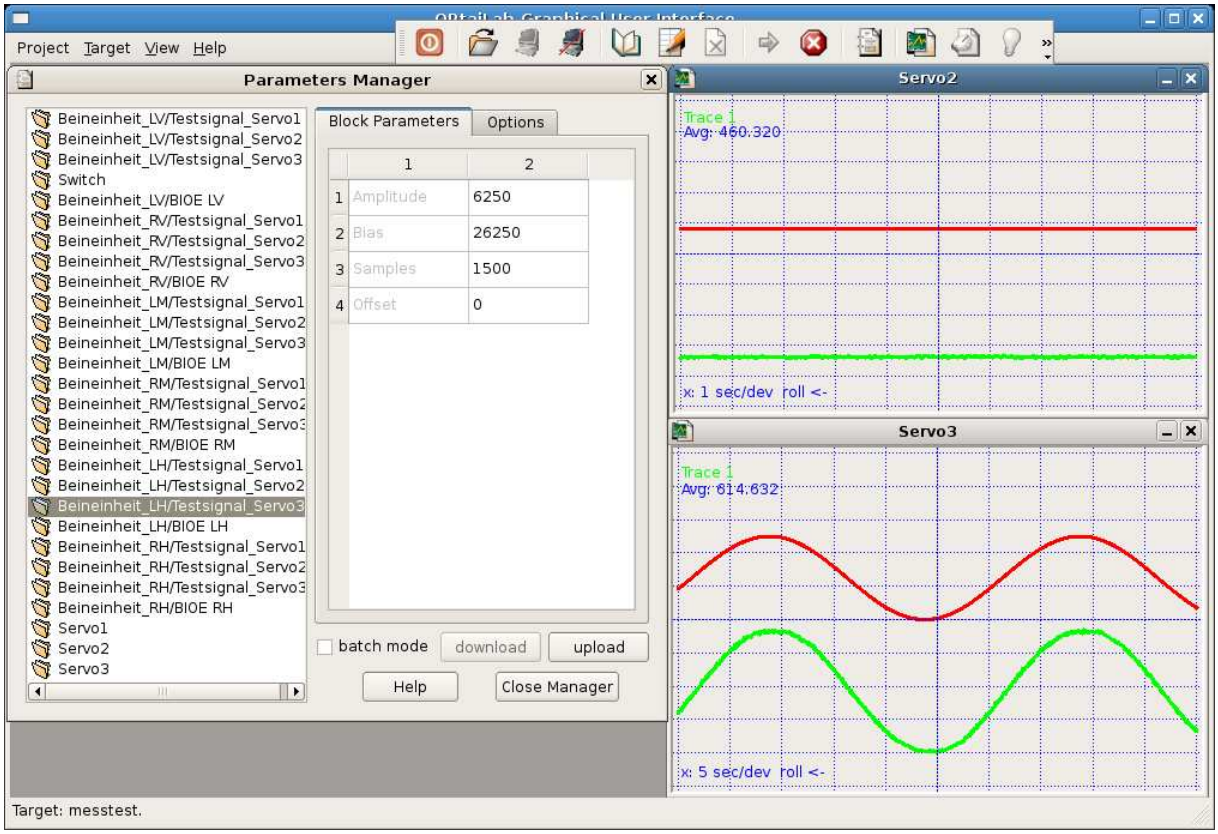


Bild 7.3.: Screenshot von der Aufnahme der Servodaten

Als nächstes wird nun pro Beineinheit auf die drei Servomotoren ein sinusförmiges Eingangssignal gelegt, welches die beiden ermittelten Endpunkte des PWM-Signals der Servomotoren als Amplitudenmaximum bzw. Minimum besitzt. Die Periodendauer des Sinussignals muss im Bereich zwischen 20 und 30 Sekunden gewählt werden, damit die Verarbeitungszeit des A/D-Wandler und Abtastzeit keinen bzw. nur geringen Einfluss auf den ADC-Wert haben. Um den Einfluss von zufällig auftretenden Störsignalen auf die zu ermittelnde Geradengleichung so gering wie möglich zu halten, sollen ca. 10 Perioden des Sinussignals aufgenommen werden.

Als nächsten Schritt werden nun in die Datei *initPWM.m* die Grenzen der PWM-Werte für den maximalen und minimalen Winkelausschlag der Servomotoren, sowie die Offsetwerte eingetragen. Diese dienen später zum Berechnen der Geradengleichung und auch als Begrenzung für die maximal bzw. minimalen PWM-Werte die vom Steuerungsprogramm ausgegeben werden dürfen. Diese Begrenzungen sind sehr wichtig, da der Winkelausschlag der Servomotoren durch die konstruktive Beschaffenheit der Beine begrenzt ist. Bei einer Über- bzw. Unterschreitung dieser Werte kann es zur Zerstörung der Servomotoren kommen. In dieser Datei darf deshalb auf keinen Fall eine willkürliche Veränderung durchgeführt werden.

In Listing 7.1 sind Auszüge des Sourcecodes zu sehen.

Listing 7.1: initPWM.m

```
1 %Initilisierung der Grenzwerte und Offsets für die Berechnung der
2 %Geradengleichungen und Begrenzungen der PWM-Ausgangserte
3
```

```
%Beineinheit links vorne
LV_1_phimax = 11700;
LV_1_phi_p60 = 15800;
LV_1_phimin = 35700;
LV_1_phi_n60 = 31600;
LV_2_phimax = 12000;
LV_2_phimin = 34800;
LV_3_phimax = 19000;
LV_3_phimin = 32000;

LV_1_pwm_off = 23700;
LV_2_pwm_off = 23400;
LV_3_pwm_off = 24500;
```

In der Methode *calcKoeffs.m* muss nun der minimale und maximale Winkelausschlag, der für die Aufnahme der Motordaten verwendet wurde, eingetragen werden. Als nächster Schritt wird die Methode ausgeführt und die Koeffizienten für die Geradengleichungen werden berechnet und in die Datei *koeffs* gespeichert. Diese Datei wird beim Start des Steuerungsprogramms geladen.

In Listing 7.2 ist ein Teil des Quellcodes der Methode *calcKoeffs.m* für eine Beineinheit zu sehen. In diesem Quellcode wird für die Berechnung der Koeffizienten für jeden Servomotor die Funktion *fitit.m* aufgerufen. Der Quellcode dieser Funktion ist in Listing 7.3 zu sehen. In dieser Funktion wird mit dem Befehl `polyfit` eine Ausgleichsgerade von den aufgenommen Motordaten ermittelt und daraus die benötigten Koeffizienten berechnet.

Listing 7.2: calcKoeffs.m

```
%Methode zum berechnen der Koeffizienten für die Geradengleichungen
clear all
%Laden des Initialisierungsfile
initPWM
%Aufruf der Funktion fitit zur Berechnung der Koeffizienten

%Beineinheit links vorne
[kpwm_LV_1, kadc_LV_1, LV_1_adc_off] = fitit('Servo1_LV',LV_1_phi_n60
    ,LV_1_phi_p60, -60*pi/180,60*pi/180, LV_1_pwm_off)
[kpwm_LV_2, kadc_LV_2, LV_2_adc_off] = fitit('Servo2_LV',LV_2_phimin,
    LV_2_phimax, -75*pi/180,75*pi/180, LV_2_pwm_off)
[kpwm_LV_3, kadc_LV_3, LV_3_adc_off] = fitit('Servo3_LV',LV_3_phimin,
    LV_3_phimax, -50*pi/180,35*pi/180, LV_3_pwm_off)
```

Listing 7.3: fitit.m

```
function [kpwm, kadc, adc0] = fitit(infile, pwmin, pwmax, phimin,
    phimax, pw0)

%function [kpwm, kadc, adc0] = fitit(infile, pwmin, pwmax, phimin,
    phimax,
%pw0)

```

```
6  %In dieser Funktion wird aus aufgenommenen
7  %Eingangswerte
8      %infile ... Datenfile des Servomotor
9      %pwmin .... PWM-Wert für den minimalen Motorwinkel
10     %pwmax .... PWM-Wert für den maximalen Motorwinkel
11     %phimin ... minimaler Motorwinkel
12     %phimax ... maximaler Motorwinkel
13     %pw0 ..... PWM-Offset-Wert des Servomotor
14 %Ausgangswerte
15     %kpwm ..... Steigung der Geradengleichung zwischen Winkel und PWM-
           Signal
16     %kadc ..... Steigung der Geradengleichung zwischen Winkel und ADC-
           Signal
17     %adc ..... Offset der Geradengleichung zwischen Winkel und ADC-
           Signal
18
19 data=load(infile);                          %laden des Datenfile
20 %ermitteln der Ausgleichskurve erster Ordnung
21 p = polyfit(data(:,2),data(:,3),1);
22 %Berechnung der Endpunkte und Koeffizienten für die
       Geradengleichungen
23 adcmin = p(1)*pwmin+p(2);
24 adcmax = p(1)*pwmax+p(2);
25 kpwm = (pwmax-pwmin)/(phimax-phimin);
26 kadc = (phimax-phimin)/(adcmax-adcmin);
27 adc0 = p(1)*pw0+p(2);
```

8 Fahrantrieb

8.1. Gleichstrommotoren

Der Fahrantrieb besteht aus 2 Gleichstrommotoren. Dafür wurde der Typ *1717 012SR* wie in Abb. 8.1 zu sehen ist, von der Firma *Faulhaber*, ausgewählt.

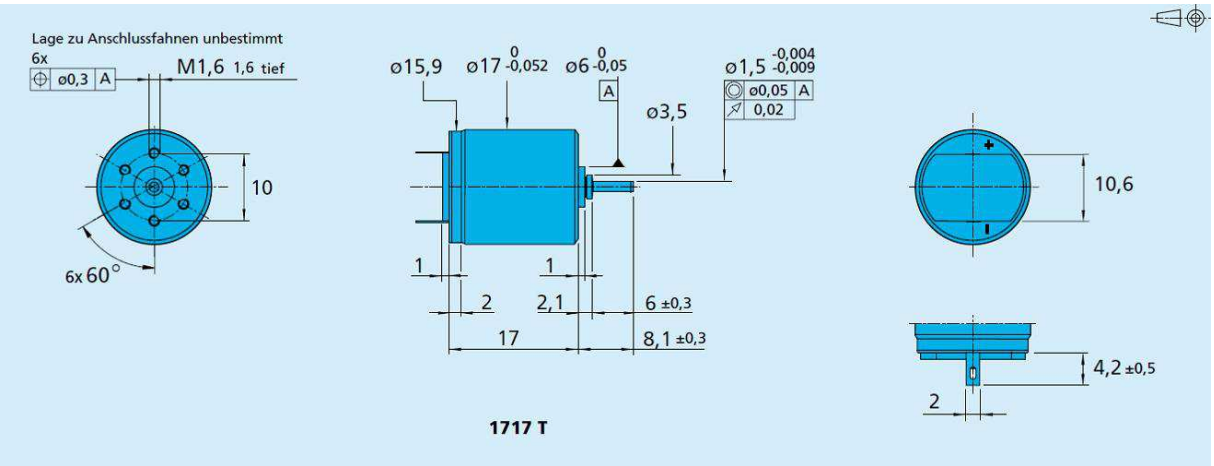


Bild 8.1.: Bauart der Gleichstrommotoren

Dieser Gleichstrommotor hat eine Nennspannung von $12V$ und gibt eine Leistung von $1.97W$ bei einem Wirkungsgrad von 70% ab. Der Motor besitzt ein Dauerdrehmoment von $2mNm$ und eine Leerlaufdrehzahl von $14000U/min$. Mit den beiden Gleichstrommotoren, welche an den vorderen Beinen des Sechsbeiners montiert sind, ist das Moment, nach dem Zwischenschalten eines Getriebes, für geringe Steigungen und nicht zu enge Radien in den Kurven ausreichend. Eine sinnvolle Erweiterung wird sein, noch zwei Gleichstrommotoren auf den beiden Hinterbeinen zu montieren. So können dann größere Steigungen und kleinere Radien in den Kurven bewältigt werden. Außerdem ist der dafür notwendige Aufwand sehr gering, da auf allen Platinen der Beineinheit die Ansteuerung für die Gleichstrommotoren vorgesehen ist. Für die Erweiterung muss dann nur das Steuerungsprogramm geringfügig geändert werden.

8.2. Planetengetriebe

Für den Fahrantrieb ist die Drehzahl direkt an der Motorwelle zu groß und das Drehmoment würde nicht ausreichen um den Sechsbeiner zu bewegen. Deshalb wird ein Getriebe für die Gleichstrommotoren benötigt. Hier bietet die Firma *Faulhuber* eine Vielzahl an Planetengetriebe an. Es wurde das Planetengetriebe der Serie *16/7*, welches in Abb. 8.2 zu sehen ist, mit dem Übersetzungsverhältnis von 246 : 1, der Firma *Faulhaber*, gewählt. Mit diesem Übersetzungsverhältnis ergibt sich ein Dauerdrehmoment von $49.2mNm$.

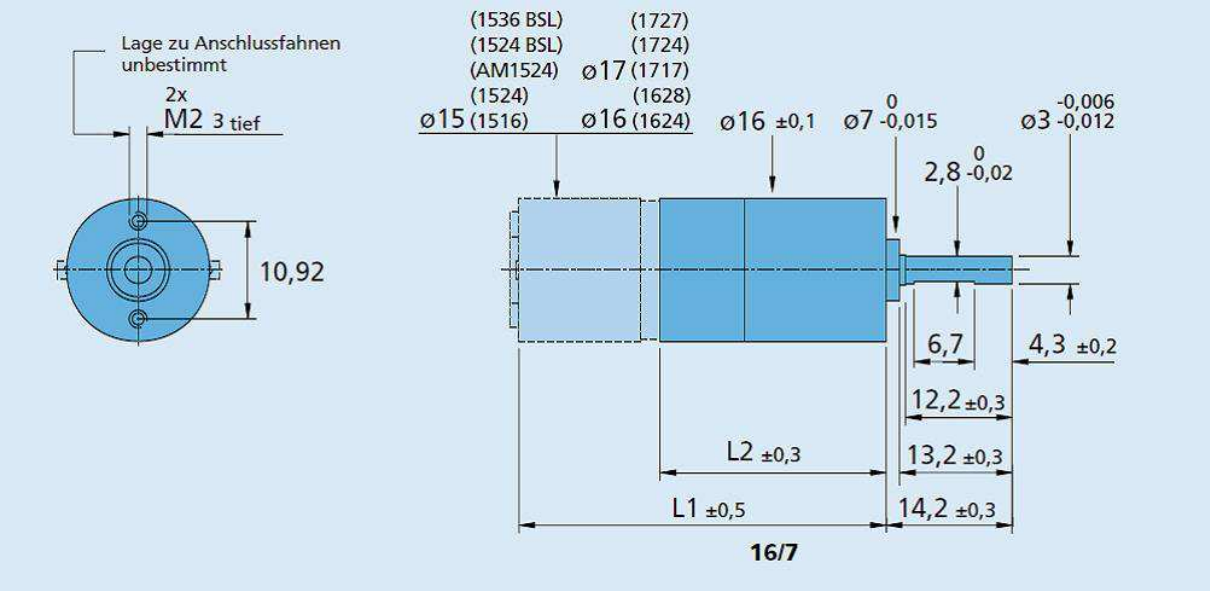


Bild 8.2.: Bauart des Planetengetriebe

Weitere technische Details zu dem Gleichstrommotor und dem Planetengetriebe können aus den Datenblättern[1] entnommen werden.

8.3. Impulsgeber

Für die Regelung der Gleichstrommotoren wird ein Positionssignal, dass im Steuerungsprogramm auf eine Geschwindigkeit umgerechnet wird, benötigt. Für die Feststellung der Position wird ein Impulsgeber verwendet. Auch in diesem Fall bietet die Fa. *Faulhaber*, für die verwendeten Gleichstrommotoren, Lösungen an. Für die beiden Motoren wurden Magnetische Impulsgeber der Serie *IE2-16* ausgewählt. In Abb. 8.3 sind die Ausgangssignale, Schaltdiagramm und Steckerinformationen des Magnetischen Impulsgeber zu entnehmen.

[1] http://www.faulhaber.com/sprache2/n164058/n.html

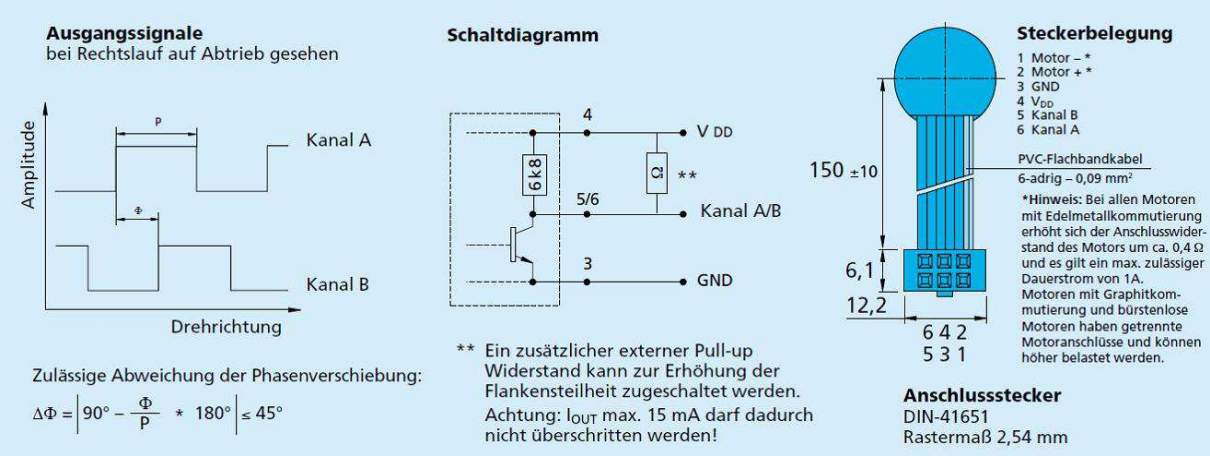


Bild 8.3.: Ausgangssignale, Schaltdiagramm und Steckerbelegung des Impulsgeber

Der Impulsgeber kann mit einer Spannung zwischen $4V$ und $18V$ Gleichstrom versorgt werden. Der Inkrementaldecoder des BIOE benötigt rechteckförmige Eingangssignale zwischen $0V$ und $5V$. Daraus ergibt sich eine Versorgungsspannung für den Impulsgeber von $5V$. Mit den beiden Kanälen A und B wird die Position und Drehrichtung des Gleichstrommotor ermittelt. Bei einer vollen Umdrehung werden 16 Impulse pro Kanal ausgegeben. In Verbindung mit dem gewählten Getriebe, welches ein Übersetzungsverhältnis von 246 : 1 besitzt, ergeben sich 3936 Impulse pro voller Umdrehung der Abtriebswelle. Diese hohe Anzahl an Impulsen ermöglicht eine gute Auflösung der Positionsbestimmung. In Abb. 8.3 sind die beiden Signale für einen Rechtslauf, auf den Abtrieb gesehen, dargestellt.

Teil II.

Software

9 *RTAI*

9.1. Was ist *RTAI*

RTAI(Real Time Application Interface) ist eine freie Erweiterung von Linux zu einem Echtzeitbetriebssystem. Durch Patchen eines normalen Linux-Kernels wird zwischen diesem und der Hardware, wie in Abb. 9.1 ersichtlich, ein Echtzeitkernel eingefügt, der die Interruptverwaltung des Prozessors übernimmt. Auf diesem Echtzeitkernel laufen dann die Echtzeittasks. Der Standard Linux-Kernel ist selbst auch ein Echtzeittask, der allerdings die niedrigste Priorität besitzt. Ein weiterer Vorteil von *RTAI* besteht darin, dass mithilfe des LXRT-Moduls Echtzeittasks auch im sogenannten „Userspace“ ausgeführt werden können. Dadurch bleiben die Linux-Schutzmechanismen aktiv und Fehler im Programmcode wirken sich in der Regel nicht auf die Stabilität des Gesamtsystems aus. Nach Abschluss der Testphase kann das Programm ohne große Änderungen in ein Kernel-Modul portiert werden, worauf im Zuge dieser Projektarbeit aber verzichtet wird.

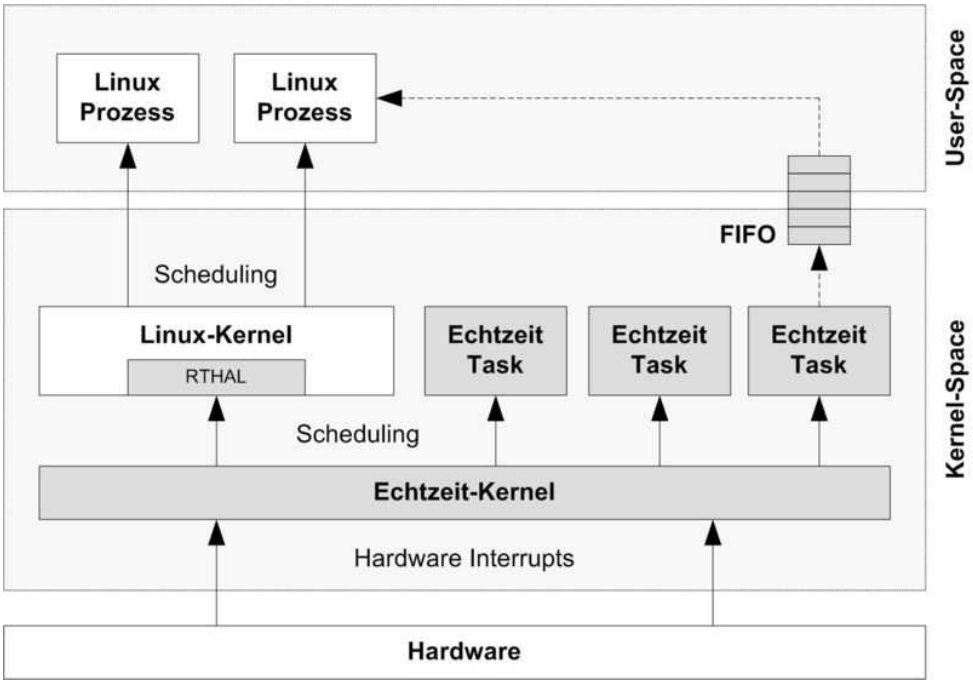


Bild 9.1.: *RTAI*-Architektur

9.2. Installation am Host-Rechner

1. **Installation des Betriebssystems Debian**
   Ausgangspunkt der Installation ist ein Standard Debian-Linux System. Die benötigte

Installations-CD steht frei zum herunterladen bereit[1].

2. **Installation der benötigten Pakete**
   Nach erfolgreicher Installation des Grundsystems werden die benötigten Pakete installiert. Dazu wird in einem Terminal mit dem Befehl `sudo su` und dem Benutzerpasswort zum Administrator-Benutzer gewechselt. Anschließend werden mit `apt-get install` die benötigten Pakete
   - `gcc`
   - `make`
   - `libc6-dev`
   - `libc-dev`
   - `libncurses5-dev`
   - `patch`

   installiert.

3. **Herunterladen von *RTAI* und Kernel**
   Anschließend wird der *RTAI*-Quellcode der verwendeten Version 3.6.2 heruntergeladen[2]. Nach dem Entpacken des Archivs nach */usr/src/* wird eine Kernel-Version ausgewählt. Welche Versionen unterstützt werden entnimmt man dem Verzeichnis */usr/src/rtai-3.6.2/base/arch/i386/patches/*. In der verwendeten *RTAI*-Version sind Patches für folgende Kernels enthalten:
   - `hal-linux-2.6.23-i386-1.12-03.patch`
   - `hal-linux-2.6.22-i386-1.10-12.patch`
   - `hal-linux-2.6.20.21-i386-1.12-03.patch`
   - `hal-linux-2.6.19-i386-1.7-01.patch`
   - `hal-linux-2.4.36-i386-1.3-04.patch`
   - `hal-linux-2.4.35.5-i386-1.3-04.patch`
   - `hal-linux-2.4.34-i386-1.3-04.patch`

   Verwendet wird die Kernelversion 2.6.23(.17) die nach dem herunterladen[3] ebenfalls nach */usr/src/* entpackt wird. Zur Vereinfachung wird mit

   ```
   ln -s linux-2.6.23.17 linux
   ```

   noch eine Verknüpfung nach */usr/src/linux/* angelegt.

4. **Kernel patchen und kompilieren**
   Um die *RTAI*-Erweiterung im Kernel zu integrieren, muss dieser nun gepatcht werden:

   ```
   cd /usr/src/linux
   patch -p1 </usr/src/rtai3.6.2/base/arch/i386/
           patches/hal-linux-2.6.23-i386-1.12-03.patch
   ```

   Anschließend wird der Kernel mit

[1] http://www.debian.de/CD/netinst/
[2] http://www.rtai.org
[3] http://www.kernel.org

```
make menuconfig
```

konfiguriert. Generell ist darauf zu achten, einen möglichst schlanken Kernel zu erzeugen, d.h. unnötige Module (Sound,...) zu deaktivieren. Folgende Einstellungen müssen auf jeden Fall vorgenommen werden:

- General Setup
  - Kernel .config support . . . *aktivieren*
- Enable Loadable Module Support
  - Module unloading . . . *aktivieren*
  - Module versioning Support . . . *deaktivieren*
- Processor Type and Features
  - Subarchitecture Type . . . *generic architecture*
  - Prozessor Family . . . *entsprechend der verwendeten CPU setzen: Pentium 4*
  - Generic x86 support . . . *aktivieren*
- Power Management Support
  - ACPI . . . *aktivieren*
  - CPU Frequency Scaling . . . *aktivieren*
- Networking
  - Wireless Extensions . . . *aktivieren*
- Device Drivers
  - Parallel Port Support . . . *aktivieren*
  - USB Support . . . *aktivieren*
  - evtl. benötigte Chipsatztreiber aktivieren (siehe `lspci`)
  - schlanken Kernel generieren (unnötige Treiber wie z.B. Sound deaktivieren)
- Cryptographic API
  - Cryptographic algorithm manager . . . *aktivieren*
  - AES cipher algotithm . . . *aktivieren*
- File Systems
  - Ext3 . . . *aktivieren*
  - Reiserfs . . . *aktivieren*
  - NFS filesysten support . . . *aktivieren*
  - NFS server support . . . *aktivieren*

Nach der Konfiguration wird der Kompilierungsvorgang mit

```
make && make install && make modules_install
```

gestartet. Nach erfolgreichem Abschluss muss nun noch der Bootloader angepasst werden. Dazu wird in der Datei */boot/grub/menu.lst* folgender Eintrag hinzugefügt:

Listing 9.1: Auszug aus menu.lst

```
1 title    Debian-Linux mit RTAI-Kernel
2 root     (hd1,0)
3 kernel   /boot/vmlinuz-2.6.23.17 root=/dev/hdb1 ro quiet splash
     lapic
4 savedefault
```

Jetzt kann der Entwicklungsrechner mit dem neuen Kernel neu hochgefahren werden.

5. ***RTAI* konfigurieren und kompilieren**
   Nach erfolgreichen Neustart wird ins Verzeichnis */usr/src/rtai-3.6.2* gewechselt und mit

   ```
   make menuconfig && make && make install
   ```

   der Installationsvorgang für *RTAI* angestoßen. Das erscheinende Menü kann sofort wieder beendet werden da die Standardeinstellungen ausreichend sind.
   Zuletzt werden noch zwei Umgebungsvariablen gesetzt, damit Kommandos und Bibliotheken ohne vollständige Pfadangabe gefunden werden:

   ```
   echo "export PATH=$PATH:/usr/realtime/bin"
          >>/etc/profile
   echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:
          /usr/realtime/lib" >>/etc/profile
   ```

6. **Erste Tests**
   Damit ist die Grundinstallation von *RTAI* abgeschlossen. Im Verzeichnis */usr/realtime/testsuite* befinden sich einige Tests die ausgeführt werden, bevor die Installation fortgesetzt wird. Listing 9.2 zeigt die Ausgabe des Userspace-Latenztestes der durch ausführen von

   ```
   cd /usr/realtime/testsuite/user/latency
   .run
   ```

   gestartet wird. Dieser Test wird unter Systemlast über längere Zeit ausgeführt und darf dabei keine „overruns“ und Ausreißer bei den Latenzzeiten aufweisen.

Listing 9.2: Ausgabe des Latenztestes

```
*
*
* Type ^C to stop this application.
*
*

## RTAI latency calibration tool ##
# period = 100000 (ns)
# average time = 1 (s)
# use the FPU
# start the timer
# timer_mode is oneshot

RTAI Testsuite - USER latency (all data in nanoseconds)
2009/09/26 23:37:35
RTH|  lat min|  ovl min|  lat avg|  lat max|  ovl max| overruns
```

```
RTD|      619|      619|     2164|     6927|     6927|        0
RTD|      693|      619|     2151|     4047|     6927|        0
RTD|      801|      619|     2136|     7372|     7372|        0
RTD|      653|      619|     2154|     5798|     7372|        0
RTD|      693|      619|     2122|     8175|     8175|        0
RTD|      575|      575|     2111|     3953|     8175|        0
RTD|      728|      575|     2163|     8532|     8532|        0
RTD|      899|      575|     2131|     3700|     8532|        0
RTD|      676|      575|     2143|     7584|     8532|        0
RTD|      845|      575|     2146|     3972|     8532|        0
RTD|      707|      575|     2136|     5982|     8532|        0
RTD|      655|      575|     2138|     8054|     8532|        0
RTD|      782|      575|     2144|     8971|     8971|        0
RTD|      645|      575|     2134|     4069|     8971|        0
RTD|      679|      575|     2149|     9033|     9033|        0
RTD|      799|      575|     2152|     3843|     9033|        0
RTD|      711|      575|     2149|     4161|     9033|        0
RTD|      643|      575|     2151|     5509|     9033|        0
RTD|      643|      575|     2141|     7485|     9033|        0
RTD|      906|      575|     2126|     4449|     9033|        0
RTD|      691|      575|     2141|     6100|     9033|        0
2009/09/26 23:37:55
RTH| lat min| ovl min| lat avg| lat max| ovl max| overruns
RTD|      892|      575|     2146|     3911|     9033|        0
RTD|      720|      575|     2141|     7920|     9033|        0
RTD|      638|      575|     2138|     9259|     9259|        0
RTD|      727|      575|     2148|     7872|     9259|        0
RTD|      672|      575|     2139|     5480|     9259|        0
RTD|      652|      575|     2166|     8556|     9259|        0
RTD|      688|      575|     2170|     8076|     9259|        0
RTD|      688|      575|     2170|     8076|     9259|        0

>>> S = 98.696, EXECTIME = 0.04981
```

7. **Einbinden der *RTAI*-Blöcke in MatLab/SimuLink**
   Im nächsten Schritt werden die *RTAI*-spezifischen Blöcke, wie zum Beispiel „Scopes“ zum Auslesen von Daten, in MatLab/SimuLink eingebunden. Zuerst wird der gesamte Inhalt des Verzeichnisses */usr/src/rtai-3.6.2/rtai-lab/matlab* in das Verzeichnis *MatlabRoot/rtw/c/rtai* kopiert, wobei *MatlabRoot* für das Verzeichnis der Matlab-Installation steht (unter Windows typischerweise *C:/Programme/MATLAB/R2007a*). Anschließend wird Matlab gestartet und die Datei *setup.m* im soeben kopierten Verzeichnis ausgeführt. Jetzt sind die *RTAI*-Blöcke im „Simulink Library Browser“ verfügbar. Abschließend werden in der Datei *rtai.tmf* noch folgende Zeilen angepasst:

   ```
   MATLAB_ROOT      = |>MATLAB_ROOT<|
   ```

   Hier muss `|>MATLAB_ROOT|` durch das Linux Verzeichnis ersetzt werden, in das später einige Matlab-bezogene Dateien kopiert werden (typischerweise */usr/local/matlab*)

   ```
   COMPUTER         = |>COMPUTER<|
   ```

   Hier wird `|>COMPUTER<|` mit `GLNX86` ersetzt (sofern Linux auf einer x86 Architektur

installiert ist). Außerdem müssen noch die Einträge `LINUX_HOME` und `RTAIDIR` modifiziert werden, damit sie auf die entsprechenden Verzeichnisse zeigen. (typischerweise */usr/src/linux* und */usr/realtime*)
Zuletzt werden noch die Zeilen

```
|>START_EXPAND_RULES<|%.o : |>EXPAND_DIR_NAME<|/%.c
  gcc -c $(CFLAGS) $<
|>END_EXPAND_RULES<|
```

im Abschnitt „Rules“ gelöscht werden, da im Linux-Makefile ansonsten die Windows-Pfade zu den benötigten s-Functions eingetragen würden.

Abschließend werden die Ordner */extern*, */rtw* und */Simulink* auf den Linux Entwicklungsrechner in das, zuvor als `MATLAB_ROOT` angegebene Verzeichnis kopiert.

8. **Laden der *RTAI*-Module**
   Bevor Echtzeitprogramme ausgeführt werden können, müssen noch die benötigten Module geladen werden. Es empfiehlt sich dafür ein entsprechendes Scriptfile anzulegen. Dazu wird im Verzeichnis */bin* die Datei *loadrtai* angelegt:

Listing 9.3: loadrtai

```
sync
insmod /usr/realtime/modules/rtai_hal.ko
insmod /usr/realtime/modules/rtai_lxrt.ko
insmod /usr/realtime/modules/rtai_fifos.ko
insmod /usr/realtime/modules/rtai_sem.ko
insmod /usr/realtime/modules/rtai_mbx.ko
insmod /usr/realtime/modules/rtai_msg.ko
insmod /usr/realtime/modules/rtai_netrpc.ko ThisNode="127.0.0.1"
sync
```

   Anschließend wird dieses Script mit

   ```
   chmod +x loadrtai
   ```

   ausführbar gemacht.

Damit ist die Installation von *RTAI* am Entwicklungsrechner abgeschlossen und es können Echtzeittasks ausgeführt werden. Näheres dazu wird in Kap. 13.4 und Kap. 13.5 erläutert. Im nächsten Schritt wird die Portierung des *RTAI*-Systems auf das Embedded-Board (Target) vorgenommen.

9.3. Portierung auf den Ziel-Rechner

1. **Formatieren der Speicherkarte**
   Zur Portierung auf das Zielsystem wird eine *CompactFlash*-Speicherkarte und ein entsprechender Kartenleser benötigt. Auf der Speicherkarte wird mithilfe eines Partitionseditors (z.B. „GParted“) eine bootfähige Ext3-Partition erstellt.

2. **Erstellen des Minimal-Dateisystems**
   Die Speicherkarte wird mit

`mount /dev/sdb /media/disk`

eingehängt. Anschließend wird das Archiv *flash.tgz* heruntergeladen[4] und die Dateien auf die Speicherkarte entpackt.

3. **Kopieren der benötigten Dateien vom Host-Rechner**
   Folgende Dateien müssen vom Entwicklungsrechner auf die CF-Karte kopiert werden:
   - Kernel
     - */boot/vmlinuz*
     - */boot/System.map*
     - */boot/config*
   - Module
     Alle benötigten Module aus dem Verzeichnis */lib/modules/*.
   - RTAI
     - Der Ordner */usr/realtime/bin*
     - Der Ordner */usr/realtime/modules*
     - Der Ordner */usr/realtime/share/rtai*

4. **Konfigurieren der Netzwerkverbindung**
   In den Dateien unter */etc/config/* werden die Netzwerkkonfigurationsdaten eingetragen:
   - ipaddr
   - hostname
   - gateway
   - netmask
   - network
   - broadcast

5. **Eintragen der zu ladenden Module**
   In der Datei */etc/modules* werden die Module eingetragen die beim Hochfahren des Rechners geladen werden sollen.

Listing 9.4: modules

```
1 rtai_hal.ko
2 rtai_lxrt.ko
3 rtai_fifos.ko
4 rtai_sem.ko
5 rtai_mbx.ko
6 rtai_msg.ko
7 rtai_netrpc.ko ThisNode="192.169.1.10"
```

6. **Installieren von GRUB**
   Abschließend wird mit dem Befehl

   `grub-install --recheck --root-directory=/media/disc /dev/sdb`

   der GRUB-Bootloader installiert.

[4] http://linux3.dti.supsi.ch/~bucher/embrtai.html

Damit ist das Minimal-Dateisystem auf der *CompactFlash*-Karte installiert und das Embedded-Board kann hochgefahren werden. Die installierten Benutzer sind:

Benutzername	Passwort
root	root
guest	guest

Tabelle 9.1.: MiniFS Logins

9.4. Herstellung der WLAN-Verbindung

9.4.1. Installation der WLAN-Karte

Zur Installation muss zuerst der benötigte Treiber heruntergeladen und installiert werden[5]. Die heruntergeladene Archivdatei *madwifi-0.9.4.tar.gz* wird nach */usr/src/* entpackt und mit

```
make && make install
```

wird der Quellcode kompiliert und installiert. Anschließend werden die entstandenen Module

ath_pci.ko ..... Atheros driver for PCI/Cardbus devices

ath_hal.ko ..... Atheros HAL

wlan.ko ..... 802.11 support layer

wlan_wep.ko ..... WEP cipher support

wlan_scan_sta.ko ..... station scanning support

ath_rate_sample.ko ..... sample rate control

auf den Zielrechner kopiert. Außerdem wird die Datei *wlanconfig* aus */usr/src/madwifi/* des Entwicklungsrechners nach */bin* des Embedded Boards kopiert.
Damit die Zuweisung der IP-Adresse mittels DHCP funktioniert wird noch ein entsprechender Client installiert. Dazu wird dieser am Entwicklungsrechner (wenn nicht standardmäßig in der Distribution enthalten) mit

```
apt-get install dhcp3-common dhcp3-client
```

installiert. Anschließend werden die Dateien

- */sbin/dhclient*
- */sbin/dhclient-script*
- */var/lib/dhcp/dhclient.leases*

in die entsprechenden Ordner des Zielsystems kopiert. Jetzt kann die WLAN-Verbindung durch Ausführen des Skripts 9.5 initialisiert werden. Damit das automatisch beim Hochfahren des Systems geschieht, wird das Skript im Verzeichnis */etc/rcS.d/* angelegt.

[5] http://madwifi-project.org/

Listing 9.5: S50wlan

```
# init WLAN-Device
modprobe ath_rate_sample
modprobe wlan_scan_sta
modprobe ath_pci countrycode=40
modprobe wlan_wep
wlanconfig ath0 destroy
wlanconfig ath0 create wlandev wifi0 wlanmode sta
ifconfig ath0 up
iwconfig ath0 essid "ROBOTIK"
iwconfig ath0 key BC6998166D
dhclient ath0
```

9.4.2. Konfiguration des Routers

Um eine Verbindung mit dem Roboter herzustellen, muss seine IP-Adresse bekannt sein. Um das zu gewährleisten, wird der verwendete WLAN-Router NETGEAR WGR614v9 so eingestellt, dass dem Sechs-Beiner immer die gleiche IP-Adresse (192.168.1.10) zugewiesen wird. (s. Abb. 9.2)

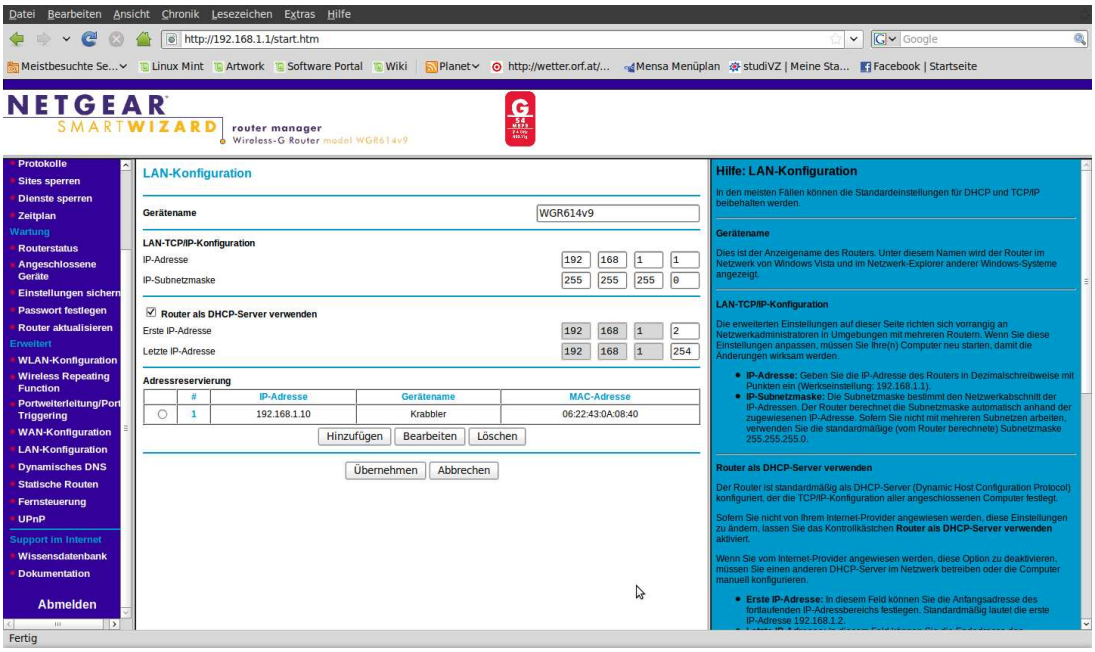


Bild 9.2.: Konfiguration des Routers

Für Benutzername und Passwort des Routers s. Tab. 9.2

Benutzername	Passwort
admin	robadmin

Tabelle 9.2.: Router Login

9.4.3. Konfiguration der Netzwerkfreigabe

Um den Datenaustausch zwischen den Systemen zu vereinfachen, ist es hilfreich das Dateisystem des Entwicklungsrechners am Embedded-Board über ein Netzwerklaufwerk einhängen zu können. Dazu werden zuerst die Pakete

- nfs-kernel-server
- nfs-common
- portmap

mit `apt-get install [paket]` am Entwicklungsrechner installiert. Anschließend werden in der Datei */etc/exports* die gewünschten Freigabeoptionen eingestellt (s. Skript 9.6). Mit

```
portmap start
nfs-kernel-server start
networking start
```

wird die Netzwerkfreigabe gestartet.

Listing 9.6: exports

```
1  # /etc/exports: the access control list for filesystems which may be
       exported
2  #                to NFS clients.  See exports(5).
3  #
4  # Example for NFSv2 and NFSv3:
5  # /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,
       sync,no_sub
6  #
7  # Example for NFSv4:
8  # /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check
       )
9  # /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
10 #
11 /usr/src 192.168.1.10(rw)
```

Jetzt wird in einem Linux-Terminal (bzw. unter Windows in einer Kommandozeilenumgebung) mit

```
telnet 192.168.1.10
```

eine Verbindung zum Sechs-Beiner hergestellt und nach erfolgreicher Anmeldung kann das Verzeichnis */usr/src/* des Entwicklungsrechners mit

```
mount -t nfs 192.168.1.4:/usr/src /mnt/
```

am Embedded-Board unter */mnt/* eingehängt werden. Die verwendete IP-Adresse ist jene des Entwicklungsrechners.

10 QRtaiLab

10.1. Was ist QRtaiLab

QRtaiLab ist ein Programm, dass auf dem Quellcode von *XRtaiLab* aufgebaut ist. Es dient als virtuelles Oszilloskop und kann zur Überwachung und Steuerung von Real-time Anwendungen in Linux verwendet werden. Dabei kann in ein laufendes Programm über Variablenmanipulation eingegriffen werden. Außerdem können Signale auf einem virtuellen Oszilloskop dargestellt bzw. auch aufgenommen und gespeichert werden. Es ist auch möglich über eine LAN-Verbindung auf Programme zuzugreifen, die nicht auf dem selben Rechner ausgeführt werden.

10.2. Installation von QRtaiLab

Um QRtaiLab auf einem Linux-Rechner installieren und ausführen zu können, muss RTAI installiert sein. Als ersten Schritt müssen die Pakete

- libqwt5-qt4
  http://packages.debian.org/de/lenny/libqwt5-qt4
- libqwt5-qt4-dev
  http://packages.debian.org/de/lenny/libqwt5-qt4
- Qwt > 5.0.2
  http://sourceforge.net/projects/qwt/files/
- Qt4 > 4.3.0
  http://qt.nokia.com/downloads

heruntergeladen und installiert werden. Um nun die aktuellste Version von qt4 verwenden zu können, muss ein Link neu gesetzt werden. Dies wird mit dem Befehl

```
ln -s /install_path/Trolltech/QT-4.4.3/include/QT-4.4.3
/usr/include/qt4
```

ausgeführt.
Abschließend fehlt nur mehr das Programm *QRtaiLab*. Dies kann vom Internet[1] heruntergeladen und entpackt werden. Um *QRtaiLab* installieren zu können, muss die Datei qrtailab.config so wie in der Listing 10.1 dargestellt ist, geändert werden. Dabei sind die Pfade für RTAI und qwt anzugeben.

Listing 10.1: qrtailab.config

```
1 ##################################################################
2 # library path
3 ##################################################################
```

[1] http://qrtailab.sourceforge.net/

```
#DEPENDPATH += . /usr/local/qwt-5.1.1/lib
INCLUDEPATH += . /usr/realtime/include /usr/include/qwt-qt4 /usr/src/
    qt-x11-opensource-src -4.4.3
LIBS += -L/usr/local/Trolltech/Qt-4.4.3/lib -lqwt-qt4
#CONFIG += debug
#############
# do same tests
############

#CONFIG += QRLtests
```

Zum Abschluss der Installation muss noch der Befehle

```
qmake-qt4 && make
```

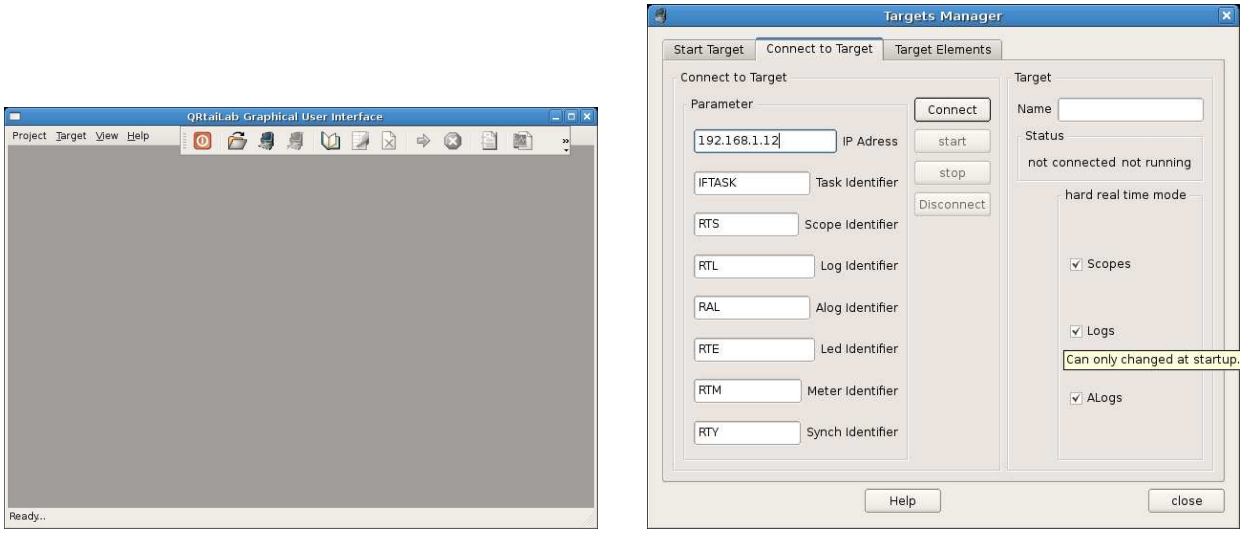
ausgeführt werden.

10.3. Einführung in QRtaiLab

Um *QRtaiLab* starten zu können, müssen als erstes die RATI-Module geladen werden. Dann kann *QRtaiLab* vom Installationsverzeichnis im Terminal mit dem Befehl

```
./qrtailab
```

gestartet werden.
In Abb. 10.1 ist der Startbildschirm von *QRtaiLab* zu sehen. Um eine Verbindung zu einem laufendem Programm herstellen zu können, muss auf den Button mit dem Ordner geklickt werden, und es öffnet sich der Target Manager der ebenfalls in dieser Abbildung gezeigt wird.

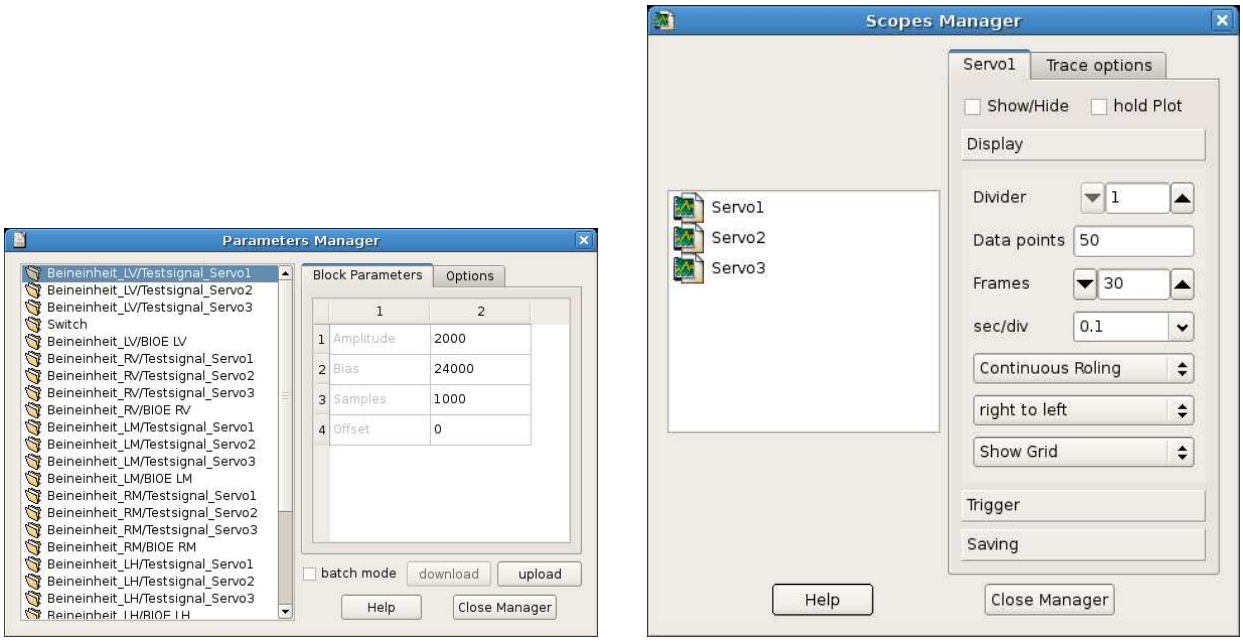


(a) Startbildschirm (b) Target Manager

Bild 10.1.: Startbildschirm und Target Manager von QRtaiLab

Im Target Manager muss nun die IP-Adresse des Rechners eingetragen werden, auf dem der Linux-Prozess ausgeführt wird. Diese ist standardmäßig auf 127.0.0.1 gestellt, welche immer auf den eigenen Computer verweist. Wenn eine Verbindung zu einem anderen Computer oder Embedded Board hergestellt werden soll, muss nur dessen IP-Adresse eingetragen werden und

auf den Button *Connect* gedrückt werden. Kann die Verbindung zu dem laufendem Programm hergestellt werden, erscheint der Name des Programmes im rechten Teil des *Target Manager*. Nun sind auch die restlichen Buttons im Startbildschirm aktiviert und der *Target Manager* kann geschlossen werden.
Zwei weitere wichtige Funktionen sind der *Parameter Manager* und der *Scope Manager*. Beide können durch den entsprechenden Button im Startbildschirm aufgerufen werden und sind in Abb. 10.2 abgebildet.
Der *Parameter Manager* dient zum Anzeigen und Ändern von Parametern in den Eingabeblöcken. In der linken Spalte sind alle Eingabeblöcke (z.B.: Sinusgenerator) dargestellt, die das Programm verwendet. Um eine davon zu manipulieren, wird dieser mit einem Doppelklick ausgewählt und in der rechten Spalte werden dessen Parameter angezeigt. Diese Parameter können mit der Maus ausgewählt und verändert werden. Mit dem Button *upload* oder mit der Entertaste kann die Veränderung bestätigt werden und der geänderte Wert wird an das laufende Programm gesendet.
Die Anzeige und Speicherung von Daten wird mit dem *Scope Manager* durchgeführt. In der linken Spalte des *Scope Managers* werden die Ausgabeblöcke angezeigt. Diese können mit der Maus ausgewählt werden. In der rechten Spalte werden Parameter angezeigt, die zur Veränderung der Darstellung der Anzeigen und zur Speicherung von Daten dienen.



(a) Parameter Manager (b) Scope Manager

Bild 10.2.: Manipluation und Darstellung von Parametern und Daten

11 *ARToolKit*

11.1. Was ist *ARToolKit*

ARToolKit (AR, Augmented Reality) ist eine Softwarebibliothek zur Ermittlung der Lagedaten von grafischen Markern in einem per Kamera aufgenommenen Abbild der Realität. Die Lagedaten werden in Form einer 3x4 Matrix geliefert und bestimmen die Position der Marker relativ zur Kamera. Mithilfe der am Sechsbeiner montierten Kamera ist es somit möglich die aktuelle Position desselben zu ermitteln (sofern sich ein Marker im Einzugsbereich befindet). Diese Funktion findet im Automatikmodus Anwendung, in dem sich der Roboter nach absolvierter Sollbahn wieder anhand des an der Wand befestigten Markers eigenständig neu ausrichtet, um anschließend die nächste Runde zu beginnen (s. Kap. 13.3).
Abbildung 11.1 zeigt einen typischen *ARToolKit*-Marker. Der schwarze Rahmen dient zur Erkennung des Markers und zur Bestimmung der meisten Lagedaten. Das Muster im weißen Feld dient zur Identifizierung des Markers und zur Bestimmung der Ausrichtung.

Bild 11.1.: *ARToolKit*-Marker

In Abb. 11.2 ist die Lage der beiden Koordinatensysteme (Marker- und Kamerasystem) dargestellt.

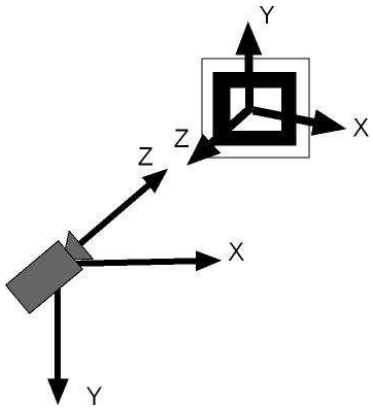


Bild 11.2.: Lage der Koordinatensysteme

11.2. Installation der Kamera

Da eine Kernelversion <2.6.26 verwendet wird, ist es zur Installation der Kamera nötig den entsprechenden UVC-Videotreiber[1] auf dem Host-System zu installieren (in neueren Kernelversionen ist dieser bereits enthalten). Die heruntergeladene Archivdatei wird nach */usr/src/* entpackt und der Treiber mit

```
make all && make install
```

kompiliert und installiert. Das dabei entstandene Modul *uvcvideo.ko* wird jetzt auf den Zielrechner kopiert. Durch Ausführen des Startskriptes 11.1 im Verzeichnis */etc/rcS.d/* (wird wiederum beim Hochfahren automatisch ausgeführt) wird der Treiber geladen und die Kamera initialisiert.

Listing 11.1: S60camera

```
1 # init Camera
2 rm /dev/video
3 rm /dev/video0
4
5 modprobe uvcvideo
6 mknod /dev/video0 char 81 0
7 chmod 644 /dev/video0
8 chgrp users /dev/video0
9 ln /dev/video0 /dev/video
```

11.3. Installation von *ARToolKit*

11.3.1. Vorbereitung

Zur Installation von *ARToolKit* müssen zuerst folgende Pakete mit `apt-get install` oder einem Paketmanager installiert werden:

- `build-essential`
- `gphoto2`
- `libclanlib-dev`
- `libglut3-dev`

11.3.2. Herunterladen und Patchen der Quelldateien

Verwendet wird die ARToolKit-Version 2.71.3[2], da dafür ein Patch[3] zur Kompatibilität mit Video4Linux2 existiert. Nach dem Entpacken der Quelldateien mit

```
tar zxvf ARToolKit-2.71.3.tgz
```

nach */usr/src/* wird der Patch eingespielt. Dazu wird unter */usr/src/* der Befehl

```
patch -p0 < artk-v4l2.patch.txt
```

ausgeführt.

[1] http://linuxtv.org/hg/~pinchartl/uvcvideo/
[2] http://sourceforge.net/projects/artoolkit/files/
[3] http://www.hitlabnz.org/forum/showthread.php?t=432

11.3.3. Installation am Host-System

Durch Ausführen von

```
./configure
```

wird das Konfigurationsmenü gestartet, in dem folgende Einstellungen gewählt werden:

```
Select a video capture driver: Video4Linux2
Build qsub lib. with texture rectangle support? (y or n): y
```

Im Verzeichnis */usr/src/ARToolKit/* werden ausschließlich die Verzeichnisse

- *./bin/Data/*
- *./include/AR/*
- *./lib/*

benötigt, alle anderen Dateien können gelöscht werden. Außerdem werden in der Datei *./lib/SRC/Gl/gsub.c* lediglich die Funktionen

- `argInt()`
- `argConvGLcpara()`
- `argConvGLcpara2()`

benötigt. Auch hier werden alle anderen Funktionen gelöscht.
Abschließend wird der Quellcode durch Ausführen von

```
make
```

im Verzeichnis */usr/src/ARToolKit/lib/SRC/* kompiliert.

11.3.4. Installation am Zielsystem

Zur Installation am eingebetteten System werden folgende Bibliotheken benötigt:

- Kopieren der *ARToolKit* Bibliotheken von */usr/src/ARToolKit/lib/* des Entwicklungsrechners auf */lib/* des Zielsystems.
  - *libAR.a*
  - *libARgsub.a*
  - *libARgsub_lite.a*
  - *libARgsubUtil.a*
  - *libARMulti.a*
  - *libARvideo.a*
- Kopieren der Grafikbibliotheken von */lib/* des Entwicklungsrechners auf */lib/* des Zielsystems.
  - *ld-linux.so.2*
  - *libcrypto.so.0.9.7*
  - *libcrypt.so.1*

  - *libc.so.6*
  - *libdl.so.2*
  - *libdrm.so.2*
  - *libgcc_s.so.1*
  - *libGL.so*
  - *libGLU.so*
  - *libglut.so*
  - *libm.so.6*
  - *libncurses.so.5*
  - *libnsl.so.1*
  - *libnss_files.so.2*
  - *libpam_misc.so.0*
  - *libpam.so.0*
  - *libpthread.so.0*
  - *libreadline.so.4*
  - *libresolv.so.2*
  - *libssl.so.0.9.7*
  - *libstdc++.so.6*
  - *libusb-0.1.so.4*
  - *libutil.so.1*
  - *libwarp.so.0*
  - *libX11.so.6.2.0*
  - *libXau.so.6*
  - *libXdmcp.so.6*
  - *libXext.so.6.4.0*
  - *libXxf86vm.so.1*
  - *libz.so*

- Anlegen von Links auf die Bibliotheken
  - *libGL.so.1* → *libGL.so*
  - *libGLU.so.1* → *libGLU.so*
  - *libglut.so.3* → *libglut.so*
  - *libm.so* → *libm.so.6*
  - *libX11.so.6* → *libX11.so.6.2.0*
  - *libXext.so.6* → *libXext.so.6.4.0*
  - *libz.so.1* → *libz.so*

11.4. Verwendung von *ARToolKit*

Zur Verwendung von *ARToolKit* wird ein Nicht-Echtzeitprogramm entwickelt. Die Funktionen zur Ermittlung der Lagedaten werden zu einem großen Teil aus [AW08] übernommen. Zur Kommunikation mit dem Echtzeitprogramm werden FIFOs benutzt. Dabei handelt es sich um bidirektionale Pufferspeicher, über die Daten zwischen einem *RTAI*-Task und einem normalen Linux-Prozess ausgetauscht werden können. Linux-Prozesse können auf einen FIFO wie auf eine normale Datei zugreifen. Anstelle einer Datei öffnet man mit der Funktion open() einen speziellen Device-Knoten im */dev*-Verzeichnis (rtf0 bis rtf63).

- `open()` ... Öffnen eines FIFOs
- `read()` ... Lesen der Daten
- `write()` ... Schreiben von Daten

Auf RTAI-Seite stehen folgende Funktionen zum arbeiten mit FIFOs zur Verfügung:

- `rtf_create()` ... Erzeugt einen FIFO mit gegebener Größe und Nummer
- `rtf_destroy()` ... Löscht einen FIFO
- `rtf_reset()` ... Löscht den Inhalt eines FIFO
- `rtf_put()` ... Schreibt Daten in den FIFO
- `rtf_get()` ... Liest Daten aus dem FIFO
- `rtf_create_handler()` ... Registriert einen Handler, der beim Eintreffen von Daten ausgeführt wird

Die Funktion des gesamten Programms ist schematisch im Flussdiagramm in Abb. 11.3 dargestellt. Nach dem Initialisieren der Kamera und Öffnen der FIFOs wird auf eine Anforderung der Lagedaten aus dem Echtzeitprogramm gewartet. Diese erfolgt durch das Schreiben des Wertes „1“ in den FIFO0. Anschließend werden durch Aufruf der Funktion `cam_get_mat()` die aktuellen Lagedaten ermittelt. Diese erhält man in Form einer 3x4-Matrix:

$$\begin{pmatrix} \underbrace{\begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{matrix}}_{A_{cm}} & \underbrace{\begin{matrix} r_x \\ r_y \\ r_z \end{matrix}}_{{}_c r_{cm}} \end{pmatrix} \tag{11.1}$$

Dabei entspricht die Matrix A_{cm} der Rotationsmatrix zwischen Kamera- und Markerkoordinatensystem, aus der mit

$$\gamma = atan2\left(\frac{-a_{21}}{a_{11}}\right) \tag{11.2}$$

$$\beta = atan2\left(\frac{a_{31}}{a_{11}cos(\gamma) - a_{21}sin(\gamma)}\right) \tag{11.3}$$

$$\tag{11.4}$$

die Verdrehung β des Roboters um die Hochachse berechnet werden kann (s. [Gat09]). Alle anderen Verdrehungen werden vernachlässigt und in Folge nicht behandelt. Der Vektor ${}_c r_{cm}$

wiederum stellt den Verschiebungsvektor zwischen den Koordinatensystemursprüngen im Kamerakoordinatensystem dar.
Nach erfolgreicher Ermittlung der Lagedaten werden diese in den FIFO1 geschrieben und damit dem Echtzeitprogramm zur Verfügung gestellt. Ist ein Fehler aufgetreten, werden alle Werte auf 0 gesetzt. Der gesamte Quellcode befindet sich in der Datei *camera.c*[4]. Als Referenz zu C-Programmierung wurde [BK88] verwendet.

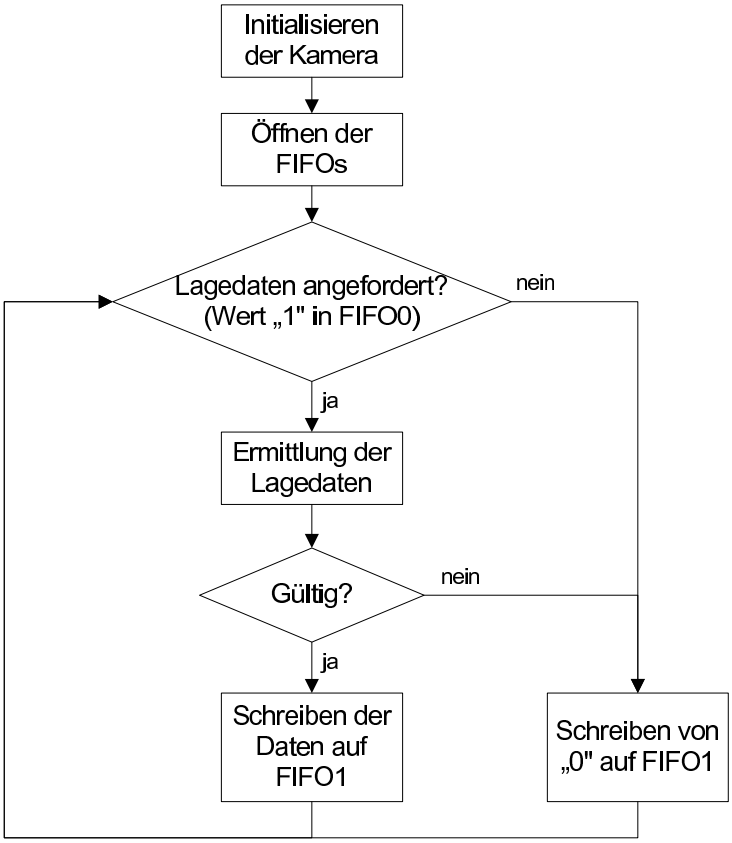


Bild 11.3.: Flussdiagramm der Lagedatenermittlung

[4]DVD:/Kamera/camera.c

12 *RTAI-XML*

12.1. Was ist *RTAI-XML*

RTAI-XML ist eine Serverkomponente des *RTAI*-Projekts, die es ermöglicht relativ einfach Steuerungsapplikationen für Echtzeitprogramme zu entwerfen.

Bild 12.1.: *RTAI-XML* Strukturdiagramm

Die Kommunikation mit *RTAI-XML* erfolgt über einen *XML-RPC* Server der auf eingehende Anfragen wartet. Für *XML-RPC* existieren eine große Anzahl vom Implementierungen in den verschiedensten Programmiersprachen und Plattformen. Für diese Projektarbeit wird eine Java Implementierung verwendet (s. Kap. 14).

12.2. Installation von *RTAI-XML*

Zur Installation von *RTAI-XML* muss zuerst der Sourcecode heruntergeladen[1] und entpackt werden. Damit *RTAI-XML* am Target-Rechner lauffähig wird, müssen die Dateien *rslave_builder* und das Makefile etwas modifiziert werden.
In *rslave_builder* muss die Zeile

```
1 if [ "$target_version" \< "rt_main version 3.4.4" ]; then interface
    =0; fi
```

auskommentiert und im Makefile bei „rtmanager“ und „rtaixml“ jeweils der Parameter -static hinzugefügt werden:

```
1 rtaixml:          rtaixml.cpp
2         $(CXX) -static -Wall -Wstrict-prototypes -O2 -I. -I$(MASTER-
              DIR) -I$(SLAVE-DIR)  -I$(RTAI-DIR)/include -I$(XMLRPC-DIR)
              /src -o $@ $<  $(slave_objects)  -lpthread
3
4 rtmanager:        rtaixml_master.cpp rtaixml_master.h
5         $(CXX) -static -Wall -Wstrict-prototypes -O2 -I. -I$(MASTER-
              DIR) -I$(XMLRPC-DIR)/src -o $@ $<  \$(master_objects)  -
              lpthread
```

[1] http://artist.dsi.unifi.it/rtaixml/

Sind die Änderungen vorgenommen wird *RTAI-XML* durch ausführen von `make all` und `make install` installiert.

12.2.1. Portierung auf das Target-System

Zur Portierung auf das Target-System wird nun das Installationsverzeichnis am Entwicklungsrechner[2] auf selbiges am Sechsbeiner kopiert. Durch die statische Kompilierung sind keine weiteren Bibliotheken notwendig.

12.2.2. Starten von RTAI-XML

Bevor der RT-Manager gestartet werden kann muss noch ein Skriptfile im Ordner */usr/realtime/rtaixml/script* angelegt werden, in dem Name und Pfad des Echtzeitprogrammes eingetragen werden.

Listing 12.1: IFTASK

```
1  #HOWTO in brief
2  #       >> The filename MUST be composed at maximum by 6 chars.
3  #Uncomment the following line if before executing this target you
4  # need to start another one defined by the variable depend.
5  #(This can be used recursively).
6  #Name of the  target.
7
8  #depend=
9
10 #[optional] Introduce a delay before target execution.
11 #delay=0.0
12
13 #Name of the target executable
14 target=Steuerung
15
16 #Absolute path
17 dir=/usr/src
18
19 #[optional] Real time priority
20 priority=8
21
22 #Final time
23 time=100000000000
24
25 #[optional] Sampling time (SCICOS only!)
26 #tsamp=
27
28 #[optional] Starting option.
29 #wait=true
30
31 #[optional] Verbose output
32 verbose=true
33
```

[2] standardmäßig */usr/realtime/rtaixml*

```
34 #[optional] Log file
35 #log=true
36
37 #[optional] Name of the real time Scope
38 rts=RTS
39
40 #[optional] Name of the real time Digital (Led)
41 rte=RTE
```

Jetzt kann der *XML-RPC* Server durch Ausführen von `rtmanager` gestartet werden. Das geschieht am besten automatisch in einem Startskript beim Hochfahren des Systems. Der Server wartet auf eine Verbindungsanfrage und startet bei deren eintreffen automatisch den, im Skript 12.1 eingetragenen Echtzeittask und übergibt dessen Steuerung an den anfragenden Client.

13 Gangsteuerung

Zur Steuerung des Roboters wurde zum größten Teil das bereits vorhandene MatLab/SimuLink Model *Gangsteuerung.mdl* verwendet. Die drei Hauptmodifikationen sind:

- Erstellen und Einfügen von BIOE-Blöcken zur Kommunikation mit der Hardware
- Erweiterung um eine Steuerung für den Fahrmodus
- Erstellen eines Sollbahngenerators zu Demonstrationszwecken

13.1. Anbindung der BIOEs

Das verwendete MatLab/SimuLink - Modell diente ursprünglich der Simulation des Roboters. Die Servomotoren in den Gelenken wurden durch entsprechende Übertragungsfunktionen modelliert. Diese Übertragungsfunktionen werden durch je einen BIOE-Block pro Bein ersetzt (s. Abb. 13.1). Die Blöcke *q2pwm* und *adc2q* dienen zum Anpassen der Wertebereiche der Eingangs- und Ausgangssignale (umrechnen Winkel→PWM, bzw. ADC→Winkel).
Unter dem BIOE-Block verbirgt sich eine C-Code s-Function mit dem Quellcode *BIOE_Device.c*. Ausschnitte daraus befinden sich im Listing 13.1. Der gesamte Quellcode befindet sich auf der beiliegenden Projekt-DVD[1]. Die Implementierung der benötigten BIOE-Funktionen wird der Datei *bioe.c* entnommen. Durch Doppelklick auf den BIOE-Block kann dieser parametrisiert werden (s. Abb. 13.2). Die wichtigsten Einstellungen sind *Device* in dem die Adresse des jeweiligen Beines (0 . . . 5), sowie *DTN* mit dem die Betriebsart des BIOEs eingestellt werden. Die genaue Bedeutung aller Parameter kann der BIOE-Beschreibung [Wei08] entnommen werden.

Listing 13.1: Auszüge aus BIOE_Device.c

```
1  .
2  .
3  .
4  /* definieren der Parameter */
5  #define NUM_PARAMS                (7)
6  #define SAMPLE_TIME               (ssGetSFcnParam(S,0))
7  #define PORT                      (int)mxGetScalar((ssGetSFcnParam(S,1)
       ))
8  #define CHANNEL                   (int)mxGetScalar((ssGetSFcnParam(S,2)
       ))
9  #define DEVICE                    (int)mxGetScalar((ssGetSFcnParam(S,3)
       ))
10 #define DTN                       (int)mxGetScalar((ssGetSFcnParam(S,4)
       ))
11 #define SPEEDFACTOR               (int)mxGetScalar((ssGetSFcnParam(S,5)
       ))
```

[1] DVD:/Gangsteuerung/BIOE_Device.c

```
12 #define PAUSEFACTOR                    (int)mxGetScalar((ssGetSFcnParam(S,6)
       ))
13 .
14 .
15 .
16 #define MDL_START
17 #if defined(MDL_START)
18
19 static void mdlStart(SimStruct *S)
20 {
21     BIOE_Init(PORT);
22     //BIOE konfigurieren
23     BIOE_ConfigTransaction(PORT, CHANNEL, DEVICE, 0, DTN, SPEEDFACTOR
           , PAUSEFACTOR);
24 }
25 #endif /*  MDL_START */
26
27 static void mdlOutputs(SimStruct *S, int_T tid)
28 {
29     int32_T *y1 = (int32_T *) ssGetOutputPortSignal(S, 0);
30     int32_T *u1 = (int32_T *) ssGetInputPortSignal(S, 0);
31     *y1 = BIOE_Transaction(PORT, CHANNEL, DEVICE, 2, *u1, SPEEDFACTOR
           , PAUSEFACTOR);
32     //gleiches für alle anderen verwendeten EPs...
33 }
34
35 static void mdlTerminate(SimStruct *S)
36 {
37     //BIOE schliessen
38     BIOE_Close(PORT);
39 }
40
41 /* BIOE-Functions:
42  * int BIOE_ReadWrite(int basePort, int Value, int CS, int CLK, int
       speedfactor)
43  * int BIOE_Transaction(int basePort,int channel,int Device,int
       Endpoint,int Value, int speedfactor, int pausefactor)
44  * int BIOE_ConfigTransaction(int basePort, int channel, int Device,
       int Action, int Data, int speedfactor, int pausefactor)
45  * void BIOE_Init(int basePort)
46  * void BIOE_Close(int basePort)
47  * int BIOE_DeviceExists(int basePort, int channel, int Device)
48  */
49
50 //Hier kommt die Implementierung der obigen BIOE-Funktionen... (
       kopiert aus bioe.c)
```

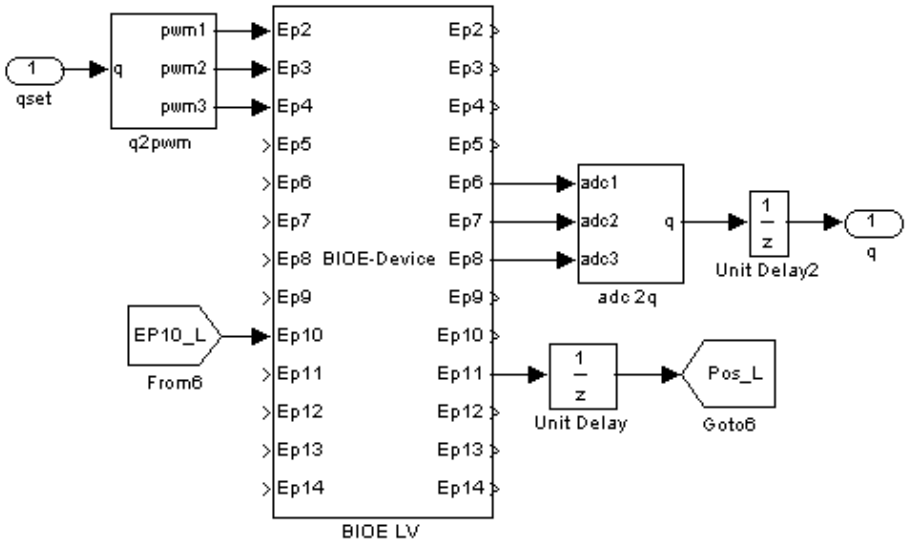


Bild 13.1.: BIOE-Anbindung

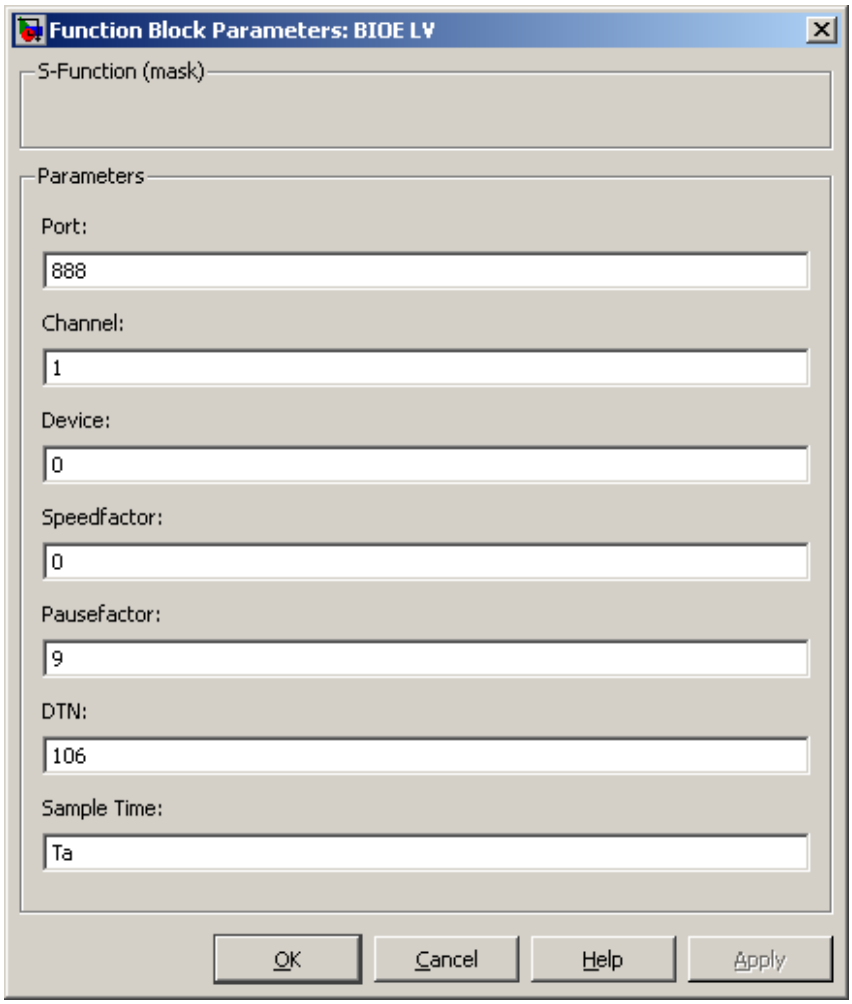


Bild 13.2.: BIOE Konfiguration

13.2. Erweiterung um den Fahrmodus

13.2.1. Berechnung der Winkel und Geschwindigkeiten

Abbildung 13.3 zeigt eine Skizze des Roboters im Fahrmodus. Die mittleren Beine sind dabei ausgeblendet, da diese blockiert sind und daher im Fahrmodus angehoben werden. Um beim Fahren ein Quergleiten der Räder zu verhindern, müssen sich die Drehachsen der Räder genau im Geschwindigkeitspol des Roboters schneiden. Die entsprechenden Winkel α und β berechnen sich mit [Bar91] aus den Gleichungen

$$\alpha = \arctan \frac{L/2}{r - B/2} \tag{13.1}$$

und

$$\beta = \arctan \frac{L/2}{r + B/2}. \tag{13.2}$$

Zusätzlich werden, zur Berechnung der Radgeschwindigkeiten, die Abstände R_1 und R_2 vom Geschwindigkeitspol zu den beiden angetriebenen Rädern benötigt. Diese errechnen sich aus

$$R_{1,2} = \sqrt{(r \mp B/2)^2 + (L/2)^2} \mp L_b. \tag{13.3}$$

Damit können die Radgeschwindigkeiten aus

$$\left(\frac{v}{r}\right) = \left(\frac{v_1}{R_1}\right) = \left(\frac{v_2}{R_2}\right) \tag{13.4}$$

ermittelt werden.

13.2.2. Modifikationen der bestehenden Zustandsautomaten

Zustandsautomat des gesamten Roboters

Abbildung 13.4 zeigt den Zustandsautomat des gesamten Roboters. Dabei war der schwarze Teil bereits vorhanden und der rot gefärbte Teil wird neu hinzugefügt. Nach Starten des Fahrmodus wird in den entsprechenden Zustand gewechselt, was dem Start des Zustandsautomaten für den Fahrmodus aus Kap. 13.2.3 entspricht. Nach Beenden desselben wird erst abgewartet, bis der Roboter seine Ausgangsposition erreicht hat, bevor wieder in den Grundzustand gewechselt wird. Das dient dazu, dass beim Starten des Gehmodus ein definierter Zustand vorherrscht, um sonst auftretende Kollisionen der Beine zu verhindern.

Zustandsautomat für die Beinbewegung

Der Zustandsautomat für die Beinbewegung (Abb. 13.5) bleibt weitgehend unverändert und wird lediglich um einen Zustand erweitert. Dieser dient wiederum dazu, die Beine nach Beenden des Gehmodus wieder in Ausgangsposition zu bringen.

13.2.3. Zustandsautomat für den Fahrmodus

Abbildung 13.6 zeigt den Zustandsautomat des Fahrmodus. Nach dessen Start werden zuerst paarweise (mitte - rechts vorne/links hinten - links vorne/rechts hinten) alle Beine in die gewünschten Positionen gebracht. Anschließend werden die mittleren (blockierten) Beine angehoben und die Fahrmotoren aktiviert. Um zu starke Kräfte an den oberen Servos (Hüfte) zu vermeiden, werden bei einer Änderung des Radius die Motoren deaktiviert.

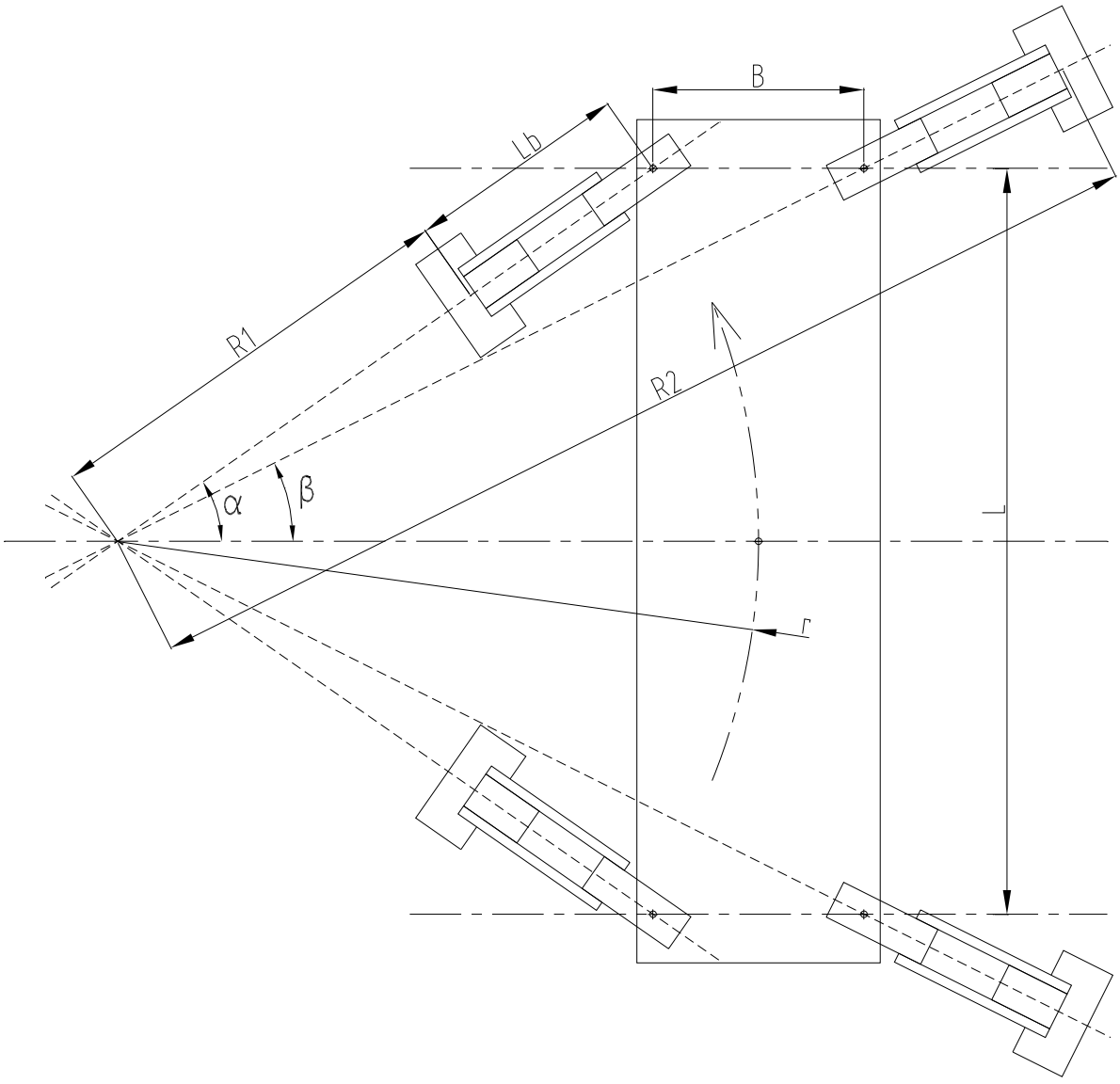


Bild 13.3.: Sechsbeiner im Fahrmodus

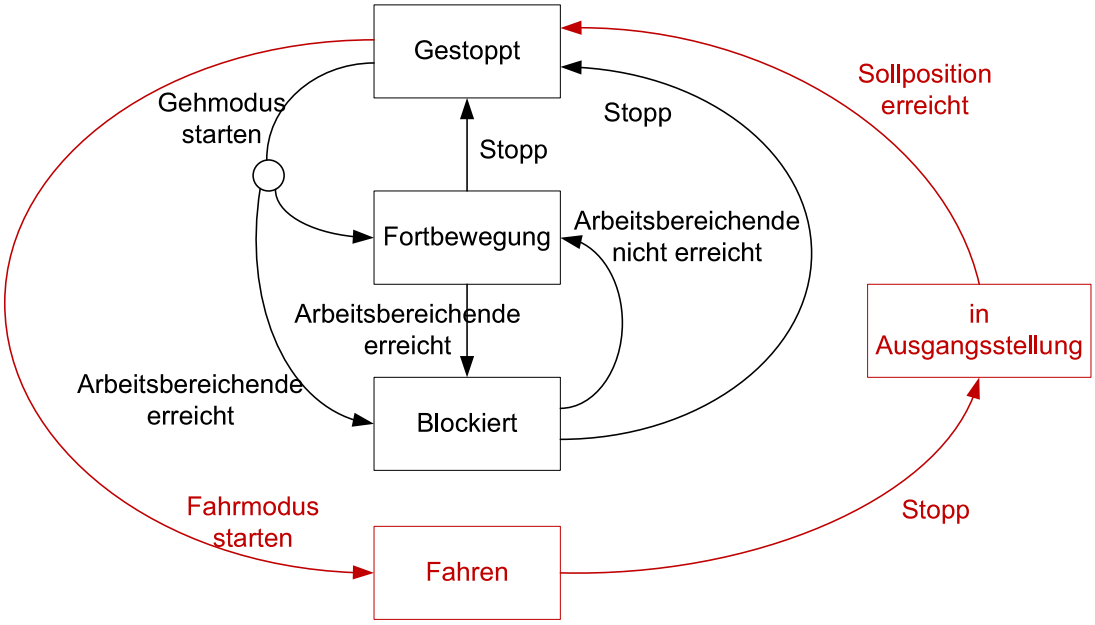


Bild 13.4.: Zustandsautomat für den gesamten Roboter

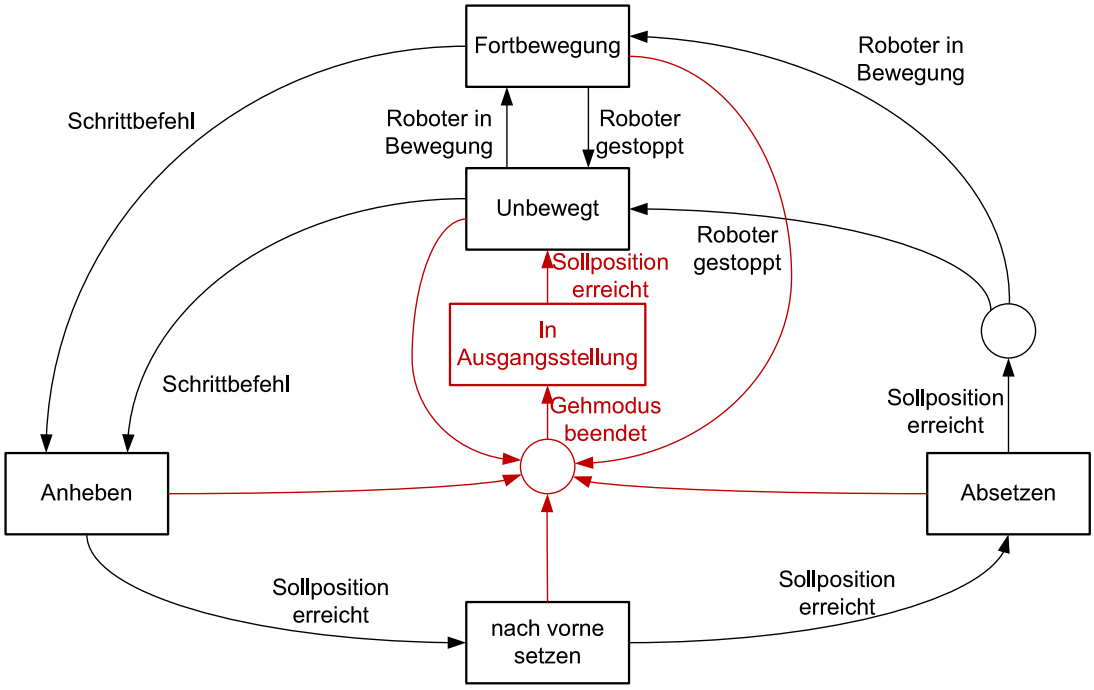


Bild 13.5.: Zustandsautomat für die Beinbewegung

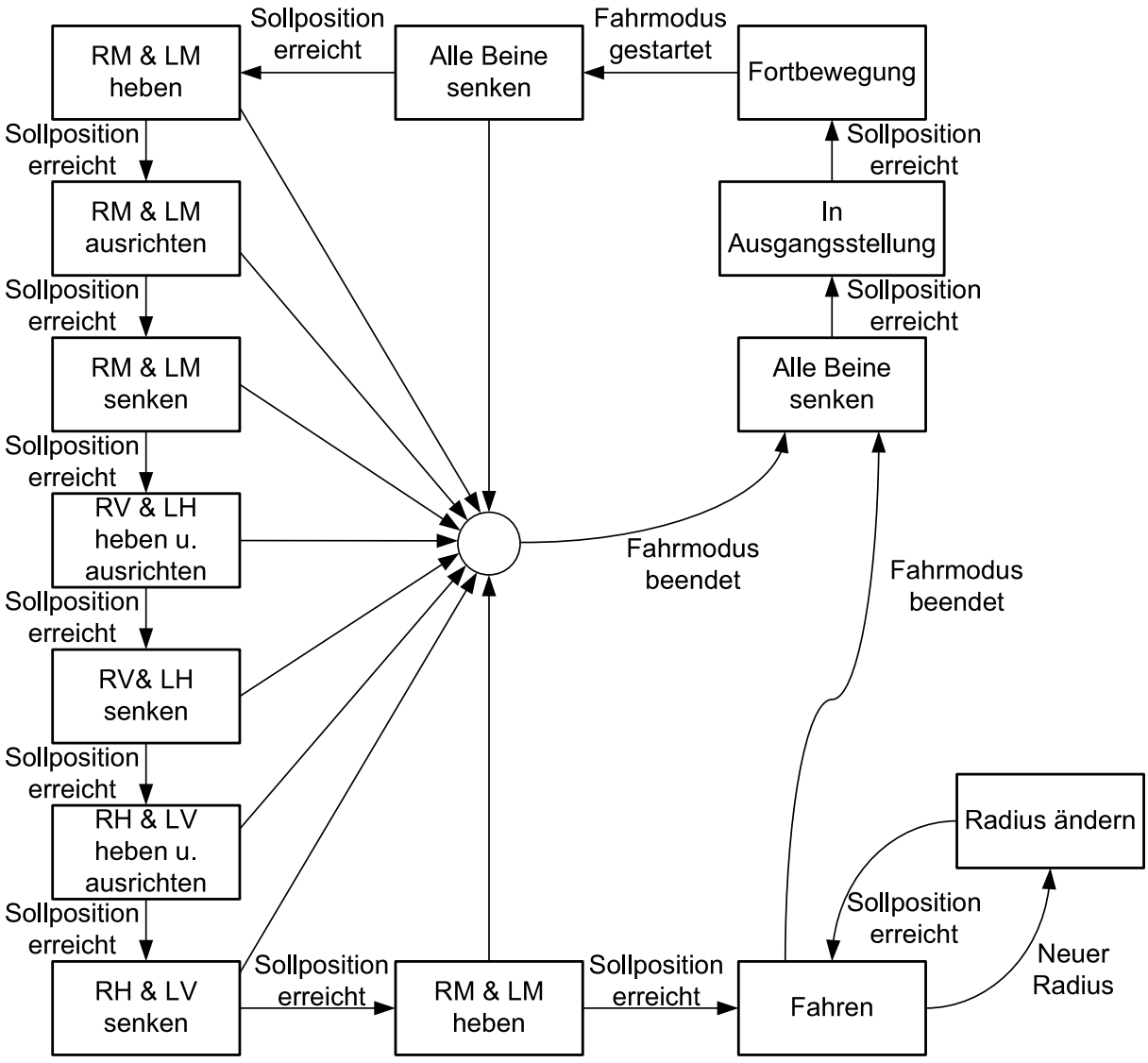


Bild 13.6.: Zustandsautomat des Fahrmodus

13.3. Sollbahngenerator

Zu Demonstrationszwecken wird abschließend ein Sollbahngenerator implementiert der 2 Sollbahnen erzeugt, welche sämtliche Funktionen des Roboters zeigen sollen. Die generierten Bahnen sind in Abb. 13.7 und Abb. 13.8 skizziert. An den Ausrichtungspunkten bringt sich der Roboter selbstständig mithilfe von *ARToolKit* in eine definierte Sollposition (Position und Verdrehung), bevor der Sollbahn weiter gefolgt wird.

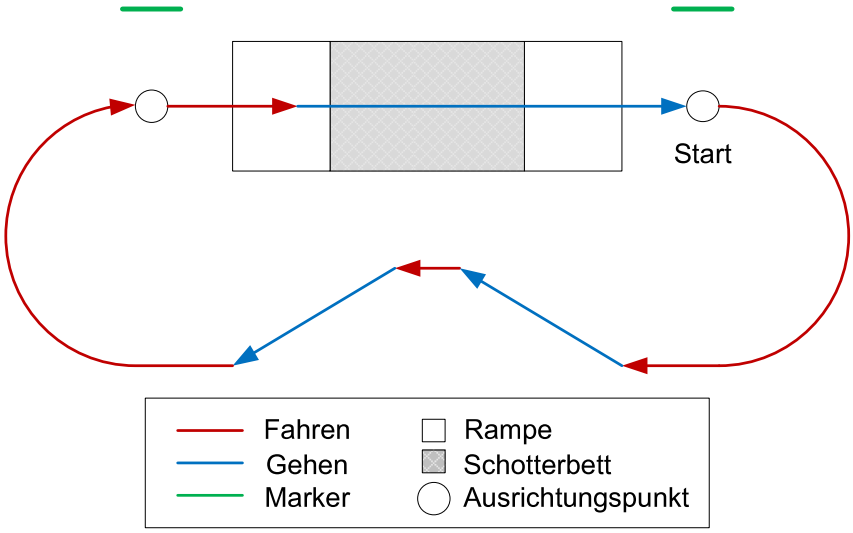


Bild 13.7.: Sollbahn 1

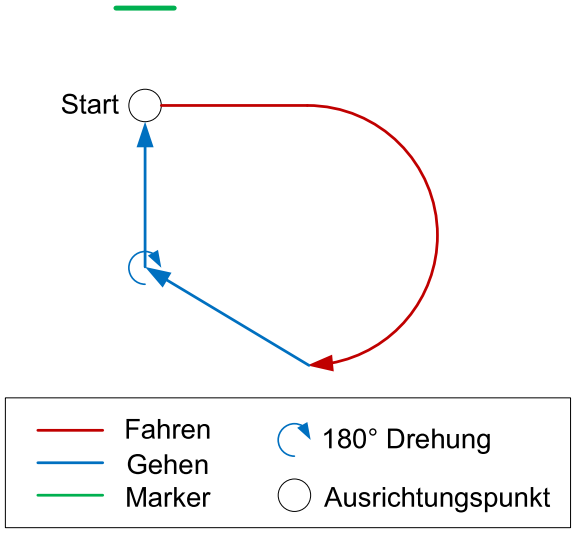


Bild 13.8.: Sollbahn 2

13.4. Generieren und Kompilieren des Echtzeitcodes

Bevor der Echtzeitcode erzeugt werden kann, müssen noch einige Simulationsparameter eingestellt werden. Durch Drücken von *Strg+e* wird das Konfigurationsfenster geöffnet. Jetzt kann der verwendete Solver und die Abtastzeit (s. Abb. 13.9) eingestellt werden. Außerdem muss

im Bereich *Real-Time Workshop* noch *RTAI* als „System target file“ eingestellt werden (s. Abb. 13.10). Jetzt kann die Codegenerierung mit einem Klick auf den *Build*-Button gestartet werden.

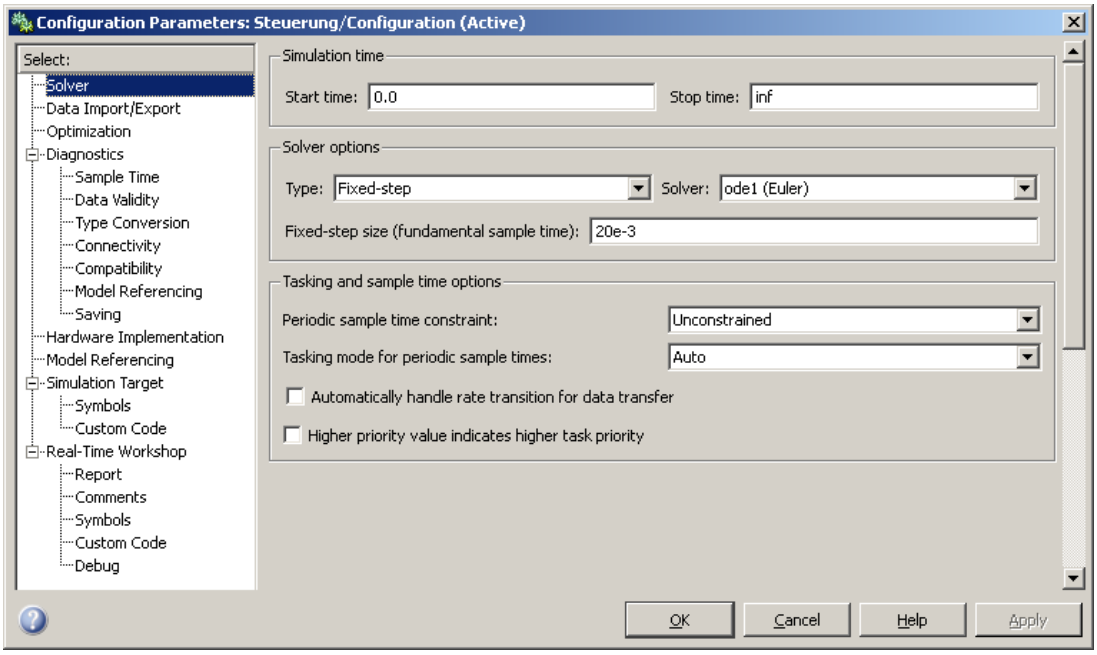


Bild 13.9.: Konfiguration des Solvers

Dadurch wird der Ordner *Steuerung_rtai*, in welchem sich der Echtzeitcode inklusive dem zugehörigem Makefile befindet erzeugt. In diesen Ordner werden zusätzlich noch die verwendeten Matlab S-Functions

- *beinkoordination.c*
- *sollbahn.c*
- *BIOE_Device.c*
- *beinwinkel.c*
- *fahrsteuerung.c*
- *sollgeschwindigkeiten.c*
- *counter.c*
- *sq2a.c*
- *sq2g.c*

kopiert. Da bei der Steuerung mit *RTAI-XML* normalerweise bei Änderung eines Parameters auch sämtliche andere aktualisiert werden, müssen noch zwei Dateien angepasst werden. In der Datei *Steuerung_pt.c* werden aus der Konstanten *rtBlockTuning[]* alle Parameter, außer die zur Steuerung des Roboters notwendigen, gelöscht (s. Code 13.2). Jetzt muss noch in der Datei *Steuerung.c* die Variable `mapInfo.Parameters.numBlockTuning` auf die Anzahl der verbleibenden Parameter (in diesem Fall 5) gesetzt werden.

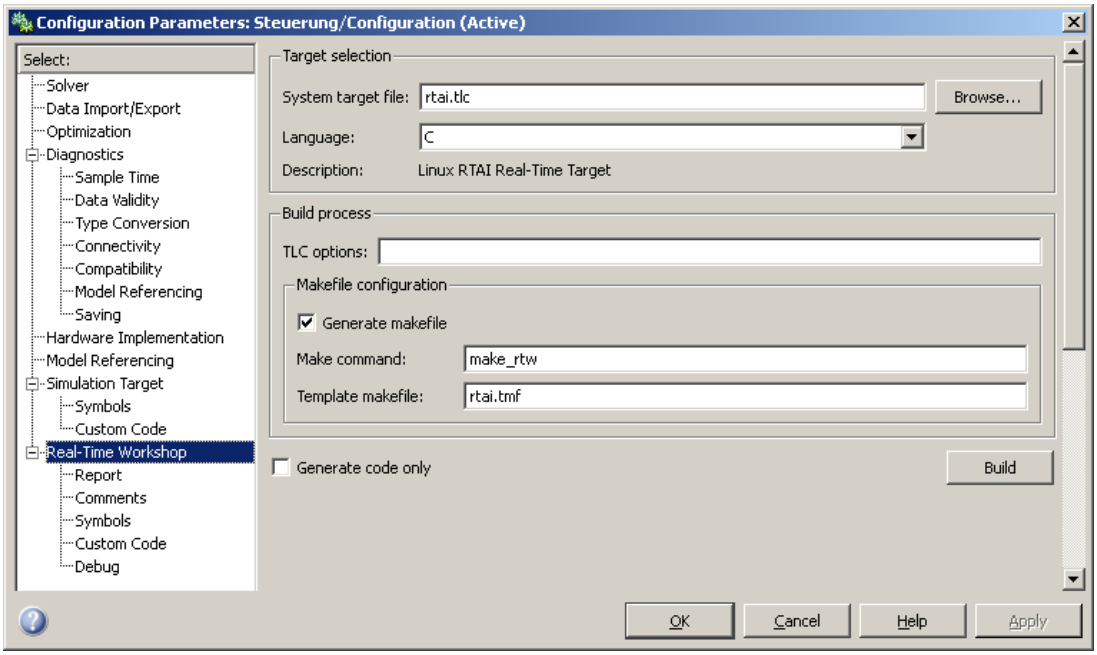


Bild 13.10.: Konfiguration des Real-Time Workshops

Listing 13.2: Auszug aus *Steuerung_pt.c*

```
/* Tunable block parameters */

static const BlockTuning rtBlockTuning[] = {
  /* blockName, parameterName,
   * class, nRows, nCols, nDims, dimsOffset, source, dataType,
      numInstances,
   * mapOffset
   */
  /* Constant */
  {"Steuerung/Go!", "Value",
    {rt_SCALAR, 1, 1, 2, -1, rt_SL_PARAM, SS_DOUBLE, 1, 0}
  },
  /* Constant */
  {"Steuerung/v", "Value",
    {rt_SCALAR, 1, 1, 2, -1, rt_SL_PARAM, SS_DOUBLE, 1, 1}
  },
  /* Constant */
  {"Steuerung/Kruemmung", "Value",
    {rt_SCALAR, 1, 1, 2, -1, rt_SL_PARAM, SS_DOUBLE, 1, 2}
  },
  /* Constant */
  {"Steuerung/alpha", "Value",
    {rt_SCALAR, 1, 1, 2, -1, rt_SL_PARAM, SS_DOUBLE, 1, 3}
  },
  /* Constant */
  {"Steuerung/auto", "Value",
    {rt_SCALAR, 1, 1, 2, -1, rt_SL_PARAM, SS_DOUBLE, 1, 31}
  },
```

```
28    {NULL, NULL,
29      {(ParamClass)0, 0, 0, 0, 0, (ParamSource)0, 0, 0, 0}
30    }
31  };
```

Anschließend wird der gesamte Ordner auf den Linux-Entwicklungsrechner kopiert und durch ausführen von

```
make -f Steuerung.mk
```

der Kompilationsvorgang gestartet. Es entsteht eine ausführbare Datei *Seuerung* im übergeordneten Verzeichnis, die abschließend in den Ordner */usr/src* (bzw. dem Ordner der im Skript 12.1 angegeben wurde) des Zielsystems kopiert wird.

13.5. Ausführen des Echtzeitprogrammes

Das Echtzeitprogramm wird automatisch gestartet, wenn über das Java-Steuerungsprogramm (s. Kap. 14) eine Verbindung hergestellt wird (Drücken des *connect*-Buttons). Zu Testzwecken kann es auch sinnvoll sein, sich über das Programm jRTAILab zu verbinden. Dabei handelt es sich um eine in Java programmierte Version des bereits bekannten QRtaiLabs aus Kap. 10. Der Vorteil von jRTAILab liegt darin, dass es plattformunabhängig ist und somit auch auf Windowssystemen läuft. Zur Verbindungsherstellung müssen lediglich die IP-Adresse sowie der Name des *RTAI-XML*-Scripts 12.1 als „Target ID“ eingetragen werden (s. Abb. 13.11). Die beiden anderen Felder können unverändert bleiben. Im Unterschied zu QRtaiLab wird auch hier der Echtzeittask automatisch von *RTAI-XML* gestartet.

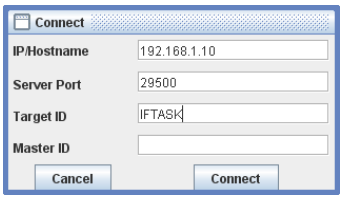


Bild 13.11.: Verbindungsherstellung mit jRTAILab

14 Java-Steuerungsprogramm

Zur Steuerung des Roboters wurde eine Java-Applikation entwickelt, die es ermöglicht ihn drahtlos über WLAN zu steuern. Die Kommunikation erfolgt dabei mittels *RTAI-XML*. Die Applikation ist in zwei Varianten verfügbar - mit (s. Abb. 14.1) und ohne (s. Abb. 14.2) 3D-Animation des 6-Beiners. Dadurch ist gewährleistet, dass sie sowohl unter Windows als auch unter Linux ausführbar ist.

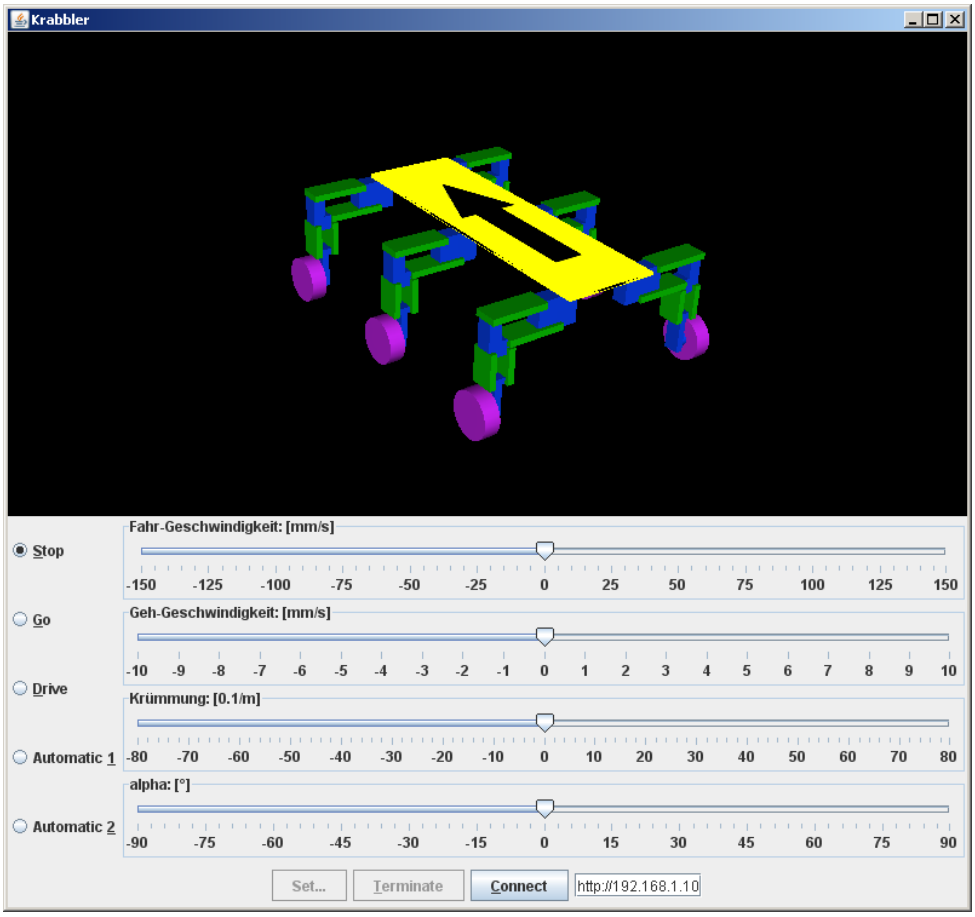


Bild 14.1.: Steuerungprogramm mit 3D-Animation

14.1. Installation und Bedienung

Um die Programme ausführen zu können muss die Java Runtime Environment 6 heruntergeladen[1] und installiert werden. Für die 3D-Animation benötigt man außerdem noch die Java3D-API ab Version 1.5.1 die ebenfalls auf dieser Seite zum herunterladen bereitsteht. Dabei sind

[1] http://www.sun.com

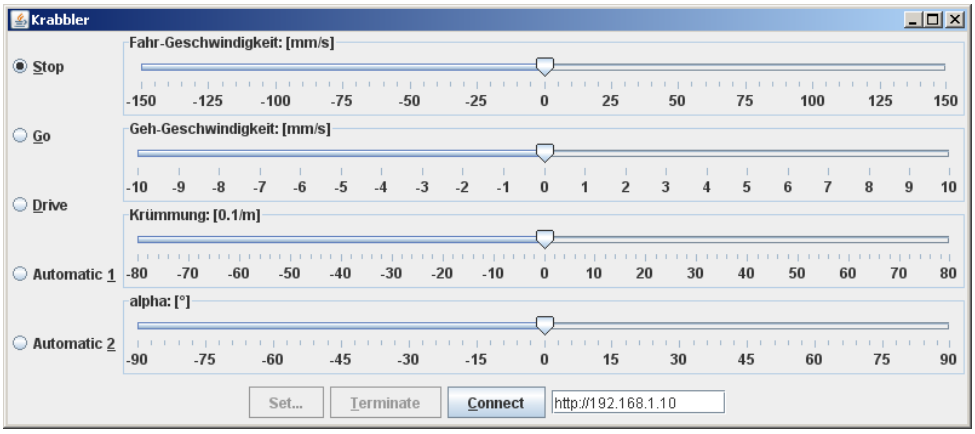


Bild 14.2.: Steuerungprogramm ohne 3D-Animation

allerdings Probleme bei der Installation unter Linux aufgetreten, die bislang nicht behoben werden konnten.

Nach dem Start des Programmes wird durch Drücken des *Connect*-Buttons die Verbindung zum Roboter hergestellt und automatisch das Echtzeitprogramm vom *RTAI-XML*-Server gestartet. Jetzt können die Servos mit dem entsprechenden Schalter am Sechsbeiner aktiviert werden (ACHTUNG: Roboter dabei anheben, um eine Überlastung der Servos zu vermeiden!), welcher daraufhin in Ausgangsstellung geht. Danach können die gewünschten Parameter (Geschwindigkeit, Bahnkrümmung, Krebswinkel und Betriebsmodus) eingestellt und mit dem *Set*-Button an den Roboter übermittelt werden. Für alle Schieberegler und Buttons wurde außerdem eine Tastatursteuerung implementiert (s. Tab. 14.1). Zum Trennen der Verbindung genügt ein Druck

Kürzel	Funktion
alt+c	Verbindung herstellen
alt+t	Verbindung trennen
alt+s	Stop
alt+d	Fahrmodus
alt+g	Gehmodus
alt+1	Automatikmodus Bahn 1
alt+2	Automatikmodus Bahn 2
Pfeil auf	Geschwindigkeit des aktuellen Modus erhöhen
Pfeil ab	Geschwindigkeit des aktuellen Modus senken
Pfeil links	Krümmung senken
Pfeil rechts	Krümmung erhöhen
Enter	Parameter senden

Tabelle 14.1.: Tastenkürzel für das Steuerungsprogramm

auf den *Terminate*-Button und das Echtzeitprogramm wird beendet. Der Roboter verharrt in seiner letzten Position. Befindet sich der Sechsbeiner im Fahrmodus, sollte dieser zuvor beendet werden da dieser auch nach Beenden der Verbindung aufrecht erhalten bleibt.

14.2. Implementierung

Als allgemeine JAVA-Referenz zur Entwicklung des Steuerungsprogrammes dienen [Mös03] und [Mös08]. Die Verwendung der Spezialpakete zur Kommunikation und Animation werden in den folgenden Kapiteln noch kurz erläutert.

14.2.1. XML-RPC

Zur Kommunikation mit *RTAI-XML* sind die entsprechenden *XML-RPC* Bibliotheken für Java notwendig[2]. Durch Einbinden dieser Bibliotheken in ein JAVA-Projekt stehen die benötigten Funktionen zur Verfügung. Für die weitere Verwendung von *RTAI-XML* und *XML-RPC* wird auf die entsprechenden Dokumentationen[3] verwiesen.

14.2.2. 3D-Animation

Zur Implementierung der 3D-Animation ist die oben erwähnte Java3D-Bibliothek notwendig. Zum Laden der Animation wird außerdem noch die Bibliothek *3d-vrml.jar* [4] benötigt, die von Hand in das Projekt einzubinden ist. Zur Verwendung sei wieder auf die Dokumentationen[5] bzw. dem Sourcecode der Steuerungsapplikation auf der beiliegenden DVD[6] verwiesen.

[2] http://ws.apache.org/xmlrpc/
[3] http://artist.dsi.unifi.it/rtaixml/ und http://www.xmlrpc.com/
[4] https://j3d-vrml97.dev.java.net/
[5] http://java.sun.com/javase/technologies/desktop/java3d/
[6] DVD://JAVASteuerungsprogramm/workspace/

A Elektronik

A.1. Akkuwächter

Bauteil	Wert	Bezeichnung
C1	47μ	Elektrolytkondensator
C2	0.1μ	Kondensator SMD 1206
D1	rot	LED SMD 1206
D2	rot	LED SMD 1206
D3	gelb	LED SMD 1206
D4	gelb	LED SMD 1206
D5	grün	LED SMD 1206
D6	grün	LED SMD 1206
D7	grün	LED SMD 1206
D8	grün	LED SMD 1206
D9	grün	LED SMD 1206
D10	grün	LED SMD 1206
D11		Diode 1N4004
IC2		IC - LM3914N
J1		Drahtbrücke
JP1		Jumper
R1	1k	Widerstand SMD 0805
R2	5k	Trimmer 15-gängig 3006P
R3	10k	Trimmer 15-gängig 3006P
R4	1k	Widerstand SMD 0805
R5	1k2	Widerstand SMD 0805
R6	2k6	Widerstand SMD 0805
R7	33k	Widerstand SMD 0805
SV2	3-polig	Stecker Raster 2.54mm

Tabelle A.1.: Bauteilliste

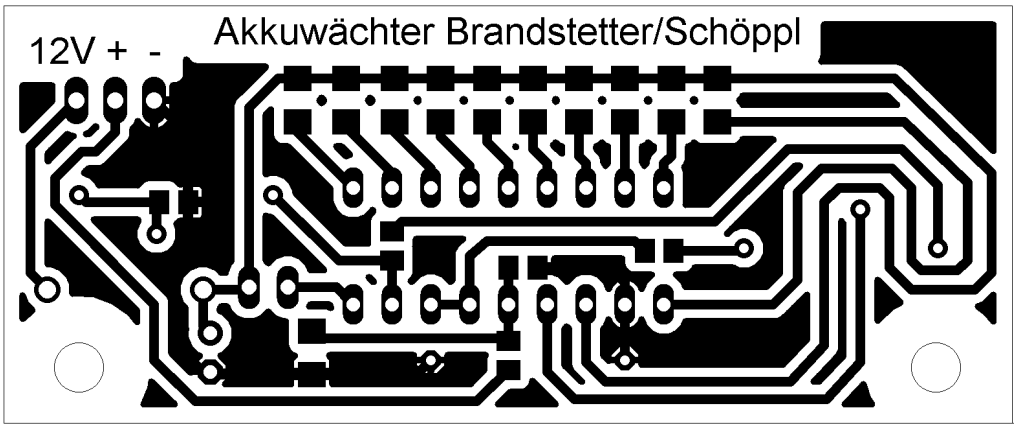


Bild A.1.: Layout Akkuwächter

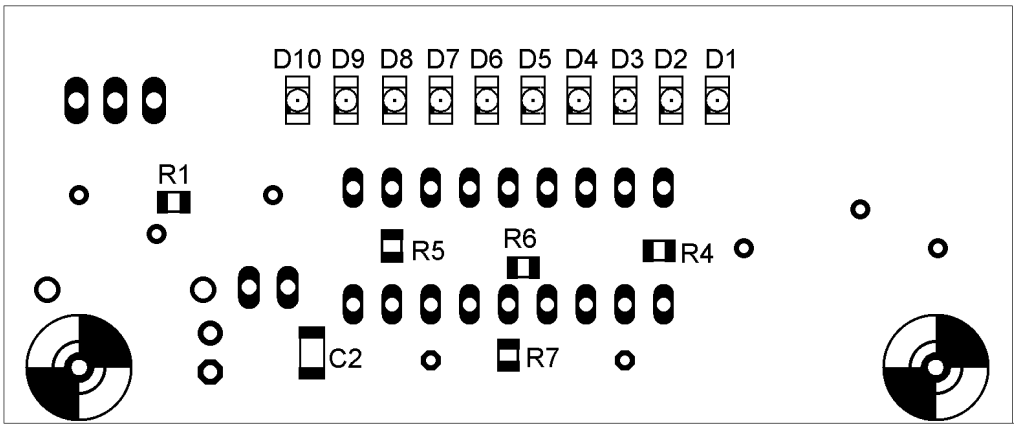


Bild A.2.: Bestückungsplan TOP

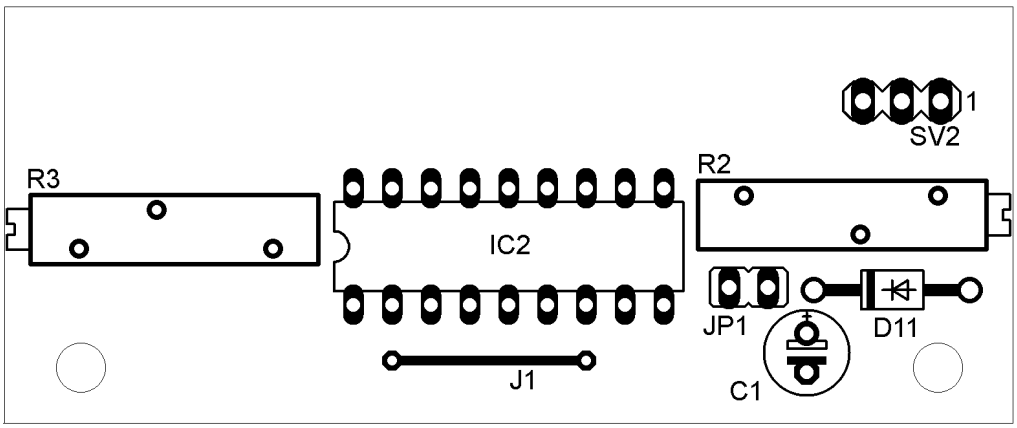


Bild A.3.: Bestückungsplan Bottom

A.2. Beineinheit

Bauteil	Wert	Bezeichnung
B1		Brückengleichrichter W10G
C1	100n	Kondensator SMD 0805
C2	100n	Kondensator SMD 0805
C3	2.2μ	Kondensator SMD 1206
C4	100n	Kondensator SMD 0805
C5	100n	Kondensator SMD 0805
C6	100n	Kondensator SMD 0805
C7	100n	Kondensator SMD 0805
C8	2.2μ	Kondensator SMD 1206
C9	2.2μ	Kondensator SMD 1206
C10	100n	Kondensator SMD 0805
C11	2.2μ	Kondensator SMD 1206
C12	100n	Kondensator SMD 0805
C13	100n	Kondensator SMD 0805
IC1		LM324D
IC2		LM324D
IC3		LM324D
IC4		L298
R1	180k	Widerstand SMD 0805
R2	180k	Widerstand SMD 0805
R3	180k	Widerstand SMD 0805
R4	180k	Widerstand SMD 0805
R5	47k	Widerstand SMD 0805
R6	47k	Widerstand SMD 0805
R7	12k	Widerstand SMD 0805
R8	120k	Widerstand SMD 0805
R9	120k	Widerstand SMD 0805
R10	50k	Trimmer 15-gängig 3006P
R11	180k	Widerstand SMD 0805
R12	180k	Widerstand SMD 0805
R13	180k	Widerstand SMD 0805
R14	180k	Widerstand SMD 0805
R15	47k	Widerstand SMD 0805
R16	47k	Widerstand SMD 0805
R17	12k	Widerstand SMD 0805
R18	120k	Widerstand SMD 0805
R19	120k	Widerstand SMD 0805
R20	180k	Widerstand SMD 0805
R21	180k	Widerstand SMD 0805
R22	180k	Widerstand SMD 0805
R23	180k	Widerstand SMD 0805
R24	47k	Widerstand SMD 0805
R25	47k	Widerstand SMD 0805

R26	12k	Widerstand SMD 0805
R27	120k	Widerstand SMD 0805
R28	120k	Widerstand SMD 0805
R29	50k	Trimmer 15-gängig 3006P
R30	50k	Trimmer 15-gängig 3006P
SV1	20-polig	Steckerleiste 1-reihig Raster 2.54mm
SV2	6-polig	Stecker 2 x 3 Reihen Raster 2.54mm
SV3	6-polig	Stecker 2 x 3 Reihen Raster 2.54mm
SV4	6-polig	Stecker 2 x 3 Reihen Raster 2.54mm
SV5	6-polig	Stecker 2 x 3 Reihen Raster 2.54mm
X3	3-polig	Stecker SL 5.08/3/90B

Tabelle A.2.: Bauteilliste

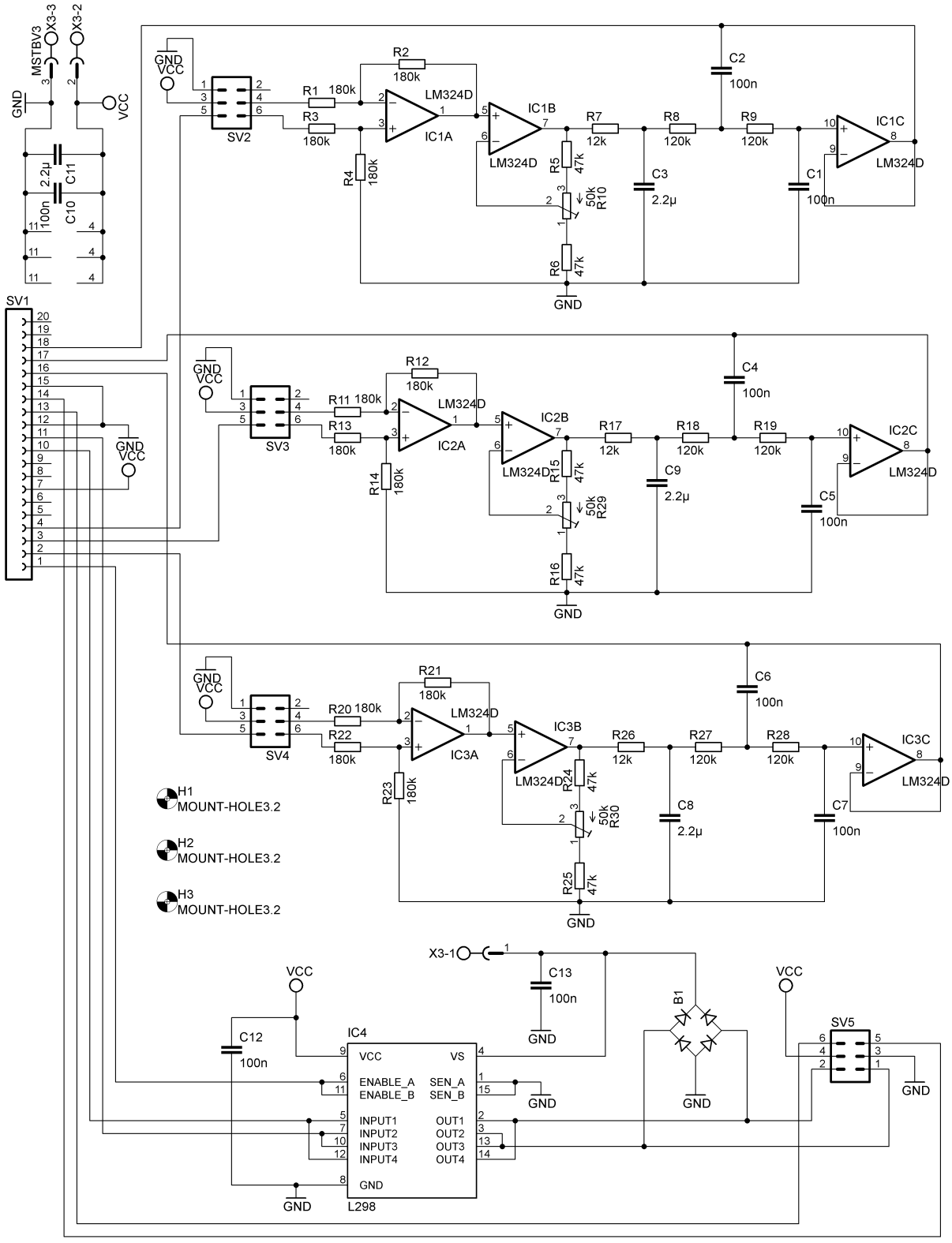


Bild A.4.: Schaltplan Beineinheit

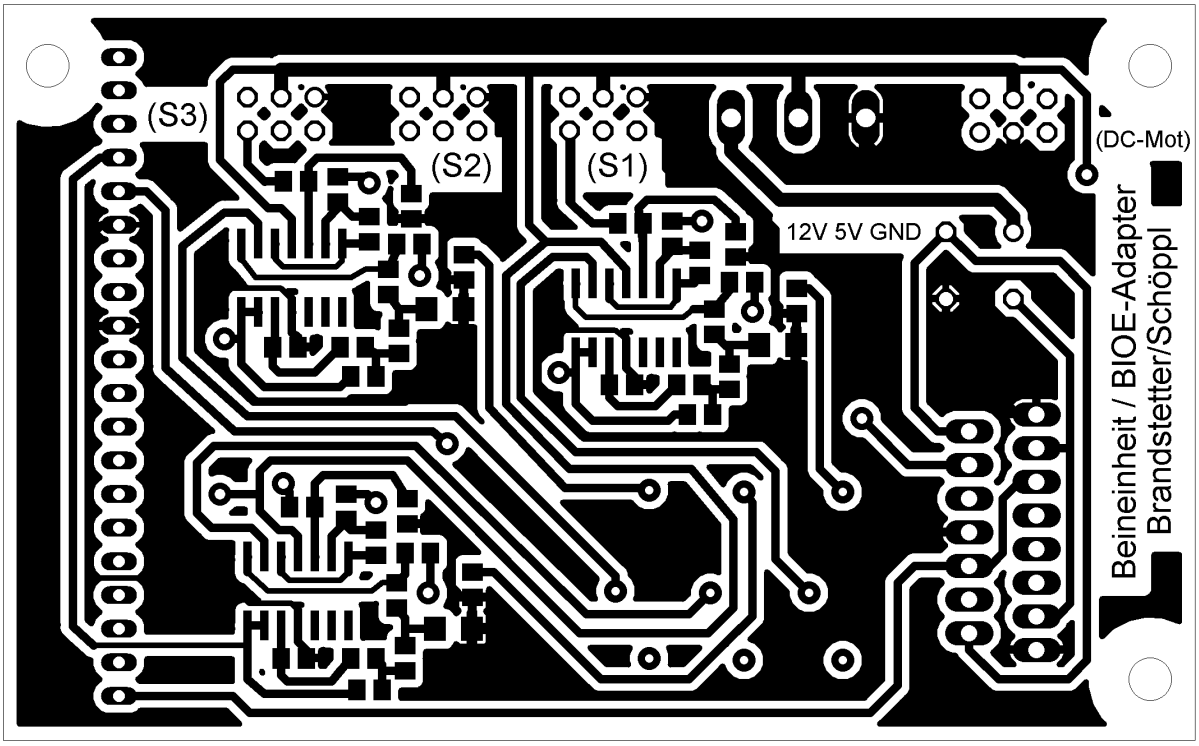


Bild A.5.: Layout Beineinheit TOP

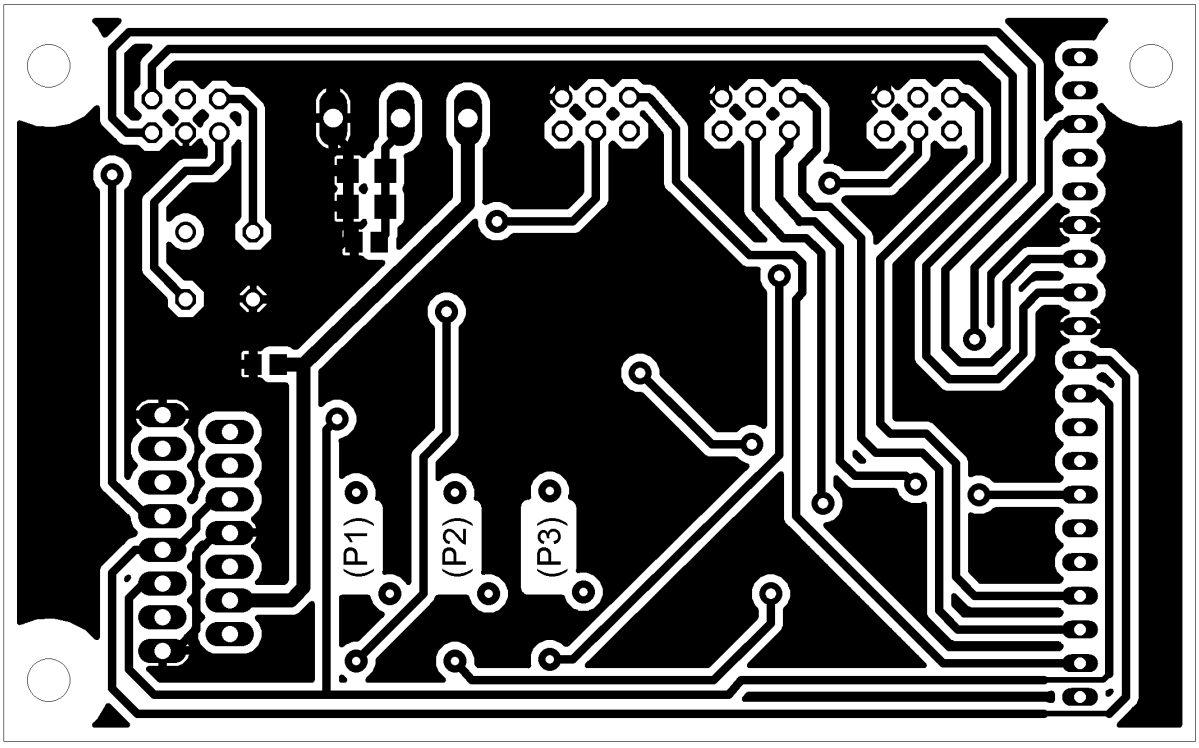


Bild A.6.: Layout Beineinheit BOTTOM

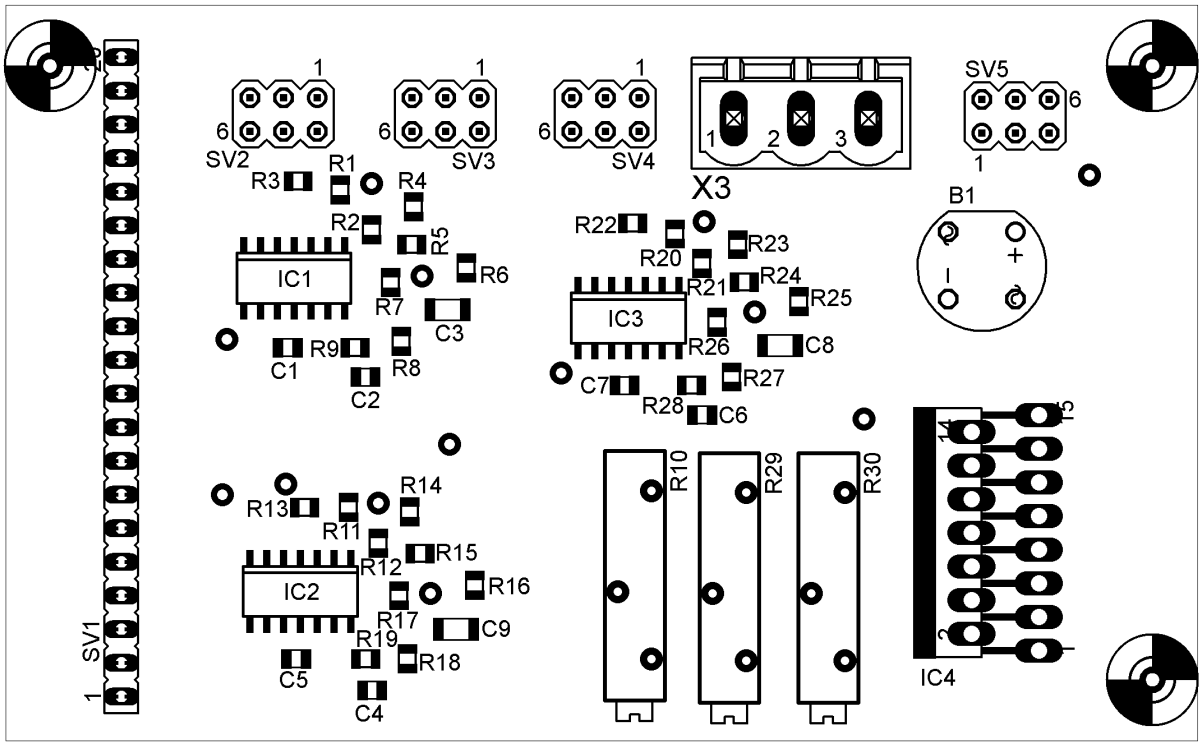


Bild A.7.: Bestückungsplan Beineinheit TOP

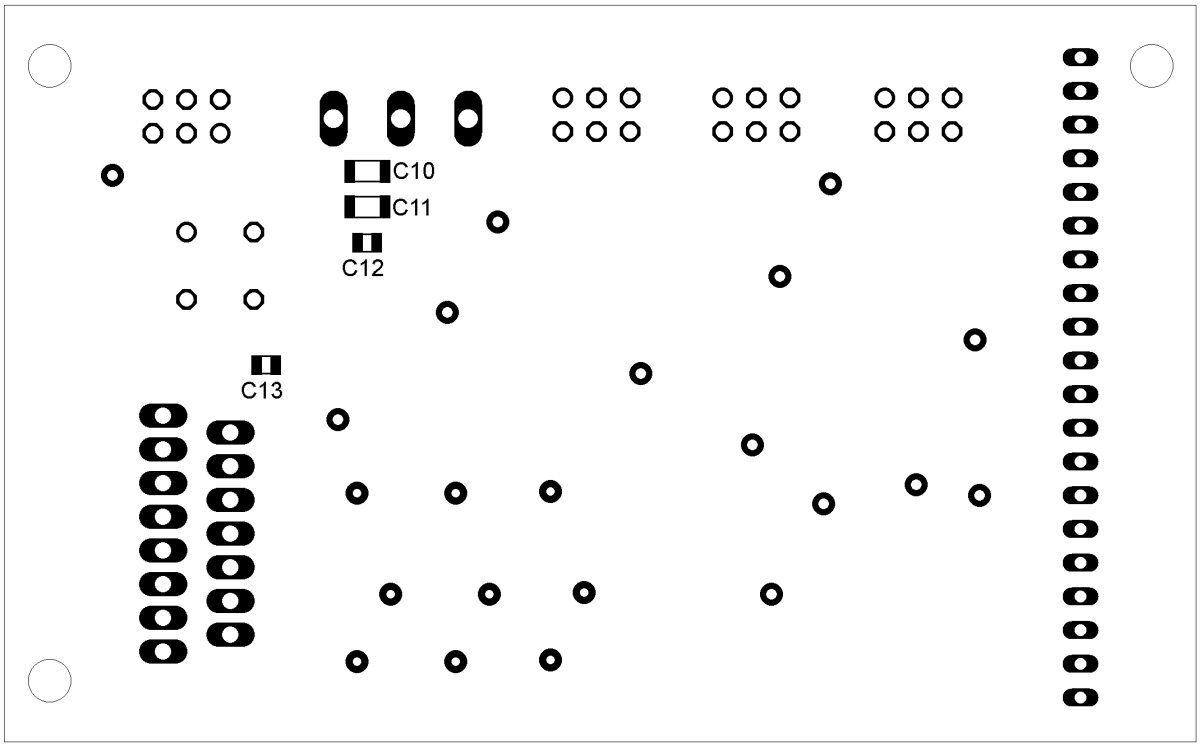


Bild A.8.: Bestückungsplan Beineinheit BOTTOM

Bauteil	Wert	Bezeichnung
C1	220μ	Elektrolytkondensator
C2	220μ	Elektrolytkondensator
C3	220μ	Elektrolytkondensator
C4	470μ	Elektrolytkondensator
C5	470μ	Elektrolytkondensator
C6	470μ	Elektrolytkondensator
C7	470μ	Elektrolytkondensator
C8	470μ	Elektrolytkondensator
D1		Diode 1N5822
D2		Diode 1N5822
D3		Diode 1N5822
D4		Diode 1N5822
D5		Diode SMD 1N4148
IC1		Spannungswandler LM2596-5.0
IC2		Spannungswandler LM2596-5.0
IC3		Spannungswandler LM2596-5.0
IC4		Spannungswandler LM2596-12.0
K1		Relay SMD IM06D
K2		Relay SMD IM06D
L1	47μ	Induktivität
L2	47μ	Induktivität
L3	47μ	Induktivität
L4	100μ	Induktivität
SV1	3-polig	Stecker 1-reihig Raster 2.54mm
SV3	3-polig	Stecker 1-reihig Raster 2.54mm
X1	2-polig	Stecker SL 5.08/2/90B
X2	2-polig	Stecker SL 5.08/2/90B
X3	3-polig	Stecker SL 5.08/3/90B
X5	14-polig	Wannenstecker

Tabelle A.3.: Bauteilliste

A.3. Versorgungsplatine

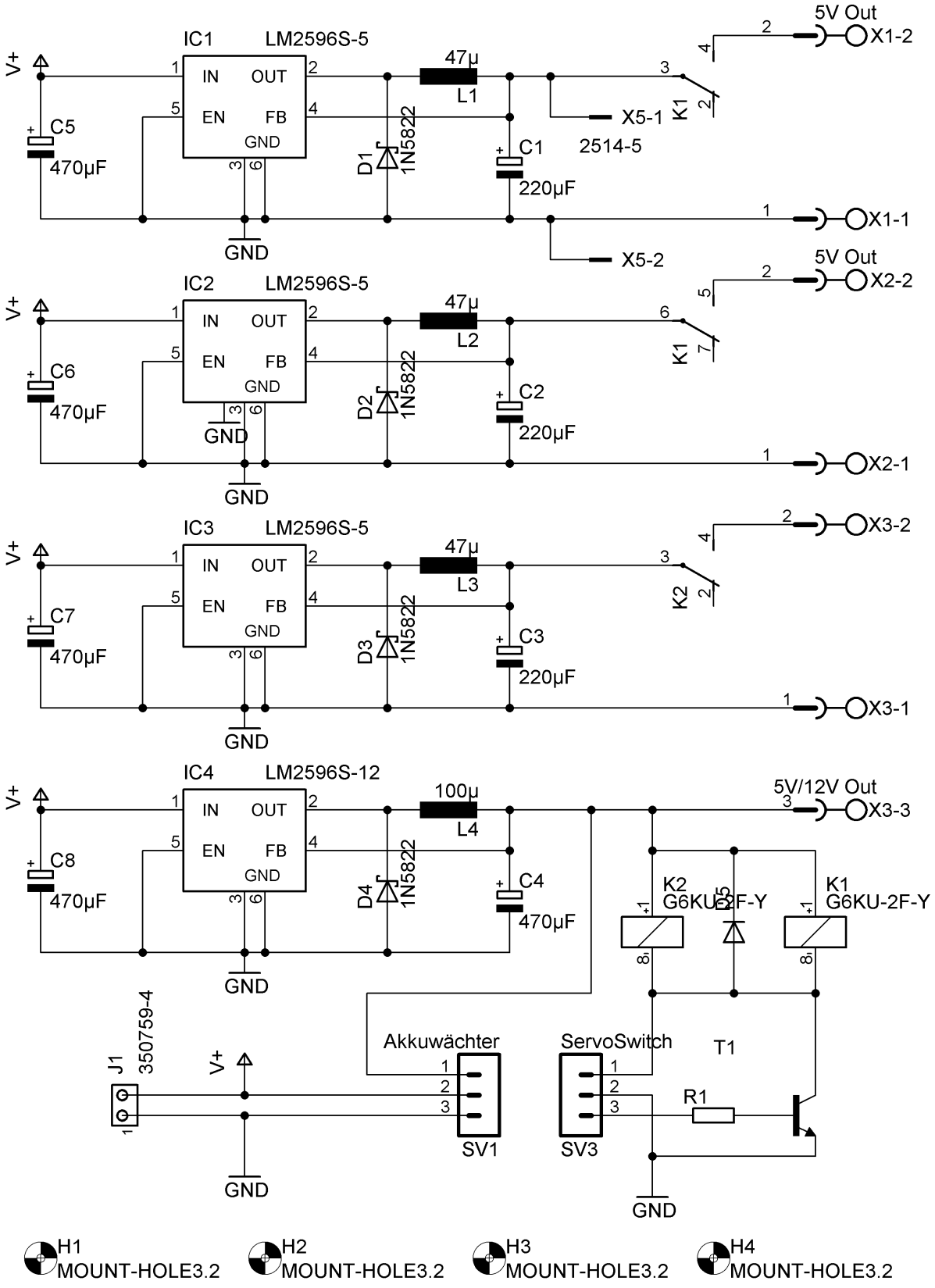


Bild A.9.: Schaltplan Versorgungsplatine

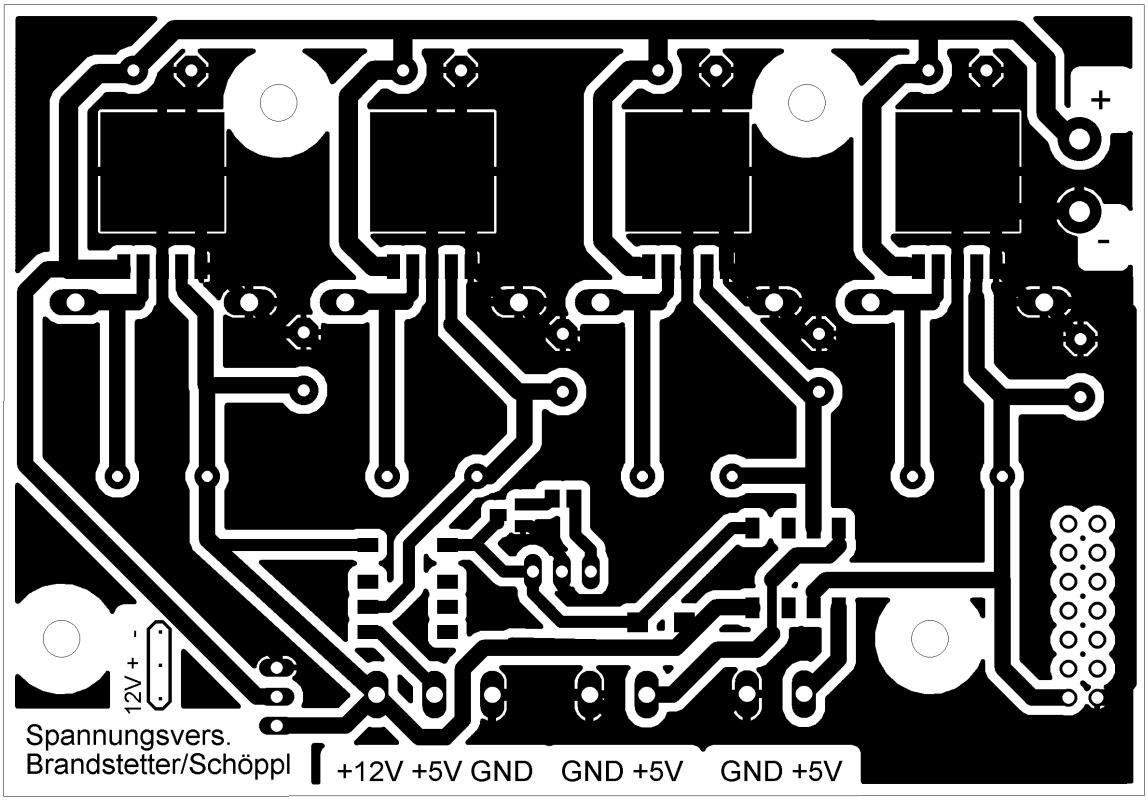


Bild A.10.: Layout Versorgungsplatine

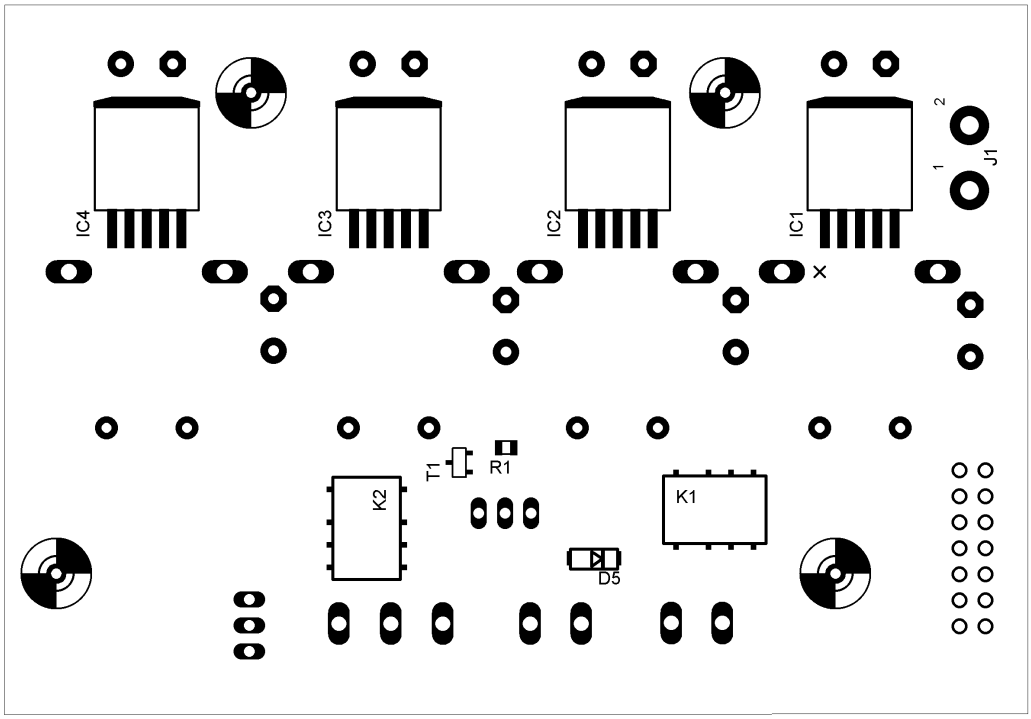


Bild A.11.: Bestückungsplan Versorgungsplatine TOP

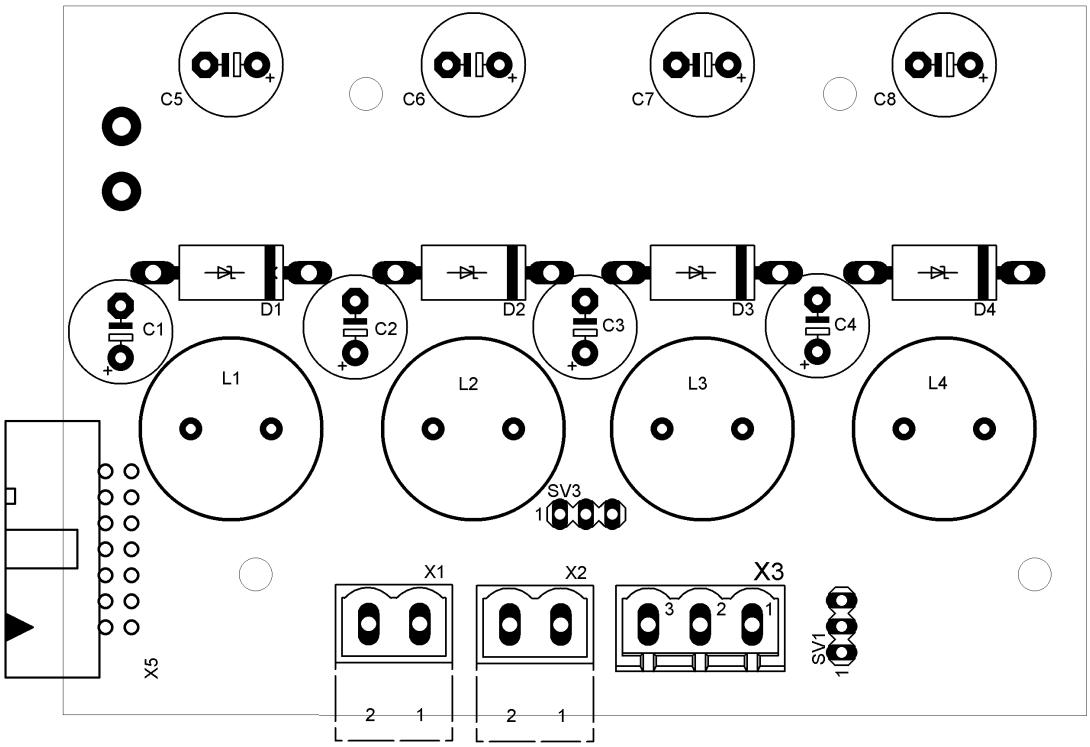


Bild A.12.: Bestückungsplan Versorgungsplatine BOTTOM

Literatur

[AW08] ANDREAS WEISSL, ANDREAS HOFER, BERNHARD SCHAUBERGER: *Konstruktion, Fertigung und Regelung eines DOF5-Roboterarms*. Johannes Kepler Universität Linz - Institut für Robotik, September 2008.

[Bar91] BARTSCH, DR.-ING HANS-JOCHEN: *Taschenbuch mathematischer Formeln*. Fachbuchverlag Leipzig, 14. Auflage, 1991.

[BK88] BRIAN KERNIGHAN, DENNIS RITCHIE: *The C Programming Language*. Prentice Hall Software Series. Prentice Hall, Inc., 2. Auflage, 1988.

[Bre05] BREMER, HARTMUT: *Vorlesungsskript Robotik I - 6. Auflage*. Johannes Kepler Universität Linz - Institut für Robotik, Juli 2005.

[Gat09] GATTRINGER, HUBERT: *Vorlesungsskriptum Steuerung und Regelung von Robotersystemen*. Johannes Kepler Universität Linz - Institut für Robotik, 2009.

[Kas06] KASTNER, MICHAEL: *Modellbildung und Steuerung einer sechsbeinigen Laufmaschine mit Fahrantrieb*. Diplomarbeit, Johannes Kepler Universität Linz - Institut für Robotik, März 2006.

[Mös03] MÖSSENBÖCK, HANSPETER: *Spechen Sie Java*. dpunkt.verlag Gmbh, 2. Auflage, 2003.

[Mös08] MÖSSENBÖCK, HANSPETER: *Vorlesungsskript Softwareentwicklung 2*. Johannes Kepler Universität Linz - Institut für Systemsoftware, 2008.

[Roh08] ROHRHOFER, ANDREAS: *Trajektorienfolgeregelung eines Segways*. Diplomarbeit, Johannes Kepler Universität Linz - Institut für Robotik, Juli 2008.

[Wei08] WEICHINGER, KLAUS: *BIOE Manual*. Johannes Kepler Universität Linz - Institut für Regelungstechnik und Prozessautomatisierung, Oktober 2008.