Joseph L. Jones

mit Beiträgen von
Anita M. Flynn, Fred Martin und Randy Sargent

Bauanleitung für mobile Roboter

mit Interactive C Handbuch

Inhalt

Kapitel 1
Einleitung

1.1 Die Roboter kommen

Schritt für Schritt schleichen sich Roboter in unser Leben.

Roboterteilsysteme sind heutzutage fast überall anzutreffen. Nahezu jedes heute hergestellte Auto hat ein Robotergehirn, d. h. hochintegrierte Mikrocontroller. Diese winzigen Computer vereinen auf einem winzigen Chip A/D-Wandler, Zeitgeber und andere Funktionen. In Fahrzeugen dienen Mikrocontroller zur Überwachung der Motorsteuerung und zur Optimierung der Leistung.

Augen und Ohren von Robotern, d. h. Photozellen, Infrarotdetektoren und andere Sensoren, suchen unser Zuhause nach Eindringlingen ab. Andere Apparate warten geduldig auf Hände, die unter einem Hahn in der Toilette einer Flugzeughalle erscheinen – sobald sie erkannt sind, fließt Wasser.

Die Muskeln der Roboter, d. h. Servomotoren und Elektromagnete, füllen Gläser automatisch mit Wasser oder Limonade und leiten Magnetbänder durch das interne Labyrinth von Videokameras.

Für sich genommen sind alle diese Roboterbauteile nichts Ungewöhnliches. Sie werden sogar als selbstverständlich betrachtet. Ungewöhnlich ist, Mikrocontroller, Sensoren und Motoren in einen funktionstüchtigen autonomen Roboter zusammengefügt zu sehen. Im alltäglichen Leben stoßen wir noch nicht sehr häufig auf Maschinen, die sich selbständig bewegen und ohne fremde Hilfe Aufgaben durchführen, die wir als bedeutend ansehen.

Dies ändert sich zunehmend. Die neuesten Errungenschaften auf dem Gebiet der Roboterprogrammierung, kontinuierliche Fortschritte in der Mikroelektronik und die Miniaturisierung von Sensoren haben die Robotertechnologie an einen kritischen Punkt gebracht. Für uns beginnt ein Zeitalter, in dem Roboter nach und nach die Laboratorien verlassen und in unseren Büros und Wohnungen Einzug halten. Bald können Sie Koffer kaufen, die Ihnen ganz von selbst durch die Abfertigungshalle in Ihr Flugzeug folgen – und zuhause kümmert sich ein intelligenter Staubsauger in Ihrer Abwesenheit um Ihre Fußböden, ohne daß Sie sich anstrengen müssen.

1.2 Sinkende Kosten

Bevor diese Wunder in die Tat umgesetzt werden können, bleibt noch eine ganze Menge Arbeit zu tun. Aufgrund der oben genannten Fortschritte sind die Kosten für Experimente im Bereich des Roboterbaus sowie für Forschungsarbeiten im Bereich der Robotik so niedrig wie nie zuvor. Noch vor ein paar Jahren konnten es sich, angesichts anfänglicher Fixkosten in Höhe von rund DM 100 000, nur einige wenige finanziell gutgestellte Forschunginstitute leisten, Experimente mit Robotern durchzuführen. Inzwischen können ernsthafte Forschungs- und Schulungsprogramme bereits mit einem Startkapital von weniger als DM 5000 durchgeführt werden. Die Tür zu diesem wichtigen und faszinierenden Gebiet wurde weit aufgestoßen.

Der Rug Warrior wurde zusammen mit dem Begleitbuch *Mobile Roboter: Von der Idee zur Implementierung* (Originalausgabe bei A. K. Peters, Ltd., 1993, ISBN 1-56881-011-3), vom Forschungsteam des KI-Labors am Massachusetts Institute of Technology entwickelt. Sowohl durch den Roboter als auch das Buch soll die neuentwickelte Technologie aus dem KI-Labor des MIT und anderen Stellen einem breiten Publikum nahegebracht werden. Wir haben den Rug Warrior deshalb so preiswert wie möglich gestaltet, ohne jedoch auf technische Neuerungen zu verzichten.

1.3 Rug Warrior

Diese Anleitung beschreibt den Zusammenbau des Elektronik- und des Mechanik-Teils des Rug Warrior-Bausatzes.

Der Bausatz (Elektronik und Mechanik) enthält einen voll integrierten Roboter, der sofort im Schulungs- oder Forschungsbereich eingesetzt werden kann.

1.4 Zusammenfassung

Die Logik- und Schnittstellenschaltung des Elektronik-Bausatzes sind vormontiert. Dadurch können Sie unverzüglich mit der Programmierung beginnen. Lesen Sie in Kapitel 2, wie Sie die Rug Warrior-Platine anschließen und mit Hilfe Ihres Computers programmieren. Abbildung 1.1 zeigt das Platinenlayout für die Bauteile und Fassungen.

Die Sensorschaltungen des Rug Warrior müssen vom Anwender selbst aufgebaut werden. Kapitel 3 führt Sie Schritt für Schritt durch den Konstruktionsvorgang. Mit den im Elektronik-Bausatz mitgelieferten Selbsttest-Routinen können Sie die Funktionalität der einzelnen Teilsysteme während der Konstruktionsphase überprüfen.

Als Programmierumgebung für den Rug Warrior wird Interactive C (IC), eine leistungsfähige und einfach zu erlernende Programmiersprache empfohlen. IC ist im Bausatz enthalten. Das Erstellen des Programms und die Fehlersuche wird durch

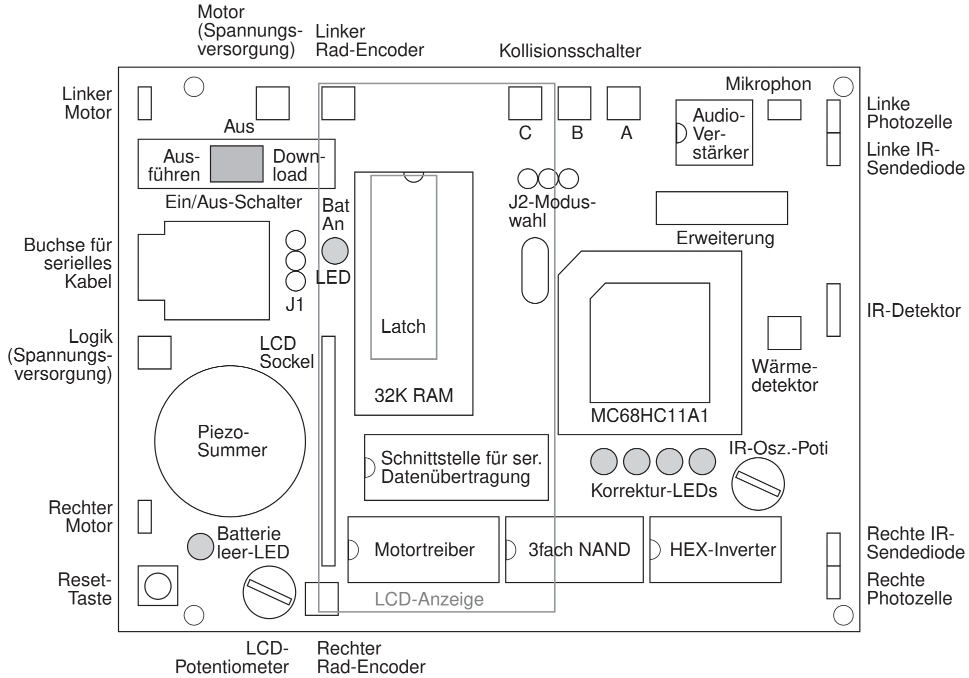

Abbildung 1.1: Layout der Rug Warrior-Platine.

die Interaktivität und das in IC vorhandene Multitasking stark vereinfacht. Sowohl das Selbsttestprogramm als auch das Demonstrationsprogramm sind in IC geschrieben. Natürlich sind Sie nicht gezwungen, Ihren Roboter mit diesem System zu programmieren. Der Mikrocontroller MC68HC11 von Motorola läßt sich mit jedem kompatiblen System programmieren.

Auf den Zusammenbau des Mechanik-Bausatzes und die Integration des Elektronik-Bausatzes geht Kapitel 4 ein.

Eine der faszinierensten Aspekte des Roboterbaus ist die Konstruktion von Sensor- und Aktuatorsystemen. Für den Einbau dieser Systeme hat der Rug Warrior integrierte Erweiterungsfunktionen. In Kapitel 5 erfahren Sie mehr über diese Option.

Kapitel 6 enthält eine Reihe von Vorschlägen zur Diagnose und Korrektur von Fehlern, die manchmal während oder nach der Konstruktionsphase auftreten.

Das Interactive C-Handbuch (Kapitel 7) beschreibt das IC-Programmiersystem. Kapitel A geht auf die mitgelieferten Bibliotheksroutinen ein. In Kapitel B wird die serielle Schnittstelle des Rug Warrior beschrieben, und ausgewählte technische Daten werden aufgelistet.

Kapitel 2

Start frei für den Roboterbau

2.1 Der Elektronik-Bausatz

Bevor Sie den Rest des Sensorschaltkreises installieren oder Ihre eigenen Konstruktionen hinzufügen, sollten Sie die Funktionalität Ihrer Platine überprüfen und sich mit den Selbsttest-Funktionen vertraut machen. Führen Sie hierzu folgende Schritte durch:

1. Installieren Sie die Programmierumgebung Interactive C (IC) auf Ihrem Computer.
2. Schließen Sie die Batterie an die Platine an.
3. Schließen Sie den Rug Warrior mit einem seriellen Kabel an den Computer an.
4. Initialisieren Sie die Platine durch Laden des P-Code (des Betriebssystems) in den Rug Warrior.
5. Stellen Sie eine Verbindung vom Rug Warrior zum Computer durch Ausführen von IC her.

Die einzelnen Schritte werden in diesem Kapitel näher erläutert.

2.2 Installation der Software

Als Programmierumgebung für den Rug Warrior empfehlen wir Interactive C. Sie befindet sich auf der Programmdiskette, die mit dem Bausatz mitgeliefert wird. Zur Installation von IC befolgen Sie die untenstehenden Anweisungen. Alternativ können Sie auch die Anweisungen in der Datei `README.IC` auf der Programmdiskette lesen und ausführen.

2.2.1 Macintosh-Anweisungen

Erstellen Sie auf Ihrer Festplatte einen neuen Ordner namens »IC« und ziehen Sie den Inhalt der Programmdiskette in diesen Ordner.

2.2.2 DOS-Anweisungen

Kopieren Sie die Dateien von der Programmdiskette mit `xcopy` auf Ihre Festplatte in ein Verzeichnis namens `ic`. Angenommen, Ihre Programmdiskette befindet sich in Laufwerk `a:`, dann lautet der Dialog folgendermaßen:

```
C:>xcopy a:*.* c:
Does IC specify a file name
or directory name on the target
(F = file, D = directory)?d
```

Die Kursivschrift kennzeichnet die durch den Benutzer einzugebenden Befehle.

2.2.3 Windows-Anweisungen

Siehe Anweisungen in der Datei `README.IC` auf der Programmdiskette.

2.3 Display- und Spannungsversorgungsanschluß

Nehmen Sie die Platine aus ihrem antistatischen Gehäuse. Stecken Sie die LCD-Anzeige auf den hervorstehenden 14-Pin-Sockel auf der Rug Warrior Platine. Bei manchen Ausführungen wurde dieser Schritt bereits vom Werk durchgeführt. Sehen Sie sich hierzu Abbildung 1.1 an. Es kann vorkommen, daß dabei eine Ecke der LCD-Anzeige auf dem Gehäuse des 8 MHz Quarzes aufliegt (dem silbernen rechteckigen Bauteil in der linken oberen Ecke des Mikrocontroller MC68HC11). Kleben Sie in diesem Fall einen Streifen Isolierband auf die Oberseite des Quarzes, um einen Kurzschluß zwischen den Leiterbahnen auf der Unterseite des Bildschirms und dem Quarzgehäuse zu verhindern.

Legen Sie vier AA Alkali-Batterien in das vorhandene Batteriefach. Achten Sie dabei darauf, daß sich der Ein/Aus-Schalter auf der Platine in der mittleren AUS-Stellung befindet. Schließen Sie danach das Kabel des Batteriefaches an den gepolten 2-Pin Stecker auf der linken Seite der Rug Warrior Platine an. Vergewissern Sie sich dabei, daß kein leitendes Material (Drähte, Lötzinn usw.) auf der Platine freiliegt. Der Anschluß ist mit »BAT PWR« gekennzeichnet.

2.4 Serieller Anschluß

Das mitgelieferte Kabel hat eine DB-25 Anschlußbuchse auf der einen Seite und einen Western-Telefonstecker (RJ-11) auf der anderen. Verwenden Sie dieses Kabel, um die Verbindung zwischen Ihrem Computer und der Rug Warrior-Platine herzustellen. Schauen Sie sich hierzu auch Abbildung 2.1 an. In die Buchse DB-25 passen die meisten seriellen Stecker von IBM-PCs und kompatiblen Computern. Wenn Ihr Computer einen 9-Pin-Stecker hat, müssen Sie entweder ein Modemkabel

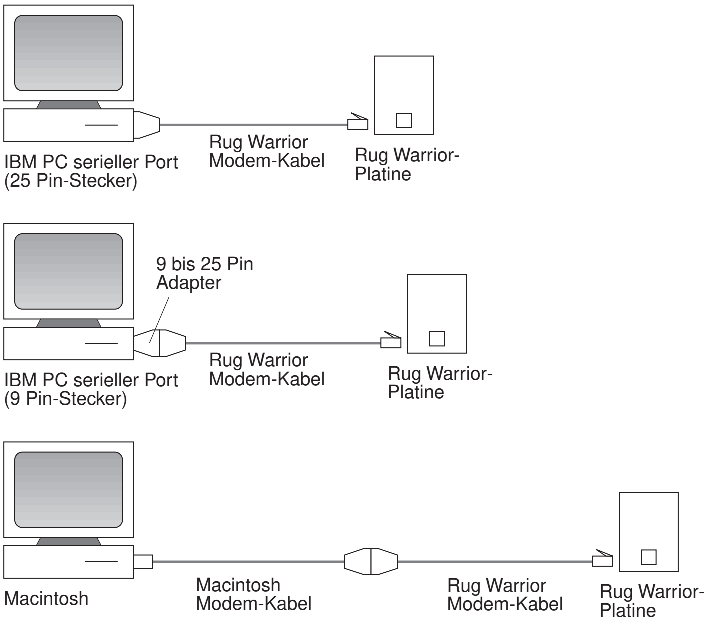

Abbildung 2.1: Anschlußmöglichkeiten zwischen dem Host-Computer und der Rug Warrior-Platine.

oder einen 9- bis 25-Pin-Adapter verwenden, um ihn mit dem Kabel des Elektronik-Bausatzes zu verbinden. Wenn Sie einen Macintosh-Rechner haben, verwenden Sie ein Standardmodemkabel (DIN-8 to DB-25).

Einige Computer sind mit Telefonbuchsen ausgestattet, die mit internen Modems verbunden sind. Schließen Sie den Rug Warrior jedoch *nicht* über diese Buchse an. Modems erzeugen modulierte Töne, die für Telefonleitungen ausgelegt sind, nicht aber für die logischen Signale, die der Rug Warrior benötigt.

Sehen Sie in Ihrem Computerhandbuch nach, wie die serielle Schnittstelle konfiguriert werden muß. Die Daten von Notebook-Computern werden z. B. häufig direkt auf ein internes Modem geleitet. In diesem Falle müssen Sie die Standardeinstellung Ihres Computers so ändern, daß das »externe Modem« angewählt ist. Wenn normalerweise ein anderes Gerät, wie beispielsweise eine Maus oder ein LAN, die Leitung belegt, müssen Sie die zugehörigen Softwaretreiber wahrscheinlich deaktivieren, da das IC-System ansonsten vermutlich keinen Zugriff auf die serielle Schnittstelle erhält.

2.5 Laden des P-Code

Der P-Code ist so etwas wie das Betriebssystem des Roboters. Mit seiner Hilfe können in IC geschriebene Programme ausgeführt werden. Normalerweise liegt der P-Code resistent im statischen RAM auf der Platine und ist batteriegepuffert. Wenn Sie die Batterien aus dem Roboter herausnehmen, wird jedoch auch der P-Code gelöscht. Es kann vorkommen, daß der P-Code während der Programmentwicklung oder der Montage des Roboters teilweise oder ganz zerstört wird. In diesem Fall muß er erneut in das RAM geladen werden.

Schließen Sie für den Ladevorgang zunächst das Netzkabel und das serielle Kabel wie oben beschrieben an, und legen Sie den EIN/AUS-Schalter nach rechts in die Stellung »DL« (Download). Die grüne »BAT ON«-LED sollte leuchten. Drücken Sie anschließend die RESET-Taste. Solange diese Taste gedrückt ist, sollte die rote »Batterie leer«-LED aufleuchten. Sobald Sie die RESET-Taste loslassen, sollte diese Leuchte erlöschen.

Wenn Sie den P-Code von einem *Macintosh-Rechner* aus installieren, suchen Sie zunächst die Datei namens `Init␣Board␣RW␣7.2␣modem`, und doppelklicken Sie auf das Icon, oder schließen Sie den Rug Warrior an den Druckerport an, und doppelklicken Sie auf die Datei `Init␣Board␣RW␣7.2␣printer`.

Beim Laden von einem *DOS-Rechner* geben Sie folgendes ein:

```
dl pcoderwl.s19
```

Die Optionen zum Laden von standardmäßig konfigurierten Computern ist in der Datei `README.IC` beschrieben.

Das Download-Programm auf Ihrem Rechner informiert Sie detailliert über die einzelnen Schritte. In der ersten Phase des Download wird ein spezieller 256 Byte langer Programmlader (»bootstrap loader«) mit 1200 Baud an den Mikrocontroller gesandt. Eine gepunktete oder gestrichelte Linie zeigt an, wieviele der Zeichen bereits geladen wurden. Dieser Programmlader hat die Aufgabe, dem Mikrocontroller mitzuteilen, wie er den darauffolgenden P-Code lädt. Nachdem das letzte Byte des Programmladers übertragen wurde, beginnt der Mikrocontroller automatisch mit der Ausführung des Programmladers.

Die Übertragungsgeschwindigkeit steigt nun auf 9600 Baud, und das Download-Programm versucht, mit dem vom Mikrocontroller ausgeführten Programmlader zu kommunizieren. Wenn die Installation des Programmladers erfolgreich war, können der Host-Computer und die Rug Warrior Platine synchronisiert werden, und das Download-Programm kann mit der Übertragung des eigentlichen P-Codes beginnen. Sobald die Übertragung beginnt, erscheinen weitere Punkte auf dem Bildschirm. Während des Ladevorgangs hat der Zustand der vier LEDs unterhalb des MC68HC11-Mikrocontrollers (an oder aus) keine Bedeutung.

Wenn der gesamte P-Code geladen wurde, legen Sie den EIN/AUS-Schalter auf die ganz linke Stellung »RUN« (Ausführen), und drücken Sie anschließend die RESET-Taste. Der Piezo-Summer gibt einen kurzen Ton aus, und auf der LCD-Anzeige erscheint die Meldung »Interactive C«. Eventuell müssen Sie die Einstellung des LCD-Potentiometer verändern, um einen besseren Kontrast zu erhalten (siehe Abbildung 1.1).

2.6 Kommunikation mit dem Host

Bei einem *Macintosh-Rechner* starten Sie das IC-Programm durch Doppelklicken auf die Anwendung `IC␣RW␣2.xxx␣modem` bzw. `IC␣RW␣2.xxx␣printer`, wenn Sie den Druckerport verwenden. Das `xxx` steht hierbei für die dreistellige Zahl der IC-Version auf Ihrer Programmdiskette.

Bei einem *DOS-Rechner* starten Sie das IC-System durch Eingabe von

```
ic
```

Der Computer stellt daraufhin eine Verbindung mit der Rug Warrior-Platine her und lädt die Standardbibliothek. Danach können Sie C-Anweisungen über die Tastatur eingeben und Programme in den Mikrocontroller des Roboters laden. Die Eingabe

```
beep();
```

führt beispielsweise dazu, daß der Piezo-Summer einen kurzen Ton ausgibt. Die Eingabe

```
2 + 2;
```

führt etwa zu folgender Meldung:

```
Downloaded 7 bytes (addresses C200-C206)
Returned <int> 4
```

2.7 Testen der Platine

Als erstes sollten Sie das Programm `rw-test.lis` laden. Geben Sie nach dem IC-Prompt einfach ein:

```
load rw-test.lis
```

Wenn der Ladevorgang beendet ist, drücken Sie die RESET-Taste, um nacheinander die Tests für die einzelnen Teilsysteme auszuwählen. Abschnitt 2.10 beschreibt ausführlich die Funtionen der einzelnen Tests. Zu Anfang liefern jedoch nur der LCD-Test, der LED-Test und der Summer-Test sinnvolle Ergebnisse.

2.8 Die Rug Warrior-Software

Neben der Programmierumgebung IC enthält die Programmdiskette den folgenden Rug Warrior-spezifischen Programmcode:

rw-test.lis
Datei zum Laden der Selbsttestfunktionen

selftest.c
Routinen zum Testen aller Sensoren und Aktuatoren des Rug Warrior

cof.c
Läßt den Rug Warrior eine Melodie spielen, die den Piezosummer testet

shaft.lis
Datei zum Laden der Rad-Encoder-Hilfsprogramme

shaft.c
Benutzerschnittstelle für den Rad-Encoder

speed.asm
Assemblerdatei, die den Impulszähler für den linken Rad-Encoder implementiert

speed.icb
Assemblierte Version von `speed.asm`, die vom IC geladen werden kann (`.asm`-Files können nicht geladen werden)

regdefs.c
Definitionen für die speicherorientierten Register des MC68HC11

lib_rw11.c
Standardschnittstelle für die Sensoren und Aktuatoren des Rug Warrior

Die mit dem Platinen-Bausatz mitgelieferte Software ist kostenlos. Sie können dieses Programm beliebig kopieren und verteilen, wobei jedoch die Verbreitung einigen Einschränkungen unterliegt (siehe `README.IC`). Die Software wird wie vorliegend bereitgestellt. Gewähr oder Garantie wird nicht übernommen.

2.9 Routinen in Assemblersprache

Das aktuelle Release der Rug Warrior-Software umfaßt nicht den speziellen MC68-HC11-Assembler (AS11), der das Schreiben von Assemblersprachroutinen und ihre Einbindung im Programm erleichtert. Für die Mehrheit der Anwender, die ausschließlich in IC programmieren wollen, ist dies eh unwichtig. Eine Beschreibung des Formats verarbeiteter Assemblerdateien (`.icb`-Dateien) finden Sie im IC-Handbuch (Kapitel 7). Sollten Sie einer jener überzeugten Assemblerprogrammierer sein, können Sie durch Entschlüsselung dieses Formats `.icb`-Dateien von Hand erstellen.

Wenn Sie einen Zugang zum Internet haben, sollten Sie hin und wieder unter der Adresse `cherupakha.media.mit.edu` nachsehen. Dort finden Sie die neueste Version von IC und Software für den Rug Warrior und ähnliche Roboterplatinen.

2.10 Selbsttest

Der Rug Warrior wird mit einem Satz Routinen ausgeliefert, die Sensorwerte anzeigen und Aktuatoren aktivieren. Mit diesen Routinen können Sie während der Montage und der Anpassung Ihres Roboters die Funktion aller Teilsysteme überprüfen.

Zum Laden der Selbsttestroutinen starten Sie IC und laden Sie `rw-test.lis`. Diese Datei lädt automatisch folgende Dateien: `speed.icb`, `shaft.c`, `regdefs.c`, `cof.c` und `selftest.c`.

Sobald `rw-test.lis` geladen ist, können Sie den gewünschten Test durch Drücken der RESET-Taste (solange bis der gewünschte Test erscheint) auswählen. Sie können einen bestimmten Test aber auch direkt über die Tastatur aufrufen. Geben Sie dazu zunächst `kill_all` ein, um alle derzeit laufenden Prozesse zu beenden. Anschließend geben Sie den Namen des gewünschten Tests ein (siehe nachfolgende Beschreibung). Alle Tests werden ohne Unterbrechung ausgeführt.

`lcd_test()`
Zeigt nacheinander alle durch die LCD-Anzeige definierten Zeichen an. Nicht alle Zeichencodes sind belegt. Bei einem nicht definierten Zeichen bleibt der Bildschirm leer.

`led_test()`
Routine für die vier Kontroll-LEDs. Die Einer in den niedrigsten vier Bits des Byte, das auf dem LCD angezeigt wird, geben an, welche LEDs an sein sollen.

`piezo_test()`
Erzeugt eine Melodie über den Piezo-Summer und zeigt dabei die Frequenz jedes Tons an.

`photo_test()`
Zeigt die von der linken und rechten Photozelle gemessene Lichtstärke an. Je niedriger die Zahl ist, desto heller ist das Licht. Die obere Zeile auf dem Bildschirm gibt die Differenz zwischen der linken und rechten Lichtintensität an. Der Pfeil zeigt auf den Sensor, der dem helleren Licht ausgesetzt ist.

`bumper_test()`
Gibt an, welche der Kollisionsschalter geschlossen sind. Der Rug Warrior erkennt die Schalterstellung, indem er den vom Spannungsaddierer ausgegebenen Analogwert auswertet. Die Spannung, die als digitaler Wert zwischen 0 und 255 dargestellt wird, ist auf der zweiten Zeile auf der LCD-Anzeige abzulesen.

`mic_test()`
Zeigt den augenblicklichen vom Mikrophon aufgezeichneten Geräuschpegel an. Dieser Wert wird ermittelt, indem das Mikrophon in sehr kurzen Abständen abgetastet wird. Ansonsten würden Geräusche von sehr kurzer Dauer, wie etwa ein Händeklatschen, eventuell nicht erfaßt.

`ir_test()`
Aktiviert abwechselnd die linke und rechte IR-Sendediode und zeigt an, ob eine Reflexion erfaßt wurde. Beachten Sie, daß die Leistung des IR-Detektors leidet, wenn er direkt fluoreszierendem Licht ausgesetzt wird.

`encoder_test()`
Zeigt über die beiden ganz rechten Benutzer-LEDs den Zustand der beiden Encoder an (HIGH oder LOW). Auf dem Bildschirm werden kontinuierlich die erfaßten Zählimpulse (Übergänge von HIGH auf LOW) ausgegeben. `encoder_test()` verwendet *nicht* die automatische Geschwindigkeitsüberwachungsfunktion aus `speed.icb`.

`motor_test()`
Aktiviert gemäß einer programmierten Folge von Geschwindigkeiten den linken und rechten Antriebsmotor. Vor dem Aufruf dieser Funktion sollten Sie die Antriebsmotoren an die Spannungsversorgung anschließen. Die Zahl nach dem `R` bzw. `L` auf der Anzeige gibt den Geschwindigkeitssollwert des rechten bzw. linken Rades als prozentualen Wert der Maximalgeschwindigkeit an. Die Zahl vor dem Buchstaben entspricht den tatsächlichen Geschwindigkeiten in Zähleinheiten pro Intervall. Während des Tests ist ein Intervall 0,5 Sekunden lang.

`user_digital_test()`
Zwei digitale Eingangsbits (PA1 und PA2) können vom Benutzer frei zugeordnet werden. Ihr Status wird kontinuierlich von `user_digital_test()` ermittelt. Sofern die Leitungen, die auf dem Erweiterungsstecker vorhanden sind, nicht mit HIGH oder LOW belegt sind, kann der angezeigte Wert zwischen 0 und 1 hin- und herspringen.

`analog_test(6|7)`
Für Benutzereingaben stehen zwei analoge Leitungen zur Verfügung, die den Mikroprozessor-Pins PE6 und PE7 zugeordnet sind. Der Wert dieser oder einer der anderen sechs analogen Leitungen wird durch `analog_test(n)` angezeigt, wobei `n` eine Zahl von 0 bis 7 ist.

`pyro_test(100,155)`
Ein Wärmesensor kann optional vom Benutzer hinzugefügt werden. Haben Sie im Rug Warrior einen Wärmesesor eingebaut, zeigt `pyro_test` seinen Zustand an. Die beiden Argumente geben die Empfindlichkeit und Reichweite der Thermoteranzeige in der oberen Zeile der LCD-Anzeige an. Wird der Wärmesensor einer einigermaßen gleichbleibend warmen Umgebung ausgesetzt, wird ein mehr oder weniger konstanter Wert angezeigt. Passiert eine Wärmequelle, wie etwa ein Mensch, den Erfassungsbereich des Sensors, steigt der Sensorwert zunächst an (bzw. fällt) und fällt dann wieder (bzw. steigt), wenn die Wärmequelle den Erfassungsbereich verlassen hat.

Der Temperaturanstieg oder -abfall hängt von der relativen Umgebungstemperatur in bezug auf die Wärmequelle und der Ausrichtung des Sensors ab.

2.11 Programmierung

Ausführliche Informationen zu Interactive C erhalten Sie in Kapitel 7. Das Roboterprogramm können Sie mit jedem Editor erstellen, der ASCII-Textdateien erzeugen kann. Hierfür eignen sich Editoren wie Emacs, Microsoft Word, SimpleText und viele andere. Eventuell müssen Sie ausprobieren, welche der Speicherfunktionen des Editors ein IC-kompatibles Format erzeugt. Nach dem Erstellen der Programmdatei können Sie das Programm mit dem Befehl `load` in den Rug Warrior laden, so wie wir es oben mit dem Programm `rw-test.lis` vorgeführt haben. Bevor Sie ein eigenes Programm schreiben, empfiehlt es sich jedoch, die Selbsttestroutinen und das Programm `demo-one.c` auszuprobieren. Diese Dateien zeigen anhand von Beispielen, wie Sie Sensoren ansprechen und Ihren Roboter zum Laufen bringen.

Kapitel 3

Montage der elektrischen Bestandteile

3.1 Notwendige Ausrüstung

Für den Zusammenbau der Rug Warrior-Platine benötigen Sie folgendes Werkzeug und Zubehör:

- Lötkolben. Der Lötkolben sollte ein qualitativ hochwertiges Instrument mit einer feinen Lötspitze sein. Er sollte eine regelbare Maximaltemperatur haben, da zu hohe Temperaturen die Bauteile beschädigen können.
- Kleiner, scharfer Seitenschneider
- Spitzzange
- Abisolierzange
- Kleines Multimeter
- Geeignetes Lötzinn

3.2 Zusätzliche Ausrüstung

Ein Multimeter, mit dem Sie Spannungen und Widerstände messen, hilft Ihnen, Kurzschlüsse und nicht geschlossene Schaltkreise zu ermitteln, und festzustellen, ob die richtigen Spannungspegel anliegen.

Ein Oszilloskop ist nicht nur hilfreich bei der Fehlersuche in Hochfrequenzschaltungen, sondern gibt auch Auskunft über die Funktionsweise der Sensoren des Roboters und anderer Schaltungen. Mit einem Oszilloskop lassen sich viele Beschaltungsfehler und andere Fehlerquellen in den Schaltungen des Roboters aufspüren.

Schraubstöcke zur Fixierung und Positionierung der Platine erleichtern ebenfalls die Montage. Wenn ein solches Gerät nicht verfügbar ist, empfiehlt es sich, die Bauteile während des Lötvorgangs mit doppelseitigem Klebeband an der Platine zu befestigen.

3.3 Vormontierte Teile

Die Platine des Rug Warrior wurde bereits im Werk zum Teil zusammengebaut und getestet. So sind der Mikroprozessor, der Speicher, die serielle Schnittstelle und einige andere Schaltlogikteile und zusätzliche Komponeten eingebaut. Sie können also sofort mit der Programmierung loslegen und den Mikrocontroller zur Fehlersuche verwenden, wenn Sie den Rest der Schaltung aufbauen.

Bevor Sie die übrigen Bauteile hinzufügen, führen Sie, wie in Kapitel 2 beschrieben, einen Selbsttest durch, sofern Sie dies noch nicht getan haben. Wir empfehlen außerdem, die Platine nach dem Installieren jedes neuen Teilsystems zu testen. So können Sie eventuell auftretende Fehler schneller lokalisieren.

Achten Sie schließlich auch darauf, daß die Spannungsversorgung unterbrochen ist, bevor Sie mit dem Anlöten neuer Bauteile beginnen.

3.4 Zusammenbau des Rug Warrior

Im Vergleich zu den meisten Platinen, die sich in Produkten der Unterhaltungselektronik befinden, ist die Rug Warrior-Platine einfach und verständlich aufgebaut. Trotzdem ist beim Zusammenbau und Einlöten der einzelnen Teile größte Sorgfalt geboten.

Anfängern empfehlen wir, die Kunst des Lötens und der Fehlersuche zunächst an weniger anspruchsvollen Projekten zu üben, bevor Sie sich an den Rug Warrior heranwagen. Wenn Sie z.B. zuvor die TuteBot-Schaltung auf einem Steckbrett aufgebaut haben, versuchen Sie einmal, die Schaltung auf eine Platine zu übertragen.

Die meisten ICs des Rug Warrior verwenden die CMOS-Technik. CMOS-Chips können jedoch durch elektrostatische Entladungen beschädigt werden. Viele Hersteller empfehlen deshalb, während der Montage ein Masseband am Handgelenk zu tragen. Nehmen Sie die Chips erst dann in die Hand, wenn Sie sie tatsächlich einbauen wollen. Achten Sie darauf, daß Sie nicht gerade über einen Teppich gelaufen sind, wenn Sie einen Chip anfassen. Der Funke, der von Ihren Fingern auf den Chip überspringt, könnte ihn beschädigen. Berühren Sie einen geerdeten Gegenstand (z.B. eine Heizung), bevor Sie einen Chip anfassen.

Zwicken Sie nach dem Löten mit einem Seitenschneider die Zuführungen der Bauteile dicht über der Platine ab.

Legen Sie die Platine nie auf leitenden Gegenständen (Drähte, abgezwickte Zuführungen, Lötzinn usw.) ab. Selbst im ausgeschalteten Zustand liegt an einigen Verbindungen noch eine geringe Spannung an, die zu einem Kurzschluß führen kann.

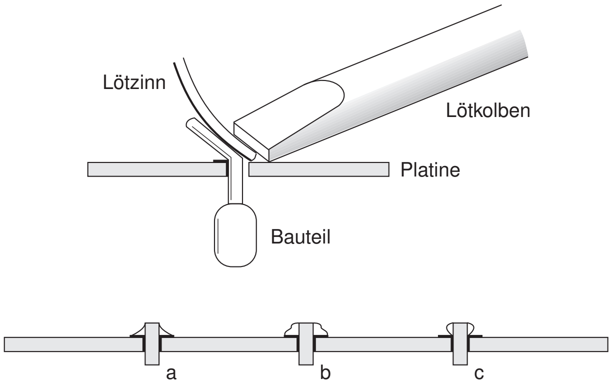

Abbildung 3.1: Löttechnik und Ergebnisse.

3.5 Löten

Rein theoretisch ist Löten ganz einfach: Zunächst werden Lötzinn *und* zu lötendes Objekt erwärmt; dadurch fließt das Lötzinn besser und stellt eine sicherere Verbindung zwischen den zu verbindenden Teilen her. Um dieses Resultat zu erzielen, ist jedoch ein gewisses Maß an Erfahrung erforderlich.

Abbildung 3.1 zeigt in einer Seitenansicht ein Bauteil, das auf eine Leiterplatine angelötet werden soll. Idealerweise berührt die Spitze des Lötkolbens die Zuführung des Bauteils, die Kontaktfläche und das Lötzinn. Sorgen Sie dafür, daß die Spitze des Lötkolbens stets mit Zinn überzogen ist, um eine gute Wärmeübertragung zu erhalten.

Die Teile *a*, *b* und *c* der Abbildung zeigen, was passieren kann: Bei einer guten Verbindung (*a*) ist die Lötstelle glatt und glänzend. *b* ist ein Beispiel für eine »kalte« Lötstelle. Die Oberfläche ist rauh und stumpf. Solche Verbindungen sind häufig Ursache für Wackelkontakte. In *c* wurde nur die Zuführung des Bauteils erwärmt, aber nicht die Kontaktfläche. Das Ergebnis ist eine Verbindung, die auf den ersten Blick zwar gut aussieht, jedoch keine verläßliche Verbindung darstellt. Die Fehler in *b* und *c* lassen sich durch erneutes Erwärmen und eventueller Zugabe von etwas mehr Lötzinn beheben.

Manchmal wird beim Löten auch zuviel Lötzinn verwendet oder ein Bauteil an der verkehrten Stelle angelötet. Hier gibt es zwei hilfreiche Geräte, mit denen sich das Lötzinn leicht wieder entfernen läßt. Das erste ist ein Entlötkolben. Er erwärmt die Lötstelle und saugt danach das Lötzinn ab. Daneben gibt es eine sogenannte »Entlötsaugpumpe«. Hier sorgt eine zurückspringende Feder für einen Unterdruck im Gerät, wodurch das Lötzinn aufgesogen wird. Beide Geräte sind in Elektronikläden oder entsprechenden Versandhäusern erhältlich.

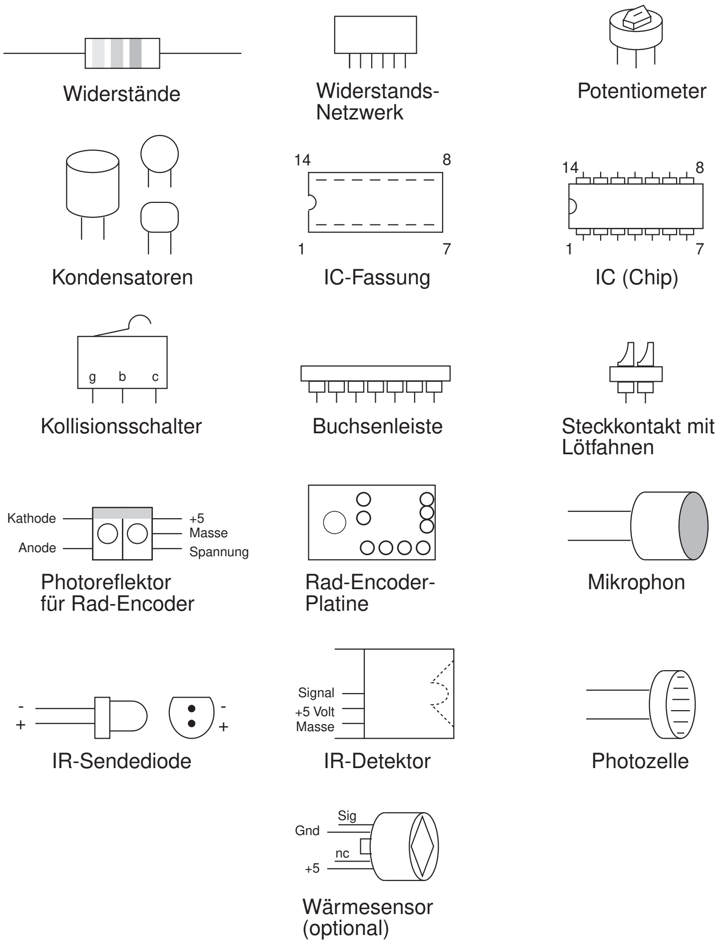

Abbildung 3.2: Kennzeichnung der Bauteile.

3.6 Montage

Dieser Abschnitt befaßt sich ausführlich mit dem Aufbau einer Schaltung. Die einzelnen Bauteile und ihre Polarität sehen Sie in Abbildung 3.2. Im Laufe der Konstruktion werden wir außerdem auf eine Reihe von Prüfverfahren eingehen.

3.6.1 Kabel

Alle Kabel, die der Rug Warrior benötigt, lassen sich aus dem mitgelieferten 10er-Verbindungskabel und den Steckkontakten mit Lötfahnen herstellen (siehe Abbildung 3.3). Teilen Sie das Kabel wie gezeigt in eine Gruppe von vier Leitungen (braun,

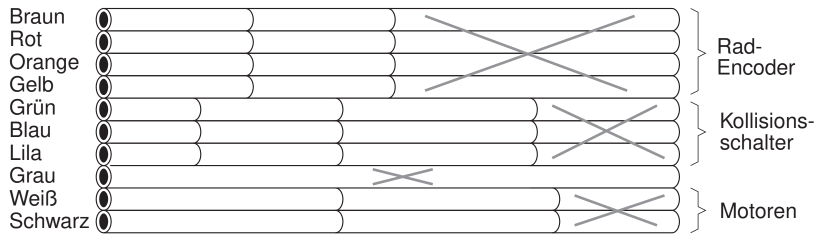

Abbildung 3.3: Teilen Sie das zehnadrige Flachbandkabel in Leitungen für die Rad-Encoder, Kollisionsschalter und Motoren auf.

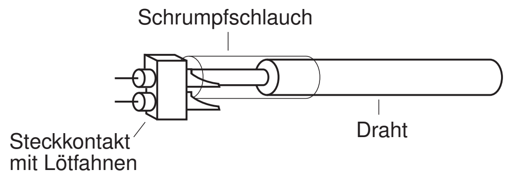

Abbildung 3.4: Anlöten eines Steckers: Zunächst den Draht abisolieren, anschließend verzinnen und den Steckkontakt anlöten. Schließlich Schrumpfschlauch erwärmen, um die Verbindung zu isolieren.

rot, orange und gelb), in eine Gruppe von drei Leitungen (grün, blau und lila) sowie in eine Gruppe von zwei Leitungen (weiß und schwarz). Achten Sie beim Schneiden jeder Gruppe auf die entsprechende Länge:

Rad-Encoder:
Zwei 4er-Verbindungskabel von je 1,20 m Länge.

Kollisionsschalter:
Je ein 3er-Verbindungskabel von 0,75, 1,45 und 1,65 m Länge.

Motoren:
Zwei 2er-Verbindungkabel von 2,10 m Länge.

Um eine gute Verbindung herzustellen, isolieren Sie zunächst ein Stück des Kabels ab (etwa 0,75 cm) und verzinnen Sie den blanken Draht (durch Überziehen mit Lötzinn). Ziehen Sie anschließend einen Schrumpfschlauch über das Kabel. Er sollte so lang sein, daß er die gesamte Verbindung bedeckt. Stecken Sie das verzinnte Ende auf den Steckkontakt des Pinsockels und fügen Sie, wenn notwendig, mehr Lötzinn hinzu. Ziehen Sie den Schrumpfschlauch über die gelötete Verbindung und erwärmen Sie ihn solange, bis er fest über der Verbindung sitzt (siehe Abbildung 3.4).

3.6.2 Mikrophon-Schaltung

Folgende Teile müssen eingebaut und angelötet werden:

- ◯ $10\,\mu$F Verstärkungskondensator. Der Pluspol wird in das untere der beiden Löcher gesteckt. Es ist mit einem + gekennzeichnet.
- ◯ $1000\,\mu$F Kondensator. Beachten Sie die Polarität des Kondensators. Da dieses Bauteil recht groß ist, sollte es auf der Seite liegend eingebaut werden.
- ◯ 2,2 kΩ Widerstand (Farbcode: rot, rot, rot) in der Mikrophonschaltung.
- ◯ $0{,}001\,\mu$F Plattenkondensator in der Mikrophonschaltung. (Auf dem Kondensator sollte unter anderem die Zahl 102 aufgedruckt sein.)
- ◯ Mikrophonfassung, eine aus einer 32er Buchsenleiste abgetrennte 2er Sockeleinheit. Einbau in die Löcher neben an der Position »Mikrophon« (siehe Abbildung 1.1).
- ◯ Sockel für den LM386, die 8-Pin IC-Fassung.
- ◯ Einbau des LM386 in die Fassung: Pin 1 in der linken unteren Ecke.
- ◯ Einbau des Mikrophons in seine Fassung: Beachten Sie die Polung des Mikrophons. Ein Pin des Mikrophons ist mit dem Gehäuse verbunden. Dieser Pin gehört in das Loch mit der Aufschrift GND (Masse) auf der Platine.
- ◇ Testen des Mikrophons: Installieren Sie das Selbsttestprogramm des Rug Warrior und rufen Sie den Mikrophontest `mic_test` auf. In einer ruhigen Umgebung sollte die LCD-Anzeige nichts oder nur einen sehr geringen Wert anzeigen. Sobald Sie sprechen, pfeifen oder gegen das Mikrophon blasen, sollte die Anzeige einen höheren Geräuschpegel registrieren.

Konstruktionshinweis: Bei einigen Anwendungen ist es eventuell empfehlenswert, das Mikrophon in einiger Entfernung zum Motor anzubringen, damit die aufzunehmenden Geräusche nicht von Motorgeräuschen übertönt werden. Verwenden Sie hierzu ein Koaxialkabel. Schließen Sie den mittleren Leiter an den Pluspol des Mikrophons an, und stecken Sie das andere Ende in das linke Loch der Mikrophonfassung. Verbinden Sie die beiden äußeren Litzen mit der Masse der Platine und dem Pin, der das Gehäuse mit dem Mikrophon verbindet.

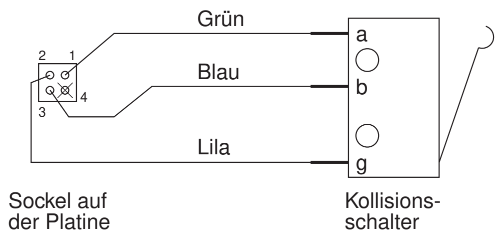

Abbildung 3.5: Schematische Darstellung des Kollisionsschalter-Kabels.

3.6.3 Sensoren und Aktuatoren

Hierfür müssen folgende Teile eingelötet werden:

◯ Zwei Motorstecker (2er Buchsenleisten) an den Positionen »linker Motor« und »rechter Motor« (siehe Abbildung 1.1).

◯ 16-Pin Fassung für den Motortreiber L293D (der L293D-Chip wurde durch einen SN754410 ersetzt).

◯ 74HC04 14-Pin Fassung.

◯ Fassung für die Kollisionsschalter, sechs paarweise anzulötende 2er Sockeleinheiten (siehe Abbildung 1.1 für die richtige Anordnung).

◯ Zwei 1,2 kΩ Widerstände (Farbcode: braun, rot, rot) für die Kollisionsschaltung.

◯ 2,2 kΩ Widerstand (Farbcode: rot, rot, rot) in der Kollisionsschaltung.

◯ 47 kΩ Widerstands-Netzwerk (ungepolt).

◇ Testen der Kollisionsschaltung. Schließen Sie alle Stecker der Kollisionsschaltung an einpolige Zweiwegeschalter an (siehe Abbildung 3.5), und rufen Sie den in Kapitel 2 beschriebenen `bumper_test()` auf, um sicherzustellen, daß die Bumper korrekt verdrahtet wurden.

Zur zusätzlichen Überprüfung können Sie die an Pin `PE3` anliegende Spannung messen, während Sie die Schalter betätigen. Jeder Schalter und jede Schalterkombination sollte eine eindeutige Spannung erzeugen, die sich um einen regelmäßigen Wert von

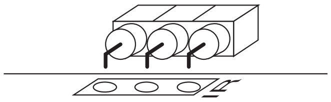

Abbildung 3.6: Einbau des IR-Detektors. Biegen Sie die Pins mit einer Zange um 90 Grad um.

den anderen unterscheidet. Folgende Schalterkombinationen und Spannungen sind möglich:

Schalter	Spannung	Schalter	Spannung
keiner	0,0 V	A & B	1,5 V
A	0,5 V	A & C	2,5 V
B	1,0 V	B & C	3,0 V
C	2,0 V	A & B & C	3,5 V

Durch die eindeutigen Spannungen kann der Roboter mit nur drei Schaltern zwischen sechs verschiedenen Richtungen unterscheiden.

Setzen Sie Installation mit folgenden Bauteilen fort:

- ◯ Zwei Rad-Encoder-Sockel mit insgesamt vier 2er Buchsenleisten. Installieren Sie diese in die auf dem Platinenlayout mit »Linker Rad-Encoder« bzw. »Rechter Rad-Encoder« gekennzeichneten Löcher (siehe Abbildung 1.1).
- ◯ Zwei Fassungen für die Photozellen (2er Buchsenleiste) an den Positionen »Linke Photozelle« und »Rechte Photozelle« (siehe Abbildung 1.1).
- ◯ Zwei Fassungen für die IR-Sendedioden (2er Buchsenleiste) an den Positionen »Linke IR-Sendediode« und »Rechte IR-Sendediode« (siehe Abbildung 1.1).
- ◯ Fassung für den IR-Detektor an der Position »IR-Detektor« (siehe Abbildung 1.1). Biegen Sie die Pins eines 3er Sockels mit einer Zange um, und bauen Sie den Sockel wie in Abbildung 3.6 gezeigt ein.
- ◯ Fassung für den Wärmesensor (zwei 2er Buchsenleisten) an der Position »Pyro-Sensor« (optional, siehe Abbildung 1.1).
- ◯ Zwei 680 Ω Widerstände (Farbcode: blau, grau, braun) für die Rad-Encoder-Schaltung. (Diese Widerstände für die IR LEDs in den Rad-Encodern dienen zur Begrenzung der Stromstärke.)
- ◯ Zwei 6,8 kΩ Widerstände (Farbcode: blau, grau, rot) für die Rad-Encoder-Schaltung.
- ◯ 5 kΩ Potentiometer für den IR-Oszillator an der Position »5K POT«.
- ◯ 0,001 μF Kondensator in der Oszillator-Schaltung (auf dem Kondensator sollte unter anderem die Zahl 102 aufgedruckt sein).

○ 6,8 kΩ Widerstand (Farbcode: blau, grau, rot) in der Oszillator-Schaltung.

○ 100 kΩ Widerstand (Farbcode: braun, schwarz, gelb) in der Oszillator-Schaltung.

○ 50 kΩ Potentiometer zur Einstellung des LCD-Anzeigenkontrastes an der Position »LCD-Poti« (siehe Abbildung 1.1).

○ Zwei 10 kΩ Widerstände (Farbcode: braun, schwarz, orange) für Photozellen.

○ Zwei 100 Ω Widerstände (Farbcode: braun, schwarz, braun) für IR-Sendedioden.

○ Stecken Sie die Lichtsensoren (ungepolte Potozellen) in ihre Fassungen. Sie können die Zuführungen etwas kürzen, damit die Sensoren flacher auf der Platine sitzen.

◇ Rufen Sie `photo_test` auf, und überprüfen Sie, ob die Pfeile auf der LCD-Anzeige auf die Photozelle weisen, die dem helleren Licht ausgesetzt ist.

○ Bauen Sie die Infrarot-LEDs (gepolt) ein. In beiden Fällen muß der Minuspol auf die Mitte der Platine zeigen. (Bei falschem Anschluß werden die Sendedioden nicht beschädigt.) Sie können die Zuführungen der IR-Sendedioden kürzen, damit sie flacher auf der Platine sitzen.

○ Löten Sie den Masseanschluß des Infrarot-Detektors GP1U52X mit einem kurzen Schaltdraht an das Gehäuse an. Dadurch nimmt der Detektor erheblich weniger Hintergrundrauschen auf.

○ Setzen Sie den Infrarot-Detektor GP1U52X auf seinen Sockel (mit dem Masseanschluß nach unten). Biegen Sie die Befestigungsdorne auf dem Detektorgehäuse ab, damit sie nicht irrtüm lich die Leitungen anderer Bauteile auf der Platine berühen und eventuell einen Kurzschluß produzieren.

○ Stecken Sie den xx74HC04x auf den Sockel (mit Aufschrift 74HC04). Die an der Stelle der x stehenden Zeichen variieren von Hersteller zu Hersteller. Der Chip muß so eingesteckt werden, daß sich Pin 1 in der unteren linken Ecke befindet.

◇ Testen Sie die Infrarotsensoren durch Aufruf von `ir_test`. Halten Sie einen dicken weißen Karton vor die IR-Sendediode, so daß das Licht zum IR-Detektor reflektiert wird. Korrigieren Sie das 40 kHz Oszillator-Potentiometer, bis Sie den maximalen Meßbereich erreicht haben.

○ Stecken Sie den Motortreiber-Chip SN754410xx in die Fassung mit der Aufschrift L293D. Pin 1 muß dabei in die untere linke Ecke eingefügt werden.

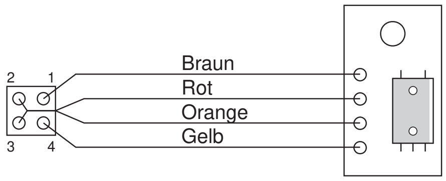

Abbildung 3.7: Rad-Encoder-Platine und Kabel. Der Photoreflektor wird auf der Oberseite der Platine montiert; auf der Unterseite befinden sich die Leiterbahnen.

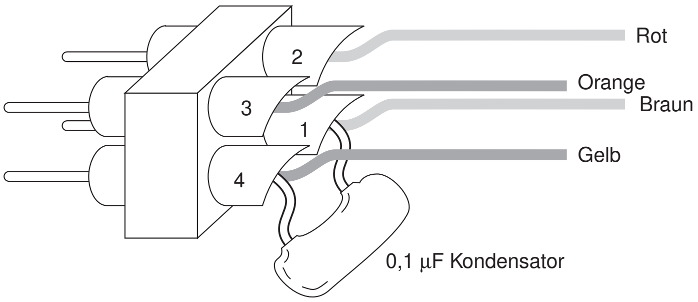

Abbildung 3.8: Ein 0,1 μF Ableitkondensator wird direkt an den Stecker eines Rad-Encoders angeschlossen.

Konstruktionshinweis: Wenn Sie Ihr eigenes mechanisches System entwerfen, wählen Sie Motoren, die im Leerlauf nicht mehr als 1 Ampere verbrauchen. Dies entspricht der maximalen Stromstärke, die der Motortreiberchip liefern kann. Die Spannunsversorgung des Motors kann eine Spannung im Bereich von 4,5 bis 35 Volt haben. Sie sollte um 1 bis 2 Volt höher sein als die Spannung, die an den Motoren anliegt, da es im Motortreiberchip zu einem Spannungsabfall kommt. Lesen Sie dazu auch den Abschnitt über die Motoren in Kapitel 6.

○ Montieren Sie die Photoreflektoren auf den Rad-Encoder-Platinen. Arbeiten Sie beim Löten besonders sorgfältig, damit der Photoreflektor-Chip nicht zu heiß wird oder schmilzt. Beachten Sie auch, daß die Befestigungslöcher sehr eng nebeneinander liegen. Vergewissern Sie sich nach dem Löten, daß keine Lötbrücken entstanden sind, durch die ein Kurzschluß an nebeneinanderliegenden Befestigungslöchern entstehen könnte.

○ Da die Rad-Encoder nicht direkt auf der Platine montiert werden, benötigen Sie ein Kabel. Verdrahten Sie die Rad-Encoderkabel wie in Abbildung 3.7 gezeigt. Schalten Sie einen 0,1 μF Ableitkondensator (mit der Zahl 104) mit den Phase- und Masseanschlüssen der beiden Rad-Encoderstecker parallel (siehe Abbildung 3.8).

Die Motoren und Rad-Encoder werden im Rahmen der Endmontage in Kapitel 4 getestet.

3.6.4 Jumper

Wenn Sie den Bausatz zusammenbauen, brauchen Sie die vom Werk vorgegebenen Jumpereinstellungen nicht zu ändern. Vielleicht möchten Sie aber die Standardeinstellungen ändern, wenn Sie Ihre eigene Basis- oder Spannungsversorgung konstruieren oder eigene Änderungen vornehmen.

Mit Jumper J1 können Sie festlegen, ob Sie eine gemeinsame oder zwei separate Spannungsversorgungen für die Schaltlogik und die Motoren verwenden wollen. Bei der Standardeinstellung wird eine separate Spannungsversorgung für die Motoren benötigt. Dies ist empfehlenswert, wenn Ihre Motoren relativ viel Hintergrundrauschen produzieren und hohe Ströme benötigen. Wenn Sie sich für eine gemeinsame Spannungsversorgung für die Motoren und die Schaltlogik entscheiden, sollten Sie die Leiterbahn (auf der Unterseite der Platine) trennen, die den oberen und mittleren Pin des J1 gekennzeichneten Bereichs verbindet. Anschießend verbinden Sie auf der Oberseite das mittlere und untere Loch mit einem kurzen Draht.

Die Einstellung des Jumpers J2 bewirkt, daß der MC68HC11 im *special test*-Modus betrieben wird. Dieser Modus wurde gewählt, da die LCD-Anzeige nur in diesem Modus betriebsbereit ist. Wenn Sie auf die LCD-Anzeige verzichten und im *expanded multiplexed*-Modus fortfahren möchten, trennen Sie die Leiterbahn, welche das mittlere und linke Loch des J2-Bereichs verbindet, und löten Sie einen Draht zwischen dem mittleren und rechten Loch an.

3.7 Zusammenfassung

Der Aufbau des Sensorschaltkreises für den Rug Warrior ist damit beendet. Ihre Platinen sollte voll funktionstüchtig sein. Rufen Sie noch einmal alle Tests der Datei `rw-test.lis` auf, um sicherzugehen, daß Ihr Aufbau fehlerfrei ist.

Wenn Sie auf einen Fehler stoßen, suchen Sie im Kapitel 6 nach Vorschlägen zur Behebung des Fehlers.

Kapitel 4

Montage der mechanischen Bauteile

Dieses Kapitel beschreibt den Zusammenbau der mechanischen Bauteile des Bausatzes sowie den Einbau des fertigen Elektronikteils. Neben dem Werkzeug für die Montage der elektronischen Teile benötigen Sie für den Mechanik-Bausatz:

- ◇ Bastel- oder Allzweckmesser (z. B. ein Präzisionsmesser)
- ◇ Einen kleinen Schlitzschraubendreher
- ◇ Eine kleine Zange
- ◇ Eine Schere

Im Mechanik-Bausatz sind folgende Komponenten enthalten (vgl. Abbildung 4.1 und Abbildung 4.2).

- ◇ 1 Chassis-Platte
- ◇ 1 Distanzstück für den Antriebsmotor
- ◇ 1 4×1 AA Batteriefach (der Elektronik-Bausatz enthält ein separates Batteriefach)
- ◇ 1 Schürze
- ◇ 1 Stützrad
- ◇ 1 Lenkachse
- ◇ 1 Gummiband
- ◇ 2 Antriebsräder
- ◇ 2 Antriebsmotoren mit Getrieben
- ◇ 2 selbstklebende Scheiben mit Streifenmuster
- ◇ 2 Kabelbinder

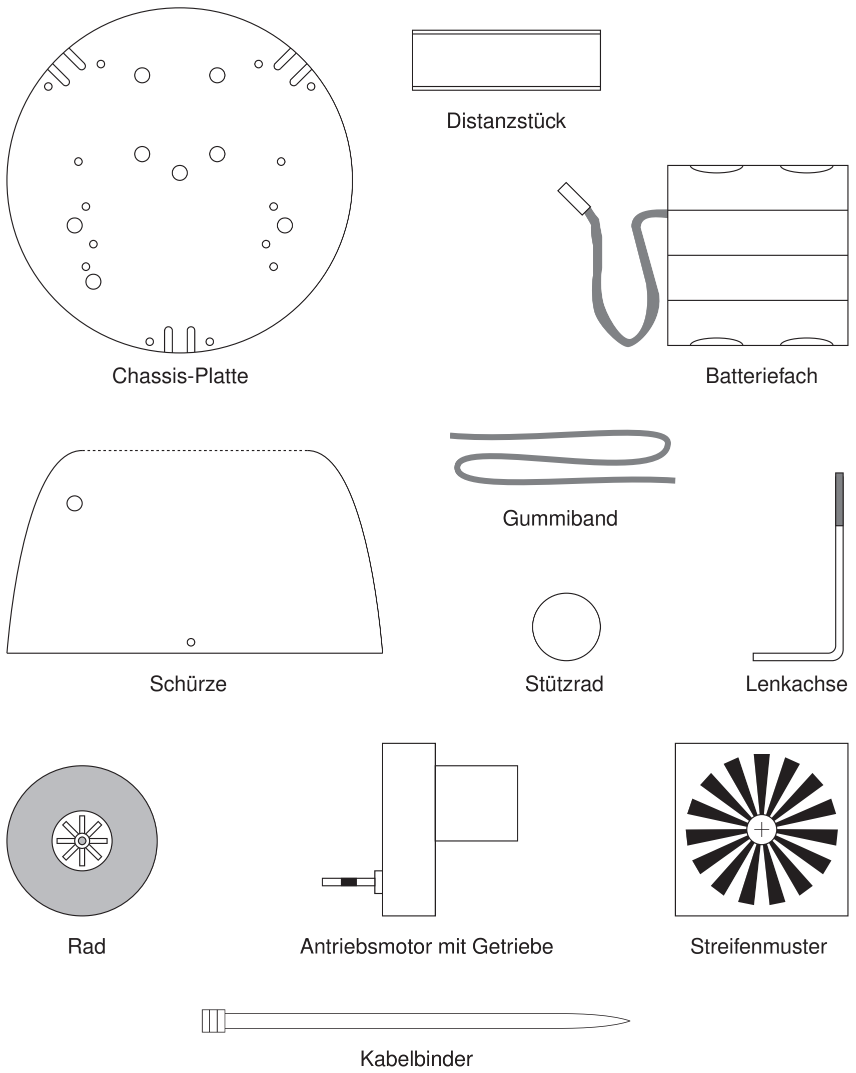

Abbildung 4.1: Mechanische Bauteile des Mechanik-Bausatzes

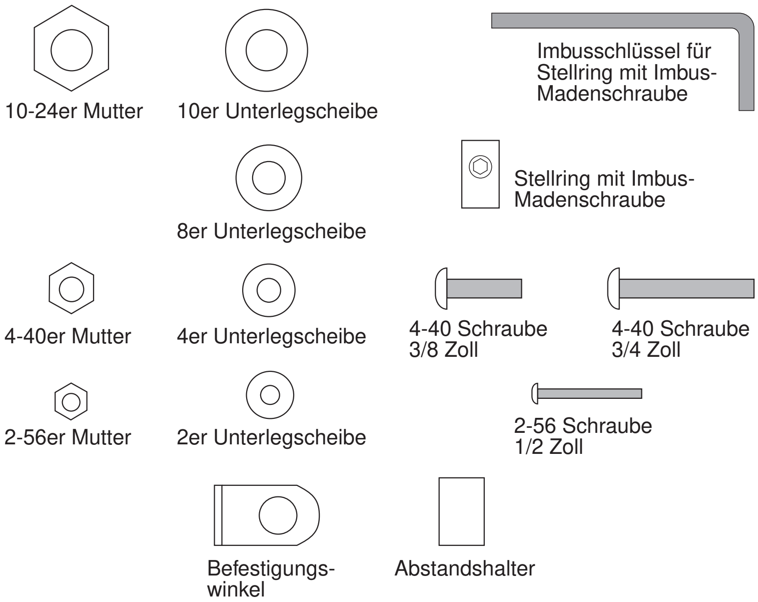

Abbildung 4.2: Befestigungsteile des Mechanik-Bausatzes

- 2 10-24er Muttern
- 2 #10 Unterlegscheiben
- 1 Inbusschlüssel für den Stellring
- 2 #8 Unterlegscheiben
- 1 Stellring mit Inbus-Madenschraube
- 20 4-40er Muttern
- 22 #4 Unterlegscheiben
- 4 4-40er, 3/4 Zoll Schrauben
- 16 4-40er, 3/8 Zoll Schrauben
- 6 2-56er Muttern
- 6 #2 Unterlegscheiben
- 6 2-56er Schrauben
- 4 Befestigungswinkel für die Motoren/Rad-Encoder
- 4 Abstandshalter für die Platine

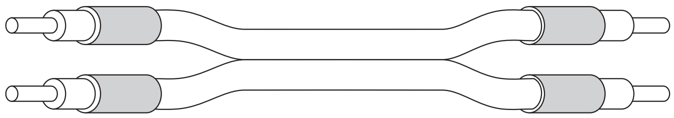

Abbildung 4.3: Bereiten Sie das Motorkabel wie gezeigt vor.

4.1 Antriebsmotoren

Verwenden Sie die schwarzen und weißen Drähte, die Sie vom zehnadrigen Kabel abgetrennt haben, und erstellen Sie daraus ein zweiadriges Kabel für die Motoren.

○ Isolieren Sie beide Enden der Motorkabel ab, und schieben Sie über alle Enden ein kurzes Stück Schrumpfschlauch, wobei Sie die abisolierten Enden frei lassen. (siehe Abbildung 4.3).

○ Biegen Sie die beiden Stromanschlußpins am Motor um 90 Grad nach außen. Löten Sie anschließend den weißen Draht des Motorkabels an den + Pol und den schwarzen Draht an den − Pol des Motors.

○ Schieben Sie den Schrumpfschlauch über die Lötstelle, und erwärmen Sie ihn, bis der Schlauch fest auf der Leitung sitzt. Wiederholen Sie den Vorgang für den zweiten Motor.

○ Löten Sie an die freien Enden der Motorkabel je einen 2-Pin Steckkontaktsockel und isolieren Sie die Lötstellen wie beim Motor mit einem Schrumpfschlauch.

4.2 Antriebsräder

An der Innenseite der Antriebsräder sind Pappscheiben mit Streifenmuster angebracht. Sie bilden einen Teil des Rad-Encodersystems des Rug Warrior. Wenn sich die Räder drehen, wird das auf die Kodierscheibe fallende Licht je nach Farbe entweder absorbiert oder auf einen gegenüberliegenden Photoreflektor reflektiert, der daraufhin ein Signal ausgibt. Damit ermittelt der Roboter, wie weit sich jedes Rad gedreht hat (siehe Abbildung 4.4).

○ Rauhen Sie mit einem Allzweckmesser an jedem Rad eine Seite etwas auf. Halten Sie die Schneide dabei stets senkrecht zur Radoberfläche, und kratzen Sie vorsichtig, damit Sie das Rad nicht aus Versehen zerschneiden.

○ Schneiden Sie mit einer Schere oder einem Allzweckmesser die beiden Muster aus. Schneiden Sie aus der Mitte der Scheibe soviel heraus, daß das Loch etwas kleiner ist als die Radnabe (etwa 2,2 cm im Durchmesser). Schneiden Sie, wie in der Abbildung gezeigt, kleine Schlitze in das Muster, damit sich die Scheibe besser an die Konturen des Rades anpassen kann.

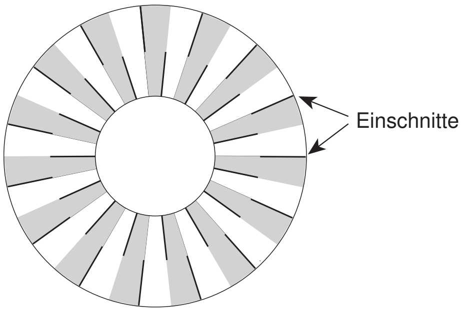

Abbildung 4.4: Einschnitte im Streifenmuster.

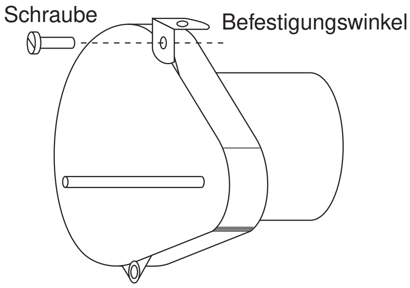

Abbildung 4.5: Rückansicht des Roboters auf den linken Motor und den Befestigungswinkel

○ Lösen Sie die ausgeschnittene Scheibe vorsichtig von der Unterlage und legen Sie diese auf die aufgerauhte Oberfläche des Rades. Drücken Sie den inneren Teil des Musters fest an, so daß das Papier von der Radnabe gehalten wird.

Konstruktionshinweis: Manchmal löst sich der äußere Teil des Streifenmusters trotz sorgfältiger Montage vom Rad. Sie können hier mit ein wenig Klebstoff am äußeren Rand nachhelfen.

○ Montieren Sie die Befestigungswinkel, wie in Abbildung 4.5 gezeigt, am Motor (noch nicht fest anziehen). Verwenden Sie eine 4-40er Schraube, sowie eine passende Unterlegschebe und Mutter. Die Montage am rechten Motor erfolgt spiegelbildlich zu der in der Abbildung gezeigten Montage am linken Motor.

○ Ziehen Sie auf jeder Motorwelle eine #8er Unterlegscheibe auf. Sie sorgt für den richtigen Abstand zwischen Rad und Motor.

- ◯ Montieren Sie die Räder auf den birnenförmigen Getriebesatz wie folgt: Legen Sie den Getriebesatz mit der nach oben zeigenden Welle flach auf den Tisch. Lassen Sie den Motor (den zylindrischen Teil) über die Kante des Tisches überhängen. Nehmen Sie ein Rad und drücken Sie es so auf die Welle, daß das Streifenmuster nach unten, d.h. auf den Getriebesatz zeigt. Dadurch wird verhindert, daß bei der Montage der Räder der Druck auf die Zahnräder des Getriebes übertragen wird.

Konstruktionshinweis: Wenn Sie jemals ein Rad abmontieren müssen, sollten Sie lieber die Welle herausdrücken, anstatt zu versuchen, das Rad vom Getriebesatz abzuziehen. Damit vermeiden Sie eventuelle Beschädigungen am Getriebe.

4.3 Das Chassis

Achten Sie darauf, daß die Chassis-Platte beim Montieren der Bauteile wie in Abbildung 4.6 ausgerichtet ist. Die Zeichnung zeigt eine Draufsicht – Motoren und zugehörige Komponenten werden an der Unterseite der Chassisplatte angebracht, Kollisionsschalter und die Platine auf der Oberseite der Platte.

- ◯ Kleben Sie ein Stück Isolierband auf die Rückseite der beiden Rad-Encoderplatinen. Stecken Sie von oben eine 3/8 Zoll 4-40er Schraube durch die Befestigungslöcher des Radencoders. Legen Sie zwei #4er Unterlegscheiben auf die Schraube, und ziehen Sie den Befestigungswinkel mit einer 4-40er Mutter an (siehe Abbildung 4.7). Die Montage des zweiten Rad-Encoders wird spiegelbildlich zur ersten durchgeführt.

- ◯ Bringen Sie den Motor/Befestigungswinkel-Satz locker mit 3/8 Zoll langen 4-40er Schrauben und Muttern sowie #4er Unterlegscheiben an der Unterseite der Chassisplatte an. Überprüfen Sie in Abbildung 4.6, welches die richtigen Löcher sind. Abbildung 4.7 zeigt die genaue Befestigung von Motor und Rad-Encoder.

- ◯ Schieben Sie das Distanzstück unter die Motoren, und drücken Sie beide Motoren gegen das Distanzstück, bis die Motoren richtig sitzen.

- ◯ Ziehen Sie einen Kabelbinder durch eines der Motorbefestigungslöcher und führen Sie ihn unter dem Motor zum gegenüberliegenden Befestigungsloch, so daß der Motor gegen den Aluminiumschacht gedrückt wird. Ziehen Sie den Kabelbinder fest. Wiederholen Sie anschließend den Vorgang am zweiten Motor.

- ◯ Ziehen Sie die Schrauben an, welche die Motoren mit den Motorbefestigungswinkeln verbinden. Zum Anschrauben müssen Sie die Gummiringe etwas zusammendrücken, damit der Schraubendreher in die Schlitze der 4-40er Schrauben paßt.

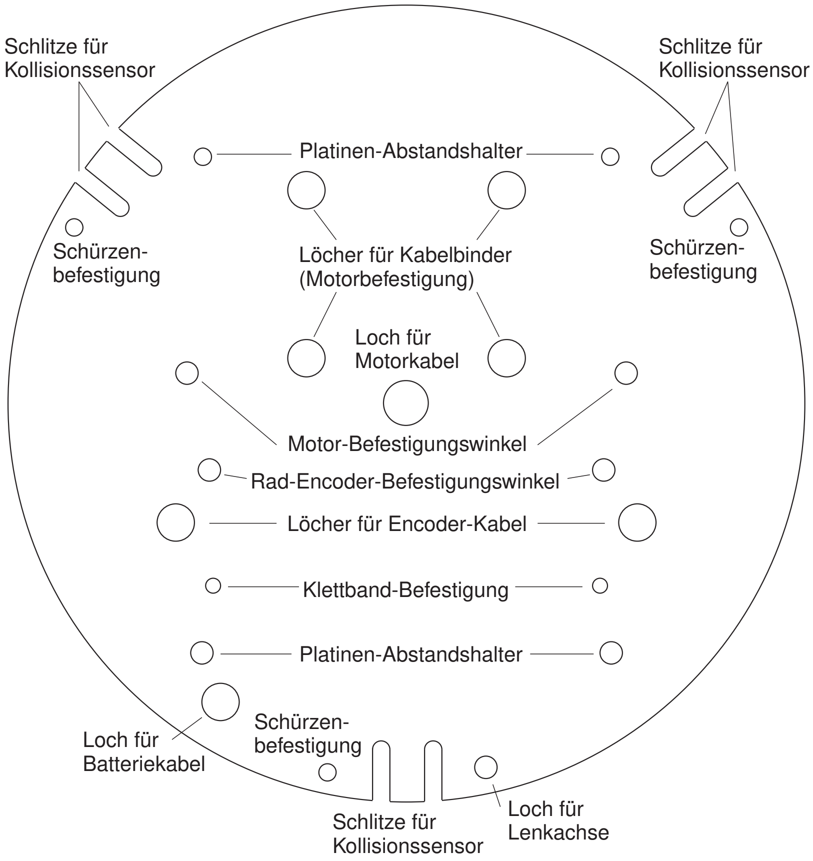

Abbildung 4.6: Funktionen der Löcher in der Chassis-Platte (Draufsicht)

- ◯ Ziehen Sie die Schrauben an, welche die Motorbefestigungswinkel mit dem Chassis verbinden.
- ◯ Führen Sie die Motorkabel durch die dafür vorgesehenen Löcher im Chassis.
- ◯ Montieren Sie die Rad-Encoder und zugehörigen Befestigungswinkel mit 4-40er Schrauben, Muttern und Unterlegscheiben.
- ◯ Führen Sie die Rad-Encoderkabel von unten durch die vorgesehenen Löcher der Chassisplatte.

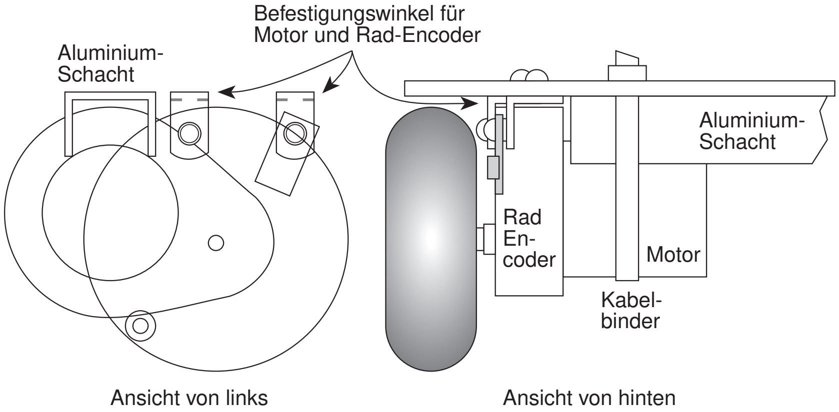

Abbildung 4.7: Motoranbringung am Chassis

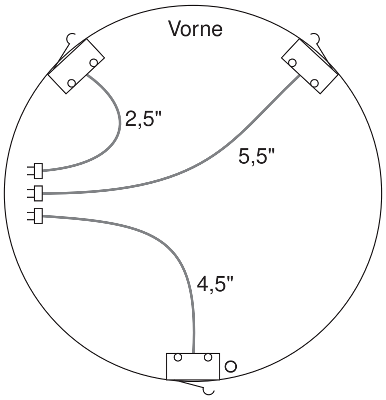

Abbildung 4.8: Position und Ausrichtung der Kollisionssensoren

○ Bringen Sie die drei Kollisionsschalter, wie in Abbildung 4.8 gezeigt, auf der Oberseite der Chassis-Platte mit 2-56er Schrauben und Unterlegscheiben an. Ziehen Sie die Schrauben jedoch noch nicht fest an, da die Kollisionsschalter beim Anbringen der Schürze noch ausgerichtet werden müssen.

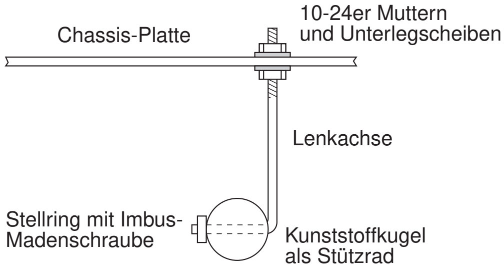

Abbildung 4.9: Montage des Stützrades

4.4 Das Stützrad

- ◯ Setzen Sie das Kunststoffrad mit einem Durchmesser von 2,5 cm auf die Lenkachse (siehe Abbildung 4.9). Halten Sie das Rad mit dem Stellring am Platz. Ziehen Sie die Inbus-Madenschraube mit dem mitgelieferten Inbusschlüssel fest an. Vergewissern Sie sich anschließend, daß das Rad auf der Lenkachse frei rotieren kann.
- ◯ Bringen Sie die Stützradkonstruktion wie gezeigt am Chassis an. Verwenden Sie dafür je zwei 10-24er Muttern und Unterlegscheiben. Achten Sie beim Anziehen der Muttern darauf, daß die Chassisplatte eben ist.

4.5 Batteriefächer

- ◯ Stechen Sie mit einer spitzen Schere oder einer Messerklinge in beide Enden des Klettbandes (etwa 0,5cm vom Rand entfernt) ein Loch.
- ◯ Befestigen Sie das Klettband an der Unterseite des Chassis mit 4-40er Schrauben, Unterlegscheiben und Muttern. Setzen Sie das Batteriefach auf das Klettband; verdrehen Sie ein Ende des Klettbandes, und lassen Sie die beiden Enden über dem Batteriefach zusammenkletten (siehe Abbildung 4.10).

4.6 Schürze und Platine

Die Schürze gleitet, an drei Gummibändern aufgehängt, um das Chassis. Die Schürze ist so befestigt, daß bei einer Kollision aus einer beliebigen Richtung einer oder mehrere der Kollisionsschalter ausgelöst werden. Abbildung 4.11 zeigt die Anbringung der Gummibänder.

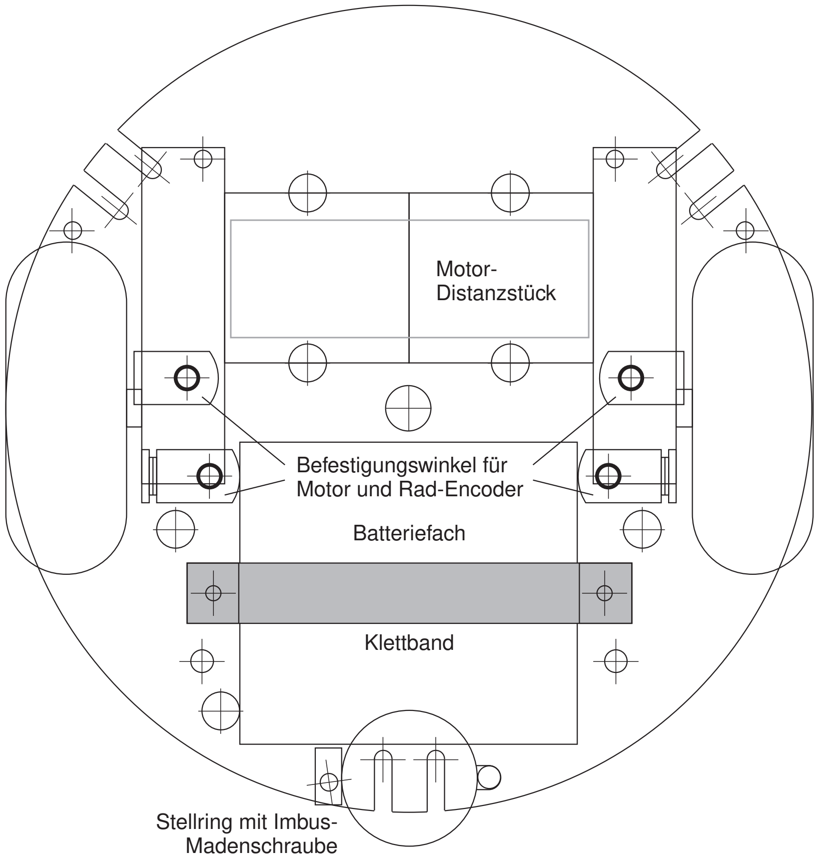

Abbildung 4.10: Draufsicht auf die zusammengebaute Platte

○ Schneiden Sie das Gummiband in Teile von ca. 7,5 cm Länge.

○ Stechen Sie in jedes Ende der Gummmibänder mit einem Stift oder einer spitzen Schere ein Loch (0,5 cm vom Rand entfernt). Führen Sie das eine Ende der Gummibänder durch die vorgesehenen Löcher in der Chassisplatte und befestigen Sie die Bänder mit 3/8 Zoll langen 4-40er Schrauben und Muttern sowie #4er Unterlegscheiben. Die Gummibänder sollten radial zur Mitte der Chassisplatte angebracht sein.

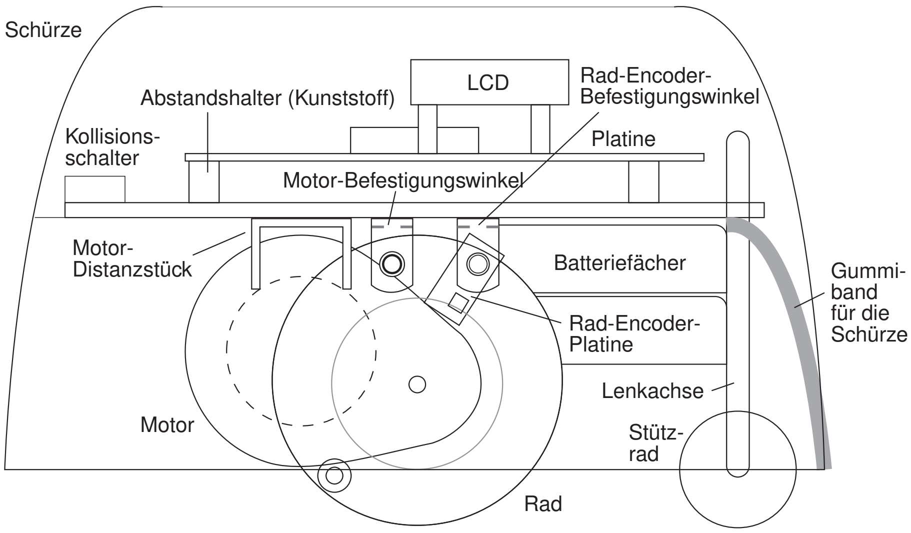

Abbildung 4.11: Seitenansicht des montierten Mechanik-Bausatzes

- ◯ Befestigen Sie die Platine auf der Oberseite des Chassis mit vier Abstandshaltern, 3/4 Zoll langen 4-40er Schrauben und Muttern sowie #4er Unterlegscheiben. Schließen Sie die Kabel für die Kollisionsschalter, Rad-Encoder und Motoren an die entsprechenden Sockel auf der Platine an. Achten Sie darauf, daß bei den Motorkabeln der weiße Draht von hinten betrachtet links liegt.

- ◯ Stellen Sie die fertige Konstruktion mit den Rädern nach unten auf den Tisch, und ziehen Sie die Schürze darüber. Bringen Sie die Schürze so an, daß die beiden größeren Löcher, durch welche die IR-Sendedioden zeigen sollen, nach vorne gerichtet sind.

- ◯ Befestigen Sie die Gummibänder an den drei Löchern am unteren Rand der Schürze. Die Schürze sollte etwa einen Zentimeter über dem Boden hängen.

- ◯ Korrigieren Sie, wenn nötig, die Anbringung der Kollisionsschalter sowie die Befestigung der Gummibänder/Schürze. Wenn die Schürze nicht in Bewegung ist, darf keiner der Kollisionsschalter heruntergedrückt sein. Vergewissern Sie sich, daß jeder der Schalter aktiviert wird, wenn auf die Schürze aus der entsprechenden Richtung Druck ausgeübt wird. Versuchen Sie tote Punkte oder Bereiche zu vermeiden, in denen die Schürze gedrückt werden kann, ohne daß einer der Kollisionsschalter ausgelöst wird. Wenn alle Schalter zu Ihrer Zufriedenheit angeordnet sind, ziehen Sie die Schrauben an den Befestigunswinkeln fest an.

◯ Befestigen Sie die beide Batteriefächer mit dem Klettband an der Unterseite des Chassis. Führen Sie die Kabel durch die entsprechenden Löcher in der Chassis-Platte, und schließen Sie die Kabel an die Spannunsversorgung des Motors und der Schaltlogik an.

◯ Bringen Sie die Batteriefächer soweit hinten wie möglich an, damit der Roboter im Gleichgewicht bleibt. Da der Roboter die Tendenz hat, bei einer raschen Richtungsumkehr nach vorne überzukippen, müssen Sie ihn eventuell hinten zusätzlich beschweren.

4.7 Feineinstellungen

◯ Halten Sie den Roboter hoch oder hängen Sie ihn auf, so daß die Räder keinen Kontakt zum Boden haben. Geben Sie einen Befehl für einen Vorwärtslauf der Motoren an, z.B. `drive(100,0);`. Beide Räder sollten sich vorwärts drehen. Ist dies nicht der Fall, schließen Sie das Kabel, das zum Motor mit der falschen Laufrichtung führt, andersherum an.

◇ Laden Sie die Datei `rw-test.lis`, und rufen Sie die Routine `motor_test` auf. Überprüfen Sie, ob der Roboter wie erwartet reagiert. Während der anfänglichen Einlaufphase des Motors werden Sie vielleicht feststellen, daß der Roboter leicht nach links oder rechts driftet, obwohl er geradeaus fahren sollte. Nach etwa 15 bis 30 Minuten Motorbetrieb sollte jedoch dieser Fehler von selbst behoben sein. Kleinere Differenzen können jedoch nie ausgeschlossen werden und müssen mit entsprechender Software zur Abfrage der Rad-Encoder behoben werden.

◇ Laden Sie den Rad-Encodertest aus den mitgelieferten Selbsttestroutinen, und rufen Sie ihn auf. Bewegen Sie die Räder per Hand, und drehen und verschieben Sie dabei die Rad-Encoderplatine, bis Sie ein zuverlässiges Signal erhalten. Vergewissern Sie sich, daß der Rad-Encoder jeden Übergang von einem dunklen auf einen hellen Streifen erfaßt (und zwar nur diesen). Solange dies nicht gewährleistet ist, sind alle Abstandsmessungen des Roboters unzuverlässig.

◯ Schieben Sie über jede IR-Sendediode ein etwa 1 bis 1,5 cm langes Stück Schrumpfschlauch, und richten Sie die Dioden auf die Löcher in der Schürze aus.

◇ Testen Sie das IR-Hinderniserfassungssystem mit Hilfe der Routine `ir_test`. Sind die Sendedioden nicht richtig ausgerichtet, kann die Kunststoffschürze intern Reflexionen erzeugen, die der Detektor fälschlicherweise als Hindernis interpretiert. Eventuell müssen Sie die Rückseite der Dioden mit Isolierband abdecken, um Streulicht zu vermeiden, da der Detektor recht empfindlich ist. Wenn Sie eine Diode während des Tests abdecken müssen, verwenden Sie nicht Ihren Daumen, sondern schwarzes Isolierband. Durch Ihre Daumen und Finger würde das IR-Licht hindurchgehen.

Und damit ist der Rug Warrior einsatzbereit. Zunächst sollten Sie das Programm `demo-one.c` auf der Programmdiskette ausprobieren. `demo-one` verfügt über drei Betriebsmodi, zwischen denen Sie durch Drücken der RESET-Taste wählen können (genau wie bei `rw-test.lis`). Der aktuelle Modus wird auf der LCD-Anzeige angezeigt.

Der erste Modus »Seek light« (Lichtverfolgung) bewirkt, daß der Roboter sich in Richtung der hellsten Lichtquelle bewegt. Dabei überwacht der Rug Warrior ständig seinen IR-Hindernisdetektor und ändert bei einer bevorstehenden Kollision den Kurs. Außerdem werden die Kollisionssensoren überwacht. Hat ein Zusammenstoß stattgefunden, setzt der Roboter zurück und versucht, das Hindernis zu umgehen. Versuchen Sie den Roboter zu veranlassen, Ihnen zu folgen, indem Sie eine Taschenlampe auf seine Photozellen richten.

Der zweite Betriebsmodus »Seek darkness« (Verstecken) bewirkt, daß der Roboter Licht vermeidet. Auch hier überwacht der Rug Warrior den IR-Detektor und die Kollisionssensoren, während er den dunkelsten Platz sucht, an dem er stehenbleibt.

Im dritten Betriebsmodus »Wait for whistle« (Warten auf einen Ton) überwacht der Rug Warrior seine Mikrophonschaltung, wobei er zunächst keine Aktion ausführt. Sobald er ein lautes Geräusch empfängt, aktiviert er seinen Piezo-Summer.

Nutzen Sie die Beispiele und Ansätze dieses Demoprogramms als Anregung, um selbst eigene Anwendungsprogramme für Ihren Roboter zu schreiben.

Kapitel 5

Ausbau des Rug Warrior

Der Rug Warrior verfügt über vier weitere Leitungen zum Einlesen zusätzlicher Sensordaten, und auf seiner Platine ist ein frei verfügbares Signal zur Steuerung eines weiteren Aktuators. Wenn Sie die Schaltung noch ein wenig weiter ausbauen, kann der Rug Warrior sogar jede beliebige Anzahl weiterer Ein- und Ausgaben verarbeiten.

5.1 Eingebaute Ports

Der Erweiterungsstecker hat zwei digitale Eingänge (PA1 und PA2) sowie zwei analoge Eingänge (PE6 und PE7), die frei zugeordnet werden können. Ebenso befindet sich auf diesem Stecker ein digitaler Ausgabe-Port (PA4). Dieser Port wird jedoch auch von der LCD-Anzeige benutzt und kann daher nur für Erweiterungen verwendet werden, wenn der LCD-Softwaretreiber deaktiviert wurde. Dazu sollten Sie den Mikrocontroller im expanded-Modus und nicht im special test-Modus betreiben. Wenn Sie keinen Wärmesensor installiert haben, können Sie außerdem den analogen Eingang PE5 verwenden, der ursprünglich für den Wärmesensor vorgesehen war.

Die Leitung PA3, die den Piezo-Summer steuert, kann zusätzlich zum Antrieb eines RC-Servomotors oder zur Steuerung eines weiteren Aktuators verwendet werden. Um PA3 zu nutzen, installieren Sie einen Sockel, oder löten Sie einen Draht auf der Platine an, wie in Abbildung 5.1 gezeigt.

Neben den erwähnten Leitungen stellt der Erweiterungsstecker die eindirektionalen Adreßleitungen des MC68HC11 Bus bereit. Adreß- und Dateleitungen für den Multiplexbetrieb stellt der LCD-Stecker zur Verfügung (siehe Abbildung 5.2). Mit Hilfe dieser Adreß- und Datenleitungen sowie anderer Steuersignale können Sie jede beliebige Anzahl von speicherorientierten I/O-Ports erstellen. Um die speicherorientierten Erweiterungsfunktionen des Rug Warrior voll auszuschöpfen, empfehlen wir dem erfahrenen Roboterkonstukteur, sich einen Satz Benutzerhandbücher für den MC68HC11A von Motorola anzuschaffen. Wenden Sie sich an die Motorola-Zentrale in München.

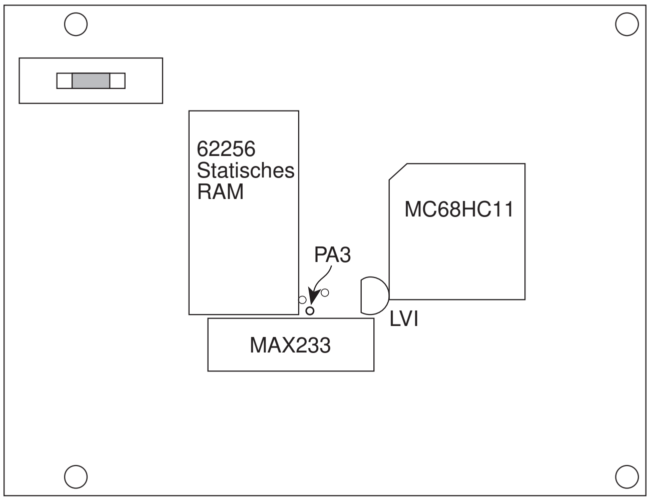

Abbildung 5.1: An der markierten Stelle befindet sich die frei verfügbare Leitung PA3 auf der Platine.

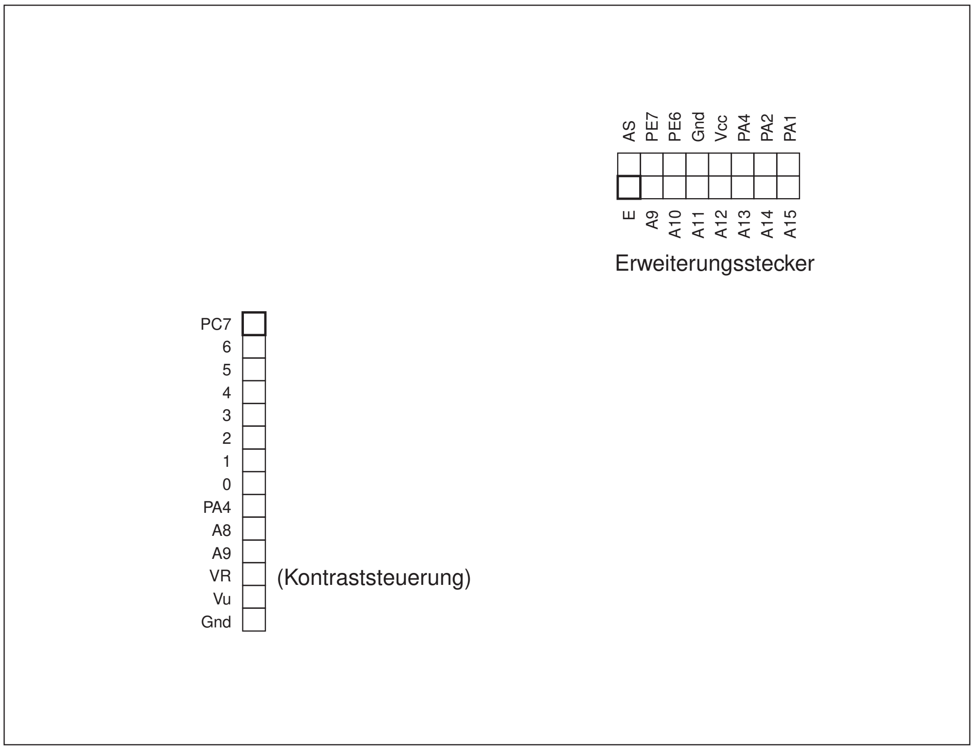

Abbildung 5.2: LCD-Anschluß und Erweiterungsstecker

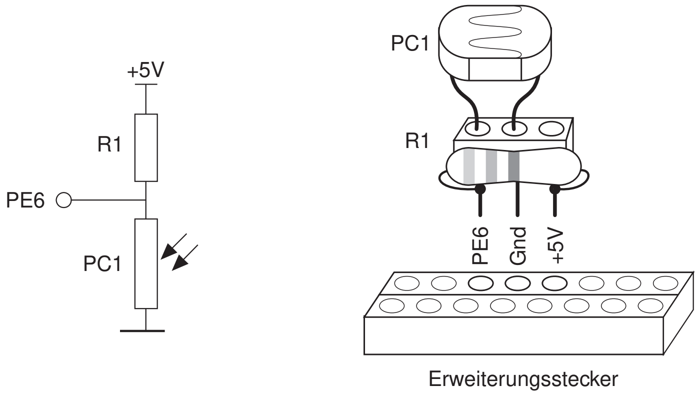

Abbildung 5.3: Eine auf einem 3er-Sockel montierte Photozellenschaltung wird direkt auf den Erweiterungsstecker aufgesteckt.

5.2 Beispiel für eine Erweiterung

Angenommen, wir wollen dem Rug Warrior die Fähigkeit verleihen zu erkennen, wenn er unter einen Tisch oder einen Stuhl gefahren ist. Vielleicht schreiben wir gerade ein Programm, das den Roboter anweist, sich in einer dunklen Ecke zu verstecken, bis er ein lautes Geräusch hört, um dann hervorzustürmen und denjenigen, der gerade den Raum betreten hat, zu überraschen. Dies läßt sich relativ einfach realisieren, indem wir am Rug Warrior eine nach oben gerichtete Photozelle anbringen.

Normalerweise sollte eine Erweiterungsschaltung auf einer kleineren Experimentierplatte (eine Platine mit vorgebohrten Löchern und einer Kupferbeschichtung) aufgebaut werden. Eine Photozellenschaltung ist jedoch so einfach, daß sie bequem auf einen 3er-Sockel paßt (siehe Abbildung 5.3).

Das Schaltdiagramm auf der linken Seite in Abbildung 5.3 zeigt, wie die Schaltung funktioniert. Der Widerstand R1 und die Photozelle PC1 bilden einen Spannungsteiler. Wir haben R1 so gewählt, daß sein Widerstand dem Widerstand von PC1 entspricht, wenn diese einer »mittleren« Lichtintensität ausgesetzt ist. Wenn Sie für diese Schaltung dieselbe Art von Photozelle verwenden wie jene, die mit dem Elektronik-Bausatz mitgeliefert wird, sollte R1 etwa $10\,\mathrm{k}\Omega$ betragen.

Rechts in Abbildung 5.3 ist der Aufbau der Schaltung zu sehen. Zunächst wird der Widerstand R1 auf den ersten und dritten Pin des 3er-Sockels aufgelötet, während PC1 in den ersten und zweiten Pin des Sockels gesteckt wird. Danach wird der Sockel auf die Anschlüsse `PE6`, `Gnd` und `+5V` des Erweiterungssteckers gesteckt. Softwaremäßig können wir auf den neuen Sensor mit der Funktion `analog(6)` zugreifen.

Der Rug Warrior unterstützt noch eine Vielzahl weiterer Sensoren und Aktuatoren. Eine ausführliche Beschreibung über den Einbau eines RC-Servomotors und eines Ultraschall-Entfernungsmessers in den Rug Warrior finden Sie in der Herbst- und Winterausgabe 1995 der Zeitschrift *The Robotics Practitioner* unter dem Titel »RugNav: a Scanning Sonar You Can Build« (Teil 1 und 2). Sollte diese Zeitschrift in Ihrer Bibliothek nicht vorhanden sein, können Sie *The Robotics Practitioner* über E-Mail unter der Adresse `trp@footfalls.com` direkt erreichen oder die Artikel unter der folgenden Postanschrift bestellen: Footfalls, Ltd., 483 S. Kirkwood Road, Suite 130, Kirkwook, MO 63122, USA.

Kapitel 6

Fehlerbeseitigung

Dieser Abschnitt enthält einige Vorschläge zur Beseitigung von Fehlern, die während des Roboterbaus auftreten können.

Platine funktioniert gar nicht

Wenn nicht einmal die Betriebsleuchte aufleuchtet, prüfen Sie, ob die Batterien genügend Spannung haben (mindestens 5 V). Vergewissern Sie sich auch, daß der Stecker richtig eingesteckt ist und daß weder er noch die Verbindungsdrähte defekt sind. Wenn Sie neue Bauteile hinzugefügt haben, prüfen Sie, ob diese einen Kurzschluß verursacht haben. Ziehen Sie den Batteriestecker von der Platine ab, und messen Sie den Widerstand zwischen Phase und Masse. Ist der Widerstand 0 oder nur ein paar Ohm, sehen Sie die Platine sorgfältig nach Lötbrücken, Wackelkontakten oder Kurzschlüssen durch.

Wenn Sie während des Testens einige der gesockelten ICs herausgezogen haben, achten Sie darauf, daß Sie diese wieder richtig herum reinstecken. Pin 1 ist durch einen Punkt oder Halbkreis markiert und sollte bei allen gesockelten Chips auf der linken Seite sein. Eine Ausnahme bildet der 28-Pin RAM-Chip, bei dem der Pin 1 oben liegt. Da Chips bei einem falschen Einbau beschädigt werden können, sollten Sie im Falle eines Falles besser ersetzt werden. Auch durch elektrische Entladungen können die Chips Schaden erleiden.

Ein guter Indikator für einen defekten Chip ist eine erhöhte Temperaturentwicklung. Außer dem Motortreiberchip (SN754410) sollte sich bei Normalbetrieb kein Chip erwärmen.

P-Code kann nicht geladen werden

Beim Versuch den P-Code vom Rechner auf die Platine zu laden (die Platine zu initialisieren), trat entweder ein System-Timeout auf, oder es wurde die Fehlermeldung angezeigt, daß ein anderes Zeichen empfangen als gesendet wurde. Prüfen Sie, ob der Schalter tatsächlich in der Position »Download« steht und nicht in der Stellung »Ausführen« (siehe Abbildung 1.1). Weitere Ursachen können eine zu niedrige Batteriespannung oder ein defekter Stecker sein. Die »BAT ON«-LED muß aufleuchten, die Batteriestandsanzeige »Batterie leer«sollte nicht aufleuchten. Messen Sie die an der Platine anliegende Spannung. Sie sollte mindestens 5 V betragen.

Eventuell ist auch die serielle Schnittstelle in Ihrem Computer falsch konfiguriert oder angeschlossen. Prüfen Sie alle Systemparameter, welche die serielle Schnittstelle betreffen. Der serielle Ausgang könnte beispielsweise auf ein internes Modem gelegt worden sein. Vergewissern Sie sich auch, daß das serielle Kabel in die richtige Buchse eingesteckt ist: Modem und nicht Drucker, bzw. der richtige COM-Port.

Das serielle Kabel könnte auch schlecht verdrahtet sein oder es besteht eine schlechte Verbindung. Eine schlechte Verdrahtung kann entstehen, wenn Sie ein anderes als das im Bausatz mitgelieferte Modemkabel verwenden. Überprüfen Sie alle Verbindungen und Kabel zwischen dem seriellen Ausgang Ihres Computers und der Rug Warrior-Buchse für den seriellen Anschluß. Beim Bau eigener Leitungen werden häufig die Empfangs- und Sendeleitungen irrtümlich vertauscht (dies richtet keinen Schaden an). Ist dies der Fall, tauschen Sie die Leitungen einfach um, und wiederholen Sie den Ladevorgang.

Ist das Problem damit noch nicht behoben, versuchen Sie den Fehler auf folgende Weise zu lokalisieren: Ziehen Sie den seriellen Stecker von der Buchse für den seriellen Anschluß auf der Rug Warrior-Platine ab. Verbinden Sie die beiden äußeren Kontakte des vieradrigen Steckers, die Sende- und Empfangsleitung, mit einem dünnen Draht (hierzu benötigen Sie eventuell Hilfe). Starten Sie währendessen das Programm zur Initialisierung der Platine. Durch die Verbindung der Sende- und Empfangsleitung haben Sie eine Rückkopplung geschaffen. Sie gewährleistet, daß das Initialisierungsprogramm dieselben Zeichen erhält, die es gesendet hat.

Wenn am Ende des Initialisierungsprogramms das Bootstrap-Programm erfolgreich geladen wird (ohne Timeout oder Fehlermeldung), ist dies ein Indiz dafür, daß Ihre seriellen Leitungen und Kabel richtig konfiguriert sind und das Problem eventuell die Rug Warrior-Platine ist.

Wenn das Laden des Bootstrap-Programms durch das Initialisierungsprogramm fehlschlägt, ist entweder das serielle Kabel oder die Verbindung defekt, oder die Konfiguration stimmt nicht.

Platine scheint richtig initialisiert zu sein, aber der IC spricht nicht an

Wenn die Initialisierung der Platine abgeschlossen ist, legen Sie den Schalter auf die Stellung »Ausführen«, und drücken Sie die RESET-Taste. Ertönt vom Piezo-Summer kein Ton, hat die Initialisierung nicht geklappt. Prüfen Sie alle Punkte noch einmal durch, und starten Sie die Initialisierung neu. Vergewissern Sie sich auch, daß der P-Code nicht defekt ist. Wenn Sie sich nicht sicher sind, ob der P-Code in Ordnung ist, tauschen Sie die Datei `pcoderwl.s19` auf Ihrer Festplatte durch die Kopie auf der mitgelieferten Programmdiskette aus.

Verhalten der Platine ist »unberechenbar«

Die häufigste Fehlerursache für ein fehlerhaftes Verhalten der Platine, z. B. Absturz und Neustart, sind: 1) eine zu geringe Batteriespannung, 2) durch die Motoren ausgelöste Spannungsspitzen. Wenn die »Batterie leer«-LED aufleuchtet, deutet dieses darauf hin, daß der Unterspannungsmelder-Chip aktiviert wurde. Dadurch wird der Prozessor angehalten und anschließend neu gestartet. Prüfen Sie, ob die Batterien für die Schaltlogik aufgeladen sind. Sollten die Batterien in Ordung sein, vergewissern Sie sich, daß alle neu hinzugefügten Sensoren- und Aktuatorenschaltungen korrekt abgeschlossen sind und nicht mehr Strom verbrauchen, als die Batterien liefern können.

Wenn das unberechenbare Verhalten der Platine nur während des Betriebs der Motoren auftritt, deutet dies auf Spannungsspitzen hin. Sie sollten nicht vorkommen, wenn Sie die Motoren aus dem Mechanik-Bausatz verwenden. Wenn Sie Ihr eigenes Fortbewegungssystem entwickelt haben, sind Spannungsspitzen jedoch durchaus möglich. Sie sollten für die Schaltlogik und die Motoren separate Spannungsversorgungen verwenden. Wenn Sie bereits separate Spannungsversorgungen verwenden, produzieren die Motoren eventuell ein zu hohes Rauschen (siehe unten).

Prozessor stürzt bei Motorbetrieb ab

Wenn Ihre Platine aufgrund der Nebengeräusche des Motors Ihres selbstentwickelten Fortbewegungssystems nicht richtig funktioniert, können Sie ein paar Tricks ausprobieren, bevor Sie Ihre Motoren gegen bessere (und teurere) austauschen. Schließen Sie je einen Kondensator (mit einer Kapazität von weniger als einem Mikrofarad, z. B. 0,1 μF) parallel zu den beiden Motoren direkt am Motor an. Damit können Sie die an den Motorbürsten erzeugten Spannungsspitzen reduzieren. Eine andere Möglichkeit ist, einen großen *ungepolten* Elektrolytkondensator mit dem Motor parallel zu schließen. Hier gilt, je größer die Kapazität ist, umso besser. Beginnen Sie mit 10 μF. Verwenden Sie zu diesem Zweck nicht die gebräuchlicheren polarisierten Elektrolytkondesatoren, da sie bei Umkehr der Motorlaufrichtung zerstört würden. Ungepolte Kondensatoren verringern Spannungsspitzen, die über einen längeren Zeitraum anhalten, z. B. beim Anlaufen des Motors oder bei Umkehr der Motorlaufrichtung.

Spannungsspitzen lassen sich auch mit zusätzlichen Widerständen reduzieren, die mit jedem der Motoranschlüsse in Serie geschaltet werden. Diese Methode funktioniert bei entsprechend hohen Widerständen garantiert; jedoch leidet die Motorleistung darunter.

Schließlich kann es auch hilfreich sein, die Batterien durch andere auszutauschen, die hohe Stoßströme liefern können. Probieren Sie für die Spannunsversorgung der Schaltlogik und/oder der Motoren anstelle der Alkalizellen Blei/Säure-Batterien oder NiCd-Akkus aus.

Sensor führt keinen Selbsttest durch

Prüfen Sie die Lötverbindungen auf der Platine. Lötbrücken, (unbeabsichtigte Verbindungen zwischen nebeneinanderliegenden Leiterbahnen oder an den Kontaktflächen) und kalte Lötstellen können Ursache für eine fehlerhafte Schaltung sein.

Infrarotsystem arbeitet nicht oder Reichweite ist zu gering

Prüfen Sie, ob die IR-Sendedioden richtig in Ihren Fassungen sitzen. Bei falscher Polung der Dioden entsteht kein Schaden. Prüfen Sie durch Verändern des Potentiometers (Aufschrift 5 K POT) auf der Platine, ob der 40 kHz-Oszillator richtig eingestellt ist. Überprüfen Sie die Chip-Fassung des 78HC04 und den 40 kHz-Oszillator auf brüchige Lötstellen.

Rad-Encoder funktionieren nicht
Das IR-Sendedioden/Detektor-Paar des Rad-Encodersystems, der Photoreflektor, muß so positioniert sein, daß er auf das an den Rädern angebrachte Streifenmuster zeigt. Die Leistung hängt stark von dem Abstand zwischen dem Streifenmuster und dem Photoreflektor ab. Idealerweise sollte der Abstand drei Millimeter betragen.

Das vom Photoreflektor ausgegebene Signal ändert sich mit jedem Farbwechsel auf der Scheibe. Sollte sich das Signal nicht ändern, ist der Kontrast zwischen den reflektierenden und nicht reflektierenden Streifen nicht ausgeprägt genug. Wenn das Streifenmuster beim Einbau Knicke erhalten hat, versuchen Sie, die Scheibe zu glätten. Sind die schwarzen Streifen verblaßt, färben Sie diese mit einem Stift nach.

Anschlußrichtung der LCD-Anzeige ist nicht eindeutig
Stecken Sie die LCD-Anzeige wie in Abbildung 1.1 gezeigt auf die Platine. Beachten Sie dabei, daß der Pin `PC7` (mit schwarzer Umrandung) des Rug Warrior Bildschirmsteckers auf den Pin `14` der LCD-Platine gehört.

LCD-Anzeige funktioniert nicht
Wird das IC-Programm gestartet, gibt es eine Meldung auf die LCD-Anzeige aus. Erscheint diese Meldung nicht, gibt es hierfür mehrere Ursachen. Prüfen Sie zunächst, ob das Programm tatsächlich läuft. Schieben Sie den EIN/AUS-Schalter auf die Position »RUN« (Ausführen), und drücken Sie die RESET-Taste. Sie sollten einen Ton vom Piezo-Summer hören und die Platine ansprechen können. Wenn Sie z. B. `1␣+␣1;` eingeben, sollten Sie folgende Meldung auf Ihrem Computerbildschirm sehen:

```
Downloading 7 bytes (addresses C200-C206):   7 loaded
Returned <int> 2
```

Wenn Sie nach dem RESET keinen Ton vom Piezo-Summer hören oder die Platine nicht ansprechen können, wird das IC-Programm eventuell nicht ausgeführt. Sehen Sie zur Fehlerbeseitigung im entsprechenden Abschnitt nach.

Wenn auf der LCD-Anzeige nichts zu sehen ist, obwohl das Programm ausgeführt wird, verändern Sie die Bildschirmeinstellung am LCD-Poti (siehe Abbildung 1.1). Normalerweise ist der Bildschirm bei den beiden Extremeinstellung ganz weiß bzw. schwarz (dies sollte er auch im Download-Modus sein).

Überprüfen Sie, ob das Display richtig in der hervorstehenden 14-Pin Buchse auf der PC-Platine sitzt. Wenn der Bildschirm auch weiterhin nichts anzeigt, prüfen Sie die Spannung am `VR`-Pin (siehe Abbildung 5.2). Mit Hilfe des LCD-Potentiometers sollten Sie die Spannung des `VR`-Pin von 0 V auf etwa 5 V bis 6 V einpegeln können.

Kapitel 7

Interactive C-Handbuch

Interactive C (kurz IC) ist eine Abwandlung der Programmiersprache C, bestehend aus einem Compiler (mit interaktiver Kommandozeilen-Übersetzung und -Debugging) und einem Laufzeitmodul in Maschinensprache. In IC ist eine Teilmenge von C implementiert, die Kontrollstrukturen (`for`, `while`, `if`, `else`), lokale und globale Variablen, Felder, Zeiger, 16-Bit und 32-Bit Ganzzahlen und 32-bit Fließkommazahlen enthält.

IC übersetzt den Quelltext in Pseudo-Code für eine spezielle Stapelmaschine, anstatt direkt Maschinencode für einen bestimmten Prozessor zu erzeugen. Dieser Pseudo-Code (oder *P-Code*) wird dann von dem in Maschinensprache geschriebenen Laufzeitprogramm interpretiert. Durch diesen für einen Compiler ungewöhnlichen Designansatz bietet IC folgende Möglichkeiten:

Interpretierte Programmausführung
Sie ermöglicht eine Laufzeit-Fehlerüberprüfung und verhindert Systemabstürze. Zum Beispiel führt IC bei Zugriffen auf Felder eine Bereichsüberprüfung durch, um vor den Auswirkungen von Programmierfehlern zu schützen.

Einfaches Design
Es ist wesentlich einfacher einen Compiler für eine Stapelmaschine zu schreiben, als einen Compiler für einen typischen Mikroprozessor. Wenn IC an einen anderen Prozessor angepaßt werden soll, muß nur der P-Code-Interpreter und nicht der Compiler umgeschrieben werden, da der P-Code von IC maschinenunabhängig ist.

Kurzer Objektcode
Der Programmcode für eine Stapelmaschine hat in der Regel einen geringeren Umfang als eine Repräsentation des Programms in der Maschinensprache eines Mikroprozessors.

Multitasking
Da der Pseudo-Code vollständig stapelbasiert ist, wird der Zustand eines Prozesses ausschließlich durch seinen Stapel und seinen Programmzähler definiert. Deshalb

kann ein Taskwechsel einfach dadurch ausgeführt werden, daß ein neuer Stapelzeiger und ein neuer Programmzähler geladen wird. Dieser Taskwechsel wird nicht vom Compiler, sondern vom Laufzeitmodul durchgeführt.

Diese Vorteile beruhen darauf, daß der von IC erzeugte P-Code interpretiert wird. Während der Laufzeit geht dies allerdings auf Kosten der Geschwindigkeit. Trotzdem läßt sich IC nicht als langsam bezeichnen.

IC wurde unter Mitwirkung von Fred Martin von Randy Sargent entworfen und implementiert.

7.1 Einsatz von IC

Wenn IC in Verbindung mit einem 6811-System benutzt wird, können zur Laufzeit in der Befehlszeile hinter der Eingabeaufforderung »`C>`« C-Ausdrücke, Funktionsaufrufe und IC-Befehle eingegeben werden.

C-Ausdrücke müssen mit einem Semikolon abgeschlossen werden. Um z.B. den arithmetischen Ausdruck $1 + 2$ auszuwerten, geben Sie beispielsweise folgendes ein:

```
C> 1 + 2;
```

Nach der Eingabe wird der Ausdruck vom Konsolencomputer kompiliert und dann zur Auswertung in das 6811-System geladen. Dieses wertet die kompilierte Form des Ausdrucks vom 6811 aus; anschließend wird das Ergebnis zum Konsolencomputer (z.B. PC oder Macintosh) zurückgesandt und auf dem Bildschirm ausgegeben.

Um eine Reihe von mehreren Ausdrücken gemeinsam auszuwerten, können Sie einen C-Block erzeugen, indem Sie die Eingabe mit einer öffnenden geschweiften Klammer »`{`« beginnen und mit einer schließenden geschweiften Klammer »`}`« beenden. Das folgenden Beispiel erzeugt eine lokale Variable `i` und gibt die Summe `i+7` auf der 6811-LCD-Anzeige aus:

```
C> {int i=3; printf("%d", i+7);}
```

7.1.1 IC-Befehle

IC reagiert auf die folgenden Befehle:

Datei laden
Der Befehl `load␣`*<filename>* übersetzt die angegebene Datei und lädt sie auf die Prozessorplatine. Hierfür muß die Platine angeschlossen sein. IC sucht Dateien zunächst im lokalen Verzeichnis und dann unter dem IC-Bibliothekspfad.

Es können mehrere Dateien gleichzeitig geladen werden, so daß Programme aus mehreren Dateien bestehen können.

Datei entfernen
Der Befehl `unload␣`*<filename>* macht das Laden einer Datei rückgängig, indem die restlichen geladenen Dateien erneut auf die Prozessorplatine geladen werden.

Dateien, Funktionen oder globale Variablen auflisten
Der Befehl `list␣files` zeigt die Namen aller zur Zeit von IC geladenen Dateien. Der Befehl `list␣functions` zeigt die Namen aller definierten C-Funktionen. Der Befehl `list␣globals` zeigt die Namen aller definierten globalen Variablen.

Alle Prozesse beenden
Der Befehl `kill_all` beendet alle gerade laufenden Prozesse.

Prozeßstatus anzeigen
Der Befehl `ps` zeigt den Status aller gerade laufenden Prozesse an.

Eine Datei editieren
Der Befehl `edit␣`*<filename>* startet einen Systemeditor, mit dem eine Datei editiert werden kann. Dieser Befehl ist auf Singletasking-Betriebssystemen, wie z. B. MS-DOS, besonders nützlich.

Eine untergeordnete Shell starten
Wenn IC auf einem MS-DOS-System läuft, öffnet der Befehl `run` eine Shell, um MS-DOS-Funktionen auszuführen.

Hilfe
Der Befehl `help` zeigt eine Hilfeseite mit IC-Befehlen an.

IC beenden
Der Befehl `quit` beendet IC. Alternativ kann STRG-C verwendet werden.

7.1.2 Editieren der Eingabezeile

IC verfügt über einen eingebauten Zeileneditor und einen Befehlspuffer, so daß früher eingegebene Anweisungen und Befehle editiert und wiederverwendet werden können. Die Kürzel für diese Funktionen entsprechen der Emacs-Standardtastenbelegung.

Um vorwärts und rückwärts durch die Liste der im Befehlspuffer gespeicherten Kommandos zu wandern, tippen Sie STRG-P oder $\uparrow$ für rückwärts und STRG-N bzw. $\downarrow$ für vorwärts.

Eine früher eingegebene Zeile aus dem Befehlspuffer kann zurückgeholt werden, indem ein Ausrufezeichen gefolgt von den ersten paar Zeichen der gewünschten Zeile und abschließend einem Leerzeichen eingetippt wird.

Abbildung 7.1 zeigt die von IC verwendete Tastenzuordnung. Bei der Eingabe von Ausdrücken hebt IC in Klammern eingefaßte Textteile hervor.

7.1.3 Die Funktion `main()`

Nachdem Funktionen auf die Platine geladen wurden, können sie von der IC-Kommandozeile aus gestartet werden. Wenn eine der Funktionen den Namen `main()` trägt, wird sie automatisch gestartet, sobald auf der Platine ein Reset ausgelöst wird.

Taste	Funktion	Taste	Funktion
ENTF	Zeichen rückwärts löschen	STRG-F	ein Zeichen vorwärts
STRG-A	Zeilenanfang	→	ein Zeichen vorwärts
STRG-B	ein Zeichen links	STRG-K	Zeile löschen
←	ein Zeichen links	STRG-U	Universal-Argument
STRG-D	Zeichen löschen	ESC D	Wort löschen
STRG-E	Zeilenende	ESC ENTF	Wort rückwärts löschen

Abbildung 7.1: Tastaturbelegung in der IC-Kommandozeile

7.2 Eine kurze Einführung in C

Die meisten C-Programme bestehen aus Funktionsdefinitionen und Datenstrukturen. Hier ist ein Beispiel für ein einfaches C-Programm, das nur eine einzige Funktion `main` definiert.

```
void main()
{
   printf("Hallo, Welt!\n");
}
```

Jede Funktion muß einen Ergebniswert haben. Das ist der Wert, den die Funktion nach ihrer Ausführung zurücksendet. Der Ergebniswert von `main` ist vom Typ `void`, dem »leeren« Typ. Andere Datentypen sind ganze Zahlen (`int` für Integer) und Fließkommazahlen (`float`). Diese *Funktionsdeklarations*-Information muß am Anfang jeder Funktionsdefinition stehen.

Unmittelbar hinter der Funktionsdeklaration folgt der Name der Funktion (in diesem Fall `main`). Als nächstes werden in Klammern die Argumente (oder Eingaben) der Funktion angegeben. `main` hat zwar keine Argumente; trotzdem muß ein leeres Klammerpaar angegeben werden.

Hinter den Argumenten steht eine öffnende geschweifte Klammer `{`. Dies kennzeichnet den Anfang des eigentlichen Funktionscodes. Mit geschweiften Klammern werden Codestücke zu *Blöcken* zusammengefaßt.

Als nächstes folgt eine Reihe von C-*Anweisungen*. Durch Anweisungen werden verschiedene Aktionen ausgelöst. Unser Beispielprogramm enthält nur eine einzige Anweisung, ein `printf` (formatted print = formatierte Ausgabe). Dadurch wird die Nachricht »`Hallo,␣Welt!`« auf der LCD-Anzeige angezeigt. Das `\n` kennzeichnet das Zeilenende.

Die `printf`-Anweisung endet mit einem Semikolon (`;`). *Alle* C-Anweisungen müssen mit einem Semikolon abgeschlossen werden. Neulinge in der C-Programmierung machen häufig den Fehler, das Semikolon wegzulassen, das am Ende jeder Anweisung stehen muß.

Die Funktion `main` endet mit der schließenden geschweiften Klammer `}`.

Lassen Sie uns einen Blick auf ein weiteres Beispiel werfen, um mehr nützliche Eigenschaften von C kennenzulernen. Der folgende Code definiert die Funktion `square`, die als Ergebnis das mathematische Quadrat einer Zahl liefert.

```
int square(int n)
{
   return n * n;
}
```

Aus ihrer Deklaration geht hervor, daß die Funktion vom Typ `int` ist, d. h. sie liefert eine ganze Zahl als Ergebnis. Ihr folgt der Funktionsname `square` und die in Klammern eingefaßte Argumentenliste. `square` hat ein Argument `n` vom Typ `int`. Wie hier zu sehen ist, wird der Typ eines Argumentes ähnlich deklariert wie der Funktionstyp.

Wenn für eine Funktion Argumente deklariert sind, dann sind diese Argumentvariablen innerhalb des »Bezugsrahmens« (oder »Scope«) der Funktion gültig, d. h. sie haben nur Bedeutung für den funktionseigenen Code. Andere Funktionen können unabhängig davon dieselben Variablennamen verwenden.

Der Code der Funktion `square` steht zwischen dem geschweiften Klammerpaar. Im nachfolgenden Beispiel besteht er aus der einzelnen Anweisung `return`. Die `return`-Anweisung beendet die Funktion und liefert als Funktionsergebnis den Wert des C-*Ausdrucks*, der auf das `return` folgt (in diesem Fall »`n␣*␣n`«).

Ausdrücke werden nach einigen Prioritätsregeln ausgewertet, die von den verschiedenen Operationen innerhalb des Ausdrucks abhängen. Im Beispiel wird nur eine Operation ausgeführt (Multiplikation), ausgedrückt durch »`*`«, so daß hier keine Prioritätsregeln angewendet werden (müssen).

Im folgenden Beispiel wird die `square`-Funktion verwendet:

```
float hypotenuse(int a, int b)
{
   float h;
   h = sqrt((float)(square(a) + square(b)));
   return h;
}
```

Dieser Programmcode demonstriert einige weitere Eigenschaften von C. Zunächst wird am Anfang der Funktion `hypotenuse` eine Variable `h` für Fließkommazahlen definiert. Im Prinzip können mit jedem neuen Programmblock (gekennzeichnet durch ein Paar geschweifter Klammern) neue lokale Variablen definiert werden.

Der Wert von `h` wird auf das Ergebnis eines Aufrufs der `sqrt`-Funktion gesetzt. `sqrt` (Wurzel) ist eine eingebaute Funktion, die als Argument eine Fließkommazahl erhält.

Eigentlich soll die bereits beschriebene Funktion `square` verwendet werden. Diese Funktion liefert als Ergebnis eine ganze Zahl, aber die `sqrt`-Funktion benötigt eine Fließkommazahl als Argument. Um diesen Typenkonflikt zu umgehen, wird für die ganzzahlige Summe `(square(a)␣+␣square(b))` eine Umwandlung in eine Fließkommazahl erzwungen, indem der gewünschte Typ in Klammern davor geschrieben wird. Die in eine Fließkommazahl umgewandelte (ganzzahlige) Summe wird an `sqrt` übergeben.

Die Funktion `hypotenuse` wird mit der `return`-Anweisung beendet. Als Ergebnis wird der Wert von `h` ausgegeben.

Damit ist die Kurzeinführung in C beendet.

7.3 Datentypen, Operatoren und Ausdrücke

Die grundlegenden Datenobjekte in einem C-Programm sind Variablen und Konstanten. In Deklarationen werden die zu verwendenden Variablen aufgelistet und deren Datentyp angegeben. Zusätzlich kann ihnen ein Anfangswert zugewiesen werden. Operatoren geben an, was mit den Variablen geschehen soll. In Ausdrücken werden Variablen und Konstanten kombiniert, um neue Werte zu erzeugen.

7.3.1 Variablennamen

In Variablennamen wird zwischen Groß- und Kleinschreibung unterschieden. Außerdem ist der Unterstrich »`_`« zugelassen; er wird oft benutzt, um lange Variablennamen lesbarer zu machen. C-Schlüsselwörter, wie `if`, `while`, usw. dürfen als Variablennamen nicht verwendet werden.

Globale Variablen und Funktionen dürfen nicht denselben Namen haben. Außerdem können Funktionen innerhalb des Bezugsrahmens einer lokalen Variable nicht benutzt werden, wenn sie denselben Namen wie die Variable haben.

7.3.2 Datentypen

IC kennt die folgenden Datentypen:

16-Bit-Ganzzahlen
16-Bit-Ganzzahlen werden durch den Typbezeichner `int` gekennzeichnet (für Integer). Integer sind vorzeichenbehaftete ganze Zahlen zwischen $-32\,768$ und $+32\,767$ (dezimal).

32-Bit-Ganzzahlen
32-Bit-Ganzzahlen werden durch den Typbezeichner `long` gekennzeichnet (für long integer). Long Integer sind vorzeichenbehaftete ganze Zahlen im Bereich von $-2\,147\,483\,648$ bis $+2\,147\,483\,647$ (dezimal).

32-Bit-Fließkommazahlen
Fließkommazahlen werden durch den Typbezeichner `float` gekennzeichnet. Sie haben eine Genauigkeit von annähernd sieben Dezimalstellen, und ihr Wert kann im Bereich von ungefähr 10^{-38} bis 10^{38} liegen.

8-Bit-Zeichen
»Zeichen« sind 8-Bit-Ganzzahlen, die durch den Typbezeichner `char` gekennzeichnet sind. Der Wert eines »Zeichens« repräsentiert normalerweise ein darstellbares Symbol, wobei der ASCII-Code zugrundegelegt wird.

Von IC werden nur Felder von Zeichen (Zeichenketten) unterstützt, keine einzelnen Zeichen.

7.3.3 Lokale und globale Variablen

Wenn eine Variable innerhalb einer Funktion oder als Argument einer Funktion deklariert wird, ist sie *lokal* gebunden. Dies bedeutet, daß die Variable nur innerhalb dieser Funktionsdefinition existiert.

Eine außerhalb einer Funktion deklarierte Variable wird als globale Variable bezeichnet. Sie ist für alle Funktionen definiert, einschließlich der Funktionen, die in anderen Dateien definiert sind.

Initialisierung von Variablen

Lokale und globale Variablen können bei ihrer Deklaration initialisiert werden. Wenn kein Initialisierungswert angegeben ist, wird die Variable mit Null initialisiert.

```
int foo()
{
   int x;       /* lokale Variable x mit Anfangswert 0 erzeugen */
   int y=7;     /* lokale Variable y mit Anfangswert 7 erzeugen */
   ...
}
float z=3.0;    /* globale Variable z mit Anfangswert 3.0 erzeugen */
```

Lokale Variablen werden bei jeder Ausführung der Funktion, in der sie enthalten sind, neu initialisiert.

Globale Variablen bei jeder Neustart-Bedingung initialiert. Folgende Umstände stellen eine Neustart-Bedingung dar:

- Neuer Programmcode wird hinuntergeladen.
- Die Funktion `main()` wird aufgerufen.
- Ein Hardware-Reset tritt auf.

Persistente globale Variablen

IC verfügt über eine besondere, *nicht initialisierte* Form globaler Variablen, die als »persistent« bezeichnet werden. Eine persistente Globale wird *nicht* nach den Regeln für normale globale Variablen initialisiert.

Um eine persistente globale Variable anzulegen, wird vor dem Typ-Spezifizierer das Schlüsselwort `persistent` angegeben. So erzeugt die Anweisung

```
persistent int i;
```

eine globale Variable `i` vom Typ `int`. Eine persistente Variable hat einen willkürlichen Anfangswert. Er hängt vom Inhalt des RAMs ab, das der Variablen zugeordnet wurde. In der Variablendeklaration einer persistenten Variable kann kein Anfangswert angegeben werden.

Persistente Variablen behalten ihren Wert, wenn der Roboter aus- und wieder eingeschaltet wird, wenn die Funktion `main` aufgerufen wird und wenn ein System-Reset auftritt. Persistente Variablen verlieren ihren Zustand nur, wenn ein neues Programm auf die Platine geladen wird. Dies passiert jedoch normalerweise nur, wenn der Anwender es will. Wenn persistente Variablen am Anfang des Codes vor allen Funktionen und nicht-persistenten Globalen deklariert werden, bekommen sie nach dem Neukompilieren des Codes denselben Speicherplatz wieder zugeordnet, so daß ihr Wert auch nach mehrfachem Laden auf die Platine erhalten bleibt.

Ist das Programm in mehrere Dateien aufgeteilt, und alle Dateien sollen dieselben persistenten Variablen verwenden, ist es empfehlenswert, alle persistenten Variablen in derselben Datei zu deklarieren, die von allen Dateien als erste geladen werden sollte.

Persistente Variablen wurden aus zwei Gründen eingeführt:

1. Einige Kalibrierungs- und Konfigurierungswerte müssen nicht bei jeder Neustart-Bedingung neu berechnet werden.
2. Ein Lernalgorithmus für Roboter kann sich über einen Zeitraum erstrecken, in dem der Roboter aus- und eingeschaltet wird.

7.3.4 Konstante

Ganze Zahlen

Ganze Zahlen (Integers) können in Dezimalschreibweise (z. B. `4053` oder `-1`), Hexadezimalschreibweise unter Verwendung des Präfix »`0x`« (z. B. `0x1fff`) oder in der nicht standardmäßigen aber nützlichen Binär-Schreibweise unter Verwendung des Präfix »`0b`« (z. B. `0b1001001`) definiert werden.

Mit einem Null-Präfix versehene oktale Konstanten werden nicht unterstützt.

Lange Ganzzahlen

Lange Ganzzahl-Konstanten werden erzeugt, indem das Suffix »`l`« oder »`L`« (kleiner oder großer Buchstabe »L«) an eine dezimale Ganzzahl angehängt wird. `0L` definiert beispielsweise die lange Ganzzahl Null. Es kann wahlweise das kleine oder große »L« verwendet werden; aus Gründen der besseren Lesbarkeit wird jedoch üblicherweise der Großbuchstabe benutzt.

Fließkommazahlen
Bei der Angabe von Fließkommazahlen kann entweder die exponentielle Schreibweise verwendet werden (z.B. `10e3` oder `10E3`) oder aber eine mit einem Dezimalpunkt gekennzeichnete Zahl. Die Fließkommazahl Null kann z.B. als `0.`, `0.0` oder `0E1` angegeben werden, aber nicht einfach als `0`.

Zeichen und Zeichenketten
In einfachen Anführungszeichen angegebene Zeichen liefern ihren ASCII-Wert (z.B. `'x'`). Zeichenketten werden in doppelte Anführungszeichen eingefaßt (z.B. `"Dies␣ist␣eine␣Zeichenkette."`).

7.3.5 Operatoren

Jeder Datentyp verfügt über einige spezifische Operatoren, die angeben, welche Operationen auf den Typ angewendet werden können.

Ganze Zahlen
Die folgenden Operationen werden für ganze Zahlen unterstützt:

Arithmetik
»Addition« `+`, »Subtraktion« `-`, »Multiplikation« `*` und »Division« `/`.

Vergleich
»Größer als« `>`, »kleiner als« `<`, »gleich« `==`, »ungleich« `!=`, »größer oder gleich« `>=` und »kleiner oder gleich« `<=`.

Bitweise Arithmetik
»bitweises Oder« `|`, »bitweises Und« `&`, »bitweises Exklusiv-Oder« `^` und »bitweises Nicht« ~.

Boolesche Arithmetik
»logisches Oder« `||`, »logisches Und« `&&` und »logisches Nicht« `!`.

Wenn in einer C-Anweisung ein boolescher Wert erwartet wird (z.B. in der `if`-Anweisung), hat der Ganzzahl-Wert Null die Bedeutung »falsch«; jeder andere Ganzahl-Wert bedeutet »wahr«. Die booleschen Operatoren liefern Null für »falsch« und Eins für »wahr«.

Die booleschen Operatoren `&&` und `||` beenden ihre Ausführung, sobald der Wert des Ausdrucks feststeht. Wenn zum Beispiel `a` den Wert »falsch« hat, wird im Ausdruck `a␣&&␣b` der Wert von `b` nicht ermittelt, da nach Auswertung von `a` bereits feststeht, daß der Ausdruck den Wert »falsch« hat. Der Operator `&&` »weiß« dies und wertet `b` deshalb nicht aus.

Lange Ganzzahlen
Ein Teil der für ganze Zahlen implementierten Operatoren ist auch für lange Ganzzahlen definiert: arithmetische Addition `+`, Subtraktion `-`, Multiplikation `*` und die Vergleichsoperatoren. Bitweise und Boolesche Operationen sowie die Division werden nicht unterstützt.

Fließkommazahlen
IC verwendet ein Paket von Public-Domain Fließkommaroutinen, die von Motorola vertrieben werden. In diesem Paket sind arithmetische, trigonometrische und logarithmische Funktionen enthalten. Die folgenden Operationen werden für Fließkommazahlen unterstützt:

Arithmetik
»Addition« `+`, »Subtraktion« `-`, »Multiplikation« `*` und »Division« `/`.

Vergleich
»Größer als« `>`, »kleiner als« `<`, »gleich« `==`, »ungleich« `!=`, »größer oder gleich« `>=` und »kleiner oder gleich« `<=`.

Eingebaute mathematische Funktionen
Es wird eine Menge von trigonometrischen, logarithmischen und Exponentialfunktionen unterstützt. Diese werden in Abschnitt 7.8 näher beschrieben.

Zeichen
Zeichen dürfen nur als Bestandteil von Zeichenketten vorkommen. Wird auf eine Zelle eines solchen Feldes zugegriffen, wird der Wert automatisch in seine Ganzzahldarstellung umgewandelt, so daß er über die normalen Ganzzahloperationen manipuliert werden kann. Wird ein Wert in einem Feld von Zeichen gespeichert, wird er von der Standard-16-Bit-Ganzzahl in ein 8-Bit-Zeichen umgewandelt (indem die oberen acht Bit abgeschnitten werden).

7.3.6 Zuweisungsoperatoren und Ausdrücke

Der grundlegende Zuweisungsoperator ist `=`. In der folgenden Anweisung wird auf den Wert von `a` die Zahl 2 addiert.

```
a = a + 2;
```

Dieselbe Operation wird durch die Kurzform

```
a += 2;
```

ausgeführt. Die folgenden binären Operatoren können alle auf diese Art verwendet werden:

```
+   -   *   /   %   <<   >>   &   ^   |
```

7.3.7 Inkrement- und Dekrement-Operatoren

Der Inkrement-Operator »`++`« erhöht den Wert der angegebenen Variable um 1. Die Anweisung »`a++`« hat z.B. dieselbe Wirkung, wie die Anweisungen »`a␣=␣a+1`« und »`a␣+=␣1`«.

Eine Anweisung, die den Inkrement-Operator verwendet, hat einen Wert. Beispielsweise geben die Anweisungen

```
a = 3;
printf("a=%d a+1=%d\n", a, ++a);
```

den Text »`a=3␣a+1=4`« aus.

Wenn der Inkrement-Operator vor der angegebenen Variable steht, wird der Wert der Anweisung berechnet, *nachdem* die Inkrementierung durchgeführt wurde. Die Anweisung

```
a = 3;
printf("a=%d a+1=%d\n", a, a++);
```

würde also den Text »`a=3␣a+1=3`« ausgeben; aber anschließend hätte die Variable `a` den Wert 4.

Der Dekrement-Operator »`--`« wird in derselben Weise benutzt wie der Inkrement-Operator.

7.3.8 Priorität und Reihenfolge der Auswertung

Die Tabelle in Abbildung 7.2 faßt die Regeln für Prioritäten und Assoziativität der C-Operatoren zusammen. Operatoren, die in der Tabelle weiter oben stehen, haben eine höhere Priorität. Operatoren in derselben Zeile haben die gleiche Priorität.

7.4 Kontrollfluß

Die meisten für Standard-C definierten Kontrollstrukturen können auch mit IC verwendet werden. Eine wichtige Ausnahme stellen die `switch`- und `case`-Anweisungen dar. Sie werden nicht unterstützt.

7.4.1 Anweisungen und Blöcke

Jede separate C-Anweisung wird mit einem Semikolon abgeschlossen. Mehrere Anweisungen können durch geschweifte Klammern zu einem *Block* zusammengefaßt werden. Innerhalb eines Blockes können lokale Variablen definiert werden. Auf eine schließende geschweifte Klammer folgt kein Semikolon.

<table>
<tr><th>Operator</th><th>Assoziativität</th></tr>
<tr><td>() []</td><td>von links nach rechts</td></tr>
<tr><td>! ~ ++ -- - (type)</td><td>von rechts nach links</td></tr>
<tr><td>* / %</td><td>von links nach rechts</td></tr>
<tr><td>+ -</td><td>von links nach rechts</td></tr>
<tr><td><< >></td><td>von links nach rechts</td></tr>
<tr><td>< <= > >=</td><td>von links nach rechts</td></tr>
<tr><td>== !=</td><td>von links nach rechts</td></tr>
<tr><td>&</td><td>von links nach rechts</td></tr>
<tr><td>^</td><td>von links nach rechts</td></tr>
<tr><td>|</td><td>von links nach rechts</td></tr>
<tr><td>&&</td><td>von links nach rechts</td></tr>
<tr><td>||</td><td>von rechts nach links</td></tr>
<tr><td>= += -= usw.</td><td>von rechts nach links</td></tr>
<tr><td>,</td><td>von links nach rechts</td></tr>
</table>

Abbildung 7.2: Regeln für Priorität und Assoziativität der C-Operatoren.

7.4.2 Die `if-else`-Anweisung

Die `if-else`-Anweisung wird benutzt, um Verzweigungen zu definieren. Ihre Syntax lautet:

`if (` *Ausdruck* `)`
    Anweisung-1
`else`
    Anweisung-2

Ausdruck wird ausgewertet. Ist der *Ausdruck* ungleich Null, (d. h. logisch wahr), dann wird die *Anweisung-1* ausgeführt. Der `else`-Fall ist optional. Wenn der `if`-Teil der Anweisung nicht ausgeführt wird, und die Anweisung einen `else`-Teil enthält, wird *Anweisung-2* ausgeführt.

7.4.3 Die `while`-Schleife

Die `while`-Schleife hat folgende Syntax:

`while (` *Ausdruck* `)`
    Anweisung

Zunächst wird der *Ausdruck* ausgewertet. Wenn der *Ausdruck* den Wert Falsch hat, dann wird die *Anweisung* übersprungen. Wenn der *Ausdruck* den Wert Wahr hat, dann wird die *Anweisung* ausgeführt. Anschließend wird der *Ausdruck* wieder ausgewertet und derselbe Test durchgeführt. Die Schleife endet, wenn der *Ausdruck* den Wert 0 liefert.

Es ist einfach, in C eine Endlosschleife zu programmieren, indem man die `while`-Anweisung verwendet:

```
while (1)
   Anweisung
```

7.4.4 Die `for`-Schleife

Die `for`-Schleife hat folgende Syntax:

```
for ( Ausdr-1 ; Ausdr-2 ; Ausdr-3 )
  Anweisung
```

Dies läßt sich unter Verwendung der `while`-Schleife so ausdrücken:

```
Ausdr-1 ;
while ( Ausdr-2 )
{
   Anweisung
   Ausdr-3 ;
}
```

In der Regel ist *Ausdr-1* eine Zuweisung, *Ausdr-2* ist ein Vergleich, und *Ausdr-3* ist eine Inkrementierung oder Dekrementierung. Der Code im folgenden Beispiel zählt von 0 bis 99 und gibt dabei jede Zahl aus:

```
int i;
for (i = 0; i < 100; i++)
   printf("%d\n", i);
```

7.4.5 Die `break`-Anweisung

Die `break`-Anweisung kann verwendet werden, um eine `while`- oder `for`-Schleife vorzeitig zu verlassen.

7.5 Ausgabe auf der LCD-Anzeige

IC verfügt über eine Fassung der C-Funktion `printf` für formatierte Ausgabe auf der LCD-Anzeige. Die `printf`-Anweisung hat folgende Syntax:

```
printf( Format-Zeichenkette , [Arg-1] , ... , [Arg-n] )
```

Dies soll anhand einiger Beispiele näher erläutert werden.

7.5.1 Beispiele

Beispiel 1: Eine Nachricht ausgeben

Die folgende Anweisung gibt eine Zeichenkette auf dem Bildschirm aus.

```
printf("Hallo, Welt!\n");
```

In diesem Beispiel wird die Format-Zeichenkette einfach auf dem Bildschirm ausgegeben. Das »`\n`« am Ende der Zeichenkette kennzeichnet das Zeilenende (end of line). Bei Ausgabe eines Zeilenende-Zeichens wird die LCD-Anzeige gelöscht, wenn das nächste Zeichen ausgegeben wird. Deshalb werden die meisten `printf`-Anweisungen mit einem `\n` abgeschlossen.

Beispiel 2: Eine Zahl ausgeben

Die folgende Anweisung gibt in einer kurzen Nachricht den Wert der Ganzzahl-Variablen `x` aus.

```
printf("Der Wert ist %d\n", x);
```

Hier gibt das `%d` in der Format-Zeichenkette an, daß eine ganze Zahl in Dezimal-Schreibweise ausgegeben werden soll.

Beispiel 3: Eine Zahl in Binär-Schreibweise ausgeben

Die folgende Anweisung gibt den Wert der Ganzzahl-Variablen `x` als Binärzahl aus.

```
printf("Der Wert ist %b\n", x);
```

Das `%b` in diesem Beispiel wird verwendet, um eine ganze Zahl in Binär-Schreibweise auszugeben. Nur das untere Byte der Zahl wird ausgegeben.

Beispiel 4: Eine Fließkommazahl ausgeben

Die folgende Anweisung gibt den Wert der Fließkomma-Variable `n` als Fließkommazahl aus.

```
printf("Der Wert ist %f\n", n);
```

Das `%f` bewirkt, daß eine Fließkommazahl in ihrer Fließkomma-Schreibweise ausgegeben wird.

Beispiel 5: Zwei Zahlen in Hexadezimal-Schreibweise ausgeben

```
printf("A=%x  B=%x\n", a, b);
```

Durch das `%x` wird eine ganze Zahl in Hexadezimal-Schreibweise ausgegeben.

7.5.2 Zusammenfassung der Formatierungsbefehle

Formatierungsbefehl	**Datentyp**	**Beschreibung**
`%d`	`int`	Dezimalzahl
`%x`	`int`	Hexadezimalzahl
`%b`	`int`	unteres Byte als Binärzahl
`%c`	`int`	unteres Byte als ASCII-Zeichen
`%f`	`float`	Fließkommazahl
`%s`	`char[]`	Zeichenkette

7.5.3 Anmerkungen

- ◇ Das letzte Zeichen auf der LCD-Anzeige wird als Indiz für das Funktionieren des Systems benutzt. Solange die Platine ordnungsgemäß arbeitet, blinkt das Zeichen ständig. Wenn das Zeichen nicht mehr blinkt, wurde der Prozessor angehalten.
- ◇ Zeichen, die jenseits des letzten Zeichens ausgegeben würden, werden abgeschnitten.
- ◇ Wenn eine zweizeilige Anzeige verwendet wird, betrachtet die `printf()`-Anweisung die Anzeige als eine einzelne längere Zeile.
- ◇ Die Ausgabe langer Ganzzahlen ist bisher nicht möglich.

7.6 Felder und Zeiger

IC kennt eindimensionale Felder von Zeichen, ganzen Zahlen, langen Ganzzahlen und Fließkommazahlen. Auch Zeiger auf Datenobjekte und Felder werden unterstützt.

7.6.1 Felder deklarieren und initialisieren

In der Deklaration von Feldern werden eckige Klammern verwendet. Die folgende Anweisung deklariert ein Feld von zehn ganzen Zahlen:

```
int foo[10];
```

Die Elemente in diesem Feld sind von 0 bis 9 numeriert. Um auf ein Element zuzugreifen, wird der Index des Elementes in eckigen Klammern angegeben: `foo[4]` bezeichnet das fünfte Element des Feldes `foo` (da die Zählung mit Null beginnt).

Standardmäßig werden Felder so initialisiert, daß alle Elemente den Wert Null haben. Felder können aber auch mit anderen Werten initialisiert werden, die bei der Deklaration in geschweiften Klammern eingefaßt und durch Kommata getrennt angegeben werden. Bei Verwendung dieser Syntax wird innerhalb der eckigen Klammern keine Größe für das Feld angegeben. Sie wird durch die Anzahl der Elemente festgelegt, die in der Deklaration angegeben werden. So erzeugt

```
int foo[] = {0, 4, 5, -8, 17, 301};
```

beispielsweise ein Feld von sechs ganzen Zahlen, wobei `foo[0]` den Wert 0 erhält, `foo[1]` den Wert 4, usw.

Felder von Zeichen sind normalerweise Zeichenketten. Zum Initialisieren von Zeichen-Feldern existiert eine besondere Syntax. Die Werte der Zeichen des Feldes werden in doppelte Anführungszeichen eingefaßt:

```
char kette[] = "Hallo, dort draussen";
```

Dieses Beispiel erzeugt ein Feld von Zeichen mit dem Namen `kette`, das die ASCII-Werte der angegebenen Zeichen enthält. Außerdem wird das Ende des Feldes mit dem Wert 0 markiert. Mit dieser Null-Terminierung kann das Zeichen-Feld z. B. als eine Zeichenkette für die Ausgabe verwendet werden. Zeichen-Felder können auch über die Syntax mit den geschweiften Klammern initialisiert werden, aber dann werden sie nicht automatisch Null-terminiert. Die Ausgabe von Zeichen-Feldern *ohne* Null-Terminierung ist jedoch generell problematisch.

7.6.2 Felder als Argumente übergeben

Wenn ein Feld als Argument an eine Funktion übergeben wird, dann werden nicht die Elemente des Feldes übergeben, sondern nur ein Zeiger auf das Feld. Von dem Feld existiert also nur eine Kopie im Speicher, so daß die Funktion die im Feld gespeicherten Werte ändern kann.

In normalem C gibt es zwei Möglichkeiten, ein Feld-Argument zu deklarieren: als Feld oder als Zeiger. IC erlaubt nur eine Deklaration von Feld-Argumenten als Felder. Die folgende Funktion bekommt z. B. einen Index und ein Feld als Parameter und liefert das durch den Index spezifizierte Element des Feldes:

```
int hole_element(int index, int feld[])
{
   return feld[index];
}
```

Durch die eckigen Klammern wird das Argument `feld` als ein Feld von ganzen Zahlen deklariert.

Bei Übergabe einer Feld-Variable an eine Funktion müssen keine eckigen Klammern angegeben werden:

```
{
   int feld[10];
   hole_element(3, feld);
}
```

7.6.3 Deklaration von Zeiger-Variablen

Zeiger können an Funktionen übergeben werden, die dann den Wert der Variablen ändern, auf die der Zeiger zeigt. Auf diese Weise kann dieselbe Funktion benutzt werden, um verschiedene Variablen zu verändern, indem der Funktion verschiedene Zeiger übergeben werden.

Zeiger werden durch Angabe eines Sterns (*) deklariert. In dem Beispiel

```
int *zei;
float *ger;
```

wird `zei` als ein Zeiger auf eine ganze Zahl deklariert, und `ger` wird als Zeiger auf eine Fließkommazahl deklariert.

Um eine Zeiger-Variable auf eine andere Variable zeigen zu lassen, wird das Kaufmanns-Und (&) verwendet. Der &-Operator liefert die Adresse des Inhalts einer Variablen, also von dem Speicherplatz, in dem der Wert der Variable steht. Die folgende Deklaration

```
int *zei;
int x = 5;
foo = &x;
```

bewirkt z. B., daß der Zeiger `zei` auf den Wert von `x` »zeigt« (der 5 ist).

Dieser Zeiger kann anschließend mit Hilfe des Stern-Operators auf den Wert von `x` zugreifen. Dieser Vorgang wird als *dereferenzieren* bezeichnet. Der Zeiger (die Referenz) wird benutzt, um auf den Wert zuzugreifen, auf den der Zeiger zeigt. In dem Beispiel

```
int y;
y = *zei;
```

wird `y` auf den Wert gesetzt, auf den `zei` zeigt. Im vorhergehenden Beispiel wurde `zei` auf die Adresse von `x` gesetzt, das den Wert 5 hat. Das Dereferenzieren von `zei` ergibt demnach den Wert 5, `y` wird auf 5 gesetzt.

7.6.4 Zeiger als Argumente übergeben

Ein Zeiger kann an eine Funktion übergeben werden, die den Wert der Variablen ändert, auf die der Zeiger zeigt. Dies wird als *call-by-reference* (Aufruf mit Variablenreferenz) bezeichnet. Es wird die Referenz (oder der Zeiger) auf die Variable beim Aufruf an die Funktion übergeben. Dies steht im Gegensatz zu *call-by-value* (Aufruf mit Wert), der normalen Art, eine Funktion aufzurufen. Dabei wird der Wert einer Variable beim Aufruf an die Funktion übergeben.

In dem folgenden Beispiel wird eine Funktion `mittel_sensor` definiert, die als Parameter eine Port-Nummer und einen Zeiger auf eine Ganzzahl-Variable erhält. Die Funktion bildet den Mittelwert der von einem Sensor gelieferten Werte und speichert das Ergebnis in der Variable, auf die `ergebnis` zeigt. Hier wird das Funktionsargument über den Stern als Zeiger spezifiziert:

```
void mittel_sensor(int port, int *ergebnis)
{
   int sum = 0;
   int i;
   for (i = 0; i< 10; i++) sum += analog(port);
   *ergebnis = sum/10;
}
```

Die Funktion selbst ist als `void` deklariert. Sie braucht keinen Wert zurückliefern, da sie ihre Antwort stattdessen in der Variablen speichert, auf die der angegebene Zeiger zeigt. Die Zeiger-Variable wird in der letzten Zeile der Funktion benutzt. In dieser Anweisung wird die Antwort `sum/10` in dem Speicherplatz gespeichert, auf den `ergebnis` zeigt. Der Stern wird verwendet, um `ergebnis` zu dereferenzieren und den *Ort* des Speicherplatzes zu ermitteln.

7.7 Multitasking

7.7.1 Übersicht

Eine der mächtigsten Eigenschaften von IC ist seine Multitasking-Fähigkeit. Prozesse können dynamisch zur Laufzeit erzeugt oder beendet werden. Jede C-Funktion kann als eine separate Task abgespalten werden. Derselbe Code kann von mehreren Tasks gleichzeitig ausgeführt werden, aber jede Task benutzt ihre eigenen lokalen Variablen.

Prozesse verständigen sich über globale Variablen: Ein Prozeß kann eine Globale auf einen bestimmten Wert setzen, und ein anderer Prozeß kann den Wert dieser Globalen lesen.

Jedesmal, wenn ein Prozeß mit der Ausführung an der Reihe ist, wird er für eine bestimmte Anzahl *Ticks* von je einer Millisekunde Länge ausgeführt. Dieser Wert wird für jeden Prozeß bei seiner Erzeugung angegeben. Die Standardeinstellung für die Anzahl von Ticks ist fünf. Das heißt, ein Standardprozeß wird 5 Millisekunden lang ausgeführt, dann ist der nächste Prozeß an der Reihe. Über die verschiedenen Prozesse wird in einer *Prozeßtabelle* Buch geführt. Die Prozesse in der Tabelle werden der Reihe nach ausgeführt (für eine Dauer, die der Anzahl ihrer Ticks entspricht).

Jeder Prozeß verfügt über seinen eigenen *Stapel*. Der Stapel wird benutzt, um Argumente für Funktionsaufrufe, lokale Variablen und Rücksprungadressen von Funktionsaufrufen zu speichern. Die Größe dieses Stapels wird bei Erzeugung des Prozesses festgelegt. Die Standardgröße des Stapels beträgt 256 Byte.

Prozesse, die intensiven Gebrauch von Rekursion machen oder große lokale Felder benutzen, benötigen vermutlich einen größeren Stapel, als standardmäßig vorgesehen ist. Jeder Funktionsaufruf benötigt zwei Bytes auf dem Stapel (für die Rücksprungadresse) zuzüglich der Anzahl an Bytes für die Argumente der Funktion. Wenn die aufgerufene Funktion lokale Variablen erzeugt, benötigen diese ebenfalls Platz auf dem Stapel. Außerdem entstehen bei der Auswertung von C-Ausdrücken Werte, die vorübergehend auf dem Stapel abgelegt werden.

Es ist die Aufgabe des Programmierers festzustellen, ob ein bestimmter Prozeß mehr Platz auf dem Stapel benötigt, als standardmäßig vorgesehen ist. Wenn bekannt ist, daß ein Prozeß weniger Platz auf dem Stapel benötigt, als die Standardgröße vorsieht, kann er auch mit einem *kleineren* Stapel erzeugt werden, um Speicherplatz zu sparen.

Wenn ein Prozeß erzeugt wird, erhält er eine eindeutige *Prozeß-Identifizierungsnummer* (*Prozeß-ID* oder *pid*) zugeteilt. Diese Zahl kann verwendet werden, wenn der Prozeß beendet werden soll.

7.7.2 Neue Prozesse erzeugen

Mit der Funktion `start_process` können neue Prozesse erzeugt werden. Die Funktion `start_process` erhält ein zwingendes Argument – den Funktionsaufruf, der als Prozeß gestartet werden soll. Es gibt zwei optionale Argumente: Die Anzahl Ticks für den Prozeß und die gewünschte Stapelgröße. (Wird nur ein optionales Argument angegeben, wird es als Anzahl der Ticks interpretiert, und die Standard-Stapelgröße wird verwendet.) `start_process` hat folgende Syntax:

`int start_process(` *Funktionsaufruf(...)* `, [`*Ticks*`] , [`*Stapelgröße*`] )`

`start_process` liefert die Prozeß-ID, die dem neuen Prozeß zugeordnet wurde, als ganze Zahl zurück. Als *Funktionsaufruf* kann jeder korrekte Aufruf der verwendeten Funktion angegeben werden. Der folgende Code zeigt, wie die Funktion `main` einen Prozeß erzeugt:

```
void teste_sensor(int n)
{
   while (1)
      printf("Sensor %d liefert %d\n", n, digital(n));
}
void main()
{
   start_process(teste_sensor(2));
}
```

Wenn eine C-Funktion beendet wird, liefert sie normalerweise einen Ergebniswert oder den Wert »`void`«. Wenn eine Funktion beendet wird, die als eigener Prozeß gestartet wurde, »stirbt« die Funktion – der Ergebniswert geht verloren (wenn einer vorhanden war). (Das stört aber nicht, da ein Prozeß sein Ergebnis in einer globalen Variable ablegen kann.)

In obenstehendem Beispiel ist die Funktion `check_sensor` als Endlosschleife definiert, so daß sie weiterläuft, bis die Platine ein Reset erhält oder ein `kill_process` ausgeführt wird.

Um einen Prozeß mit einer von den Voreinstellungen abweichenden Zahl von Ticks oder Stapelgröße zu erzeugen, muß `start_process` lediglich mit optionalen Argumenten aufgerufen werden. Zum Beispiel erzeugt

```
start_process(teste_sensor(2), 1, 50);
```

einen `teste_sensor`-Prozeß, der bei jeder Aktivierung 1 Millisekunde läuft und einen 50 Bytes großen Stapel hat. (Für die oben gegebene Definition von `teste_sensor` ist ein kleiner Stapel ausreichend.)

7.7.3 Prozesse beenden

Um einen Prozeß zu beenden, kann die Funktion `kill_process` verwendet werden. Dazu wird sie mit der Prozeß-ID des zu beendenden Prozesses aufgerufen. `kill_process` hat folgende Syntax:

```
int kill_process(int pid)
```

Als Ergebnis liefert `kill_process` einen Wert, der angibt, ob der Prozeß erfolgreich beendet wurde. Wenn der Wert `0` ist, wurde der Prozeß beendet. Wenn der Wert `1` ist, wurde der Prozeß nicht gefunden.

Der folgende Code zeigt, wie der Prozeß `main` einen Prozeß `teste_sensor` startet und ihn eine Sekunde später wieder beendet:

```
void main()
{
  int pid;
  pid= start_process(teste_sensor(2));
  sleep(1.0);
  kill_process(pid);
}
```

7.7.4 Befehle zur Prozeßverwaltung

IC kennt zwei Befehle, welche die Prozeßverwaltung erleichtern. Diese Befehle funktionieren nur, wenn sie in der IC-Kommandozeile, d. h. am PC oder Macintosh, eingegeben werden. Sie sind jedoch keine C-Funktionen und können daher nicht im Programmcode benutzt werden.

`kill_all`
Beendet alle zur Zeit laufenden Prozesse.

`ps`
Gibt eine Liste mit dem Prozeß-Status aller Prozesse aus. Folgende Informationen werden angezeigt: Prozeß-ID, Status-Code, Programmzähler, Stapelzeiger, ursprünglicher Stapelzeiger, Anzahl der Ticks und der Name der Funktion, die zur Zeit ausgeführt wird.

7.7.5 Bibliotheksfunktionen zur Prozeßverwaltung

Die folgenden Funktionen sind in der Standard C-Bibliothek enthalten.

`void hog_processor()`
Gewährt dem derzeit laufenden Prozeß weitere 256 Millisekunden Ausführungszeit. Wenn diese Funktion regelmäßig aufgerufen wird, dann wird das System sich verklemmen, und es wird nur noch der Prozeß ausgeführt, der ständig `hog_processor()` aufruft. Eine solche Verklemmung kann nur durch einen System-Reset beendet werden. Diese Funktion sollte daher nur mit extremer Vorsicht eingesetzt werden. In einer Schleife sollte sie nur verwendet werden, wenn eine Verklemmung beabsichtigt ist.

`void defer()`
Bei Aufruf dieser Funktion wird der aktuelle Prozeß sofort zurückgestellt. Dies ist sinnvoll, wenn vorauszusehen ist, daß der Prozeß nichts zu tun bekommt, bevor er das nächste Mal an der Reihe ist. `defer()` ist eine in IC eingebaute Funktion.

7.8 Fließkommafunktionen

Neben der grundlegenden Fließkommaarithmetik (Addition, Subtraktion, Multiplikation und Division) und Fließkomma-Vergleichen stellt IC auch eine Reihe von Exponential- und Transzendentalfunktionen zur Verfügung:

`float sin(float winkel)`
Liefert den Sinus von `winkel`. Der Winkel und das Ergebnis sind in Radianten angegeben.

`float cos(float winkel)`
Liefert den Cosinus von `winkel`. Der Winkel und das Ergebnis sind in Radianten angegeben.

`float tan(float winkel)`
Liefert den Tangens von `winkel`. Der Winkel und das Ergebnis sind in Radianten angegeben.

`float atan(float winkel)`
Liefert den Arcustangens von `winkel`. Der Winkel und das Ergebnis sind in Radianten angegeben.

`float sqrt(float zahl)`
Liefert die Quadratwurzel von `zahl`.

`float log10(float zahl)`
Liefert den Logarithmus zur Basis 10 von `zahl`.

`float log(float zahl)`
Liefert den natürlichen Logarithmus von `zahl`.

`float exp10(float zahl)`
Liefert $10^{\texttt{zahl}}$ als Ergebnis.

`float exp(float zahl)`
Liefert $e^{\texttt{zahl}}$ als Ergebnis.

`(float) a ^ (float) b`
Liefert $\texttt{a}^{\texttt{b}}$ als Ergebnis.

7.9 Speicherzugriffsfunktionen

IC verfügt über einfache Funktionen zum direkten Auslesen und Ändern von Speicherinhalten. Diese Funktionen sollten mit Vorsicht eingesetzt werden, da es bei ihrer Verwendung leicht passieren kann, daß der Speicherinhalt durcheinander gerät und das System abstürzt. Die Funktionen werden nur selten benötigt. Die meisten Speicherzugriffe sollten über Felder oder globale Variablen durchgeführt werden.

`int peek(int ort)`
Liefert das Byte, das an der Adresse `ort` steht.

`int peekword(int ort)`
Liefert das 16-Bit-Wort, das an den Adressen `ort` und `ort`+1 steht. Die Speicherzelle `ort` enthält das höherwertige Byte, entsprechend dem 16-Bit-Adressierungsstandard des 6811.

`void poke(int ort, int byte)`
Speichert den 8-Bit-Wert `byte` unter der Speicheradresse `ort`.

`void pokeword(int ort, int wort)`
Speichert den 16-Bit-Wert `wort` unter den Speicheradressen `ort` und `ort` + 1.

`void bit_set(int ort, int maske)`
Setzt in der Speicherzelle mit der Adresse `ort` die Bits, die in `maske` gesetzt sind.

`void bit_clear(int ort, int maske)`
Löscht in der Speicherzelle mit der Adresse `ort` die Bits, die in `maske` gesetzt sind.

7.10 Fehlerbehandlung

Bei der Benutzung von IC können zwei Arten von Fehlern auftreten: *Compilerfehler* und *Laufzeitfehler*.

Fehlercode	Beschreibung
1	Kein Stapelplatz für `start_process()`
2	Kein Prozeßplatz frei
3	Feldzugriff außerhalb der Feldgrenzen
4	Stapelüberlauf in einem laufenden Prozeß
5	Operation mit einem ungültigen Zeiger
6	Fließkomma-Unterlauf
7	Fließkomma-Überlauf
8	Fließkomma-Division durch Null
9	Zahl zu klein oder zu groß für eine Umwandlung nach `int`
10	Quadratwurzel einer negativen Zahl
11	Tangens von 90 Grad
12	`log10` oder `log` einer negativen Zahl oder von Null
15	Fließkomma-Formatierungsfehler in `printf`
16	Ganzzahl-Division durch Null

Abbildung 7.3: Laufzeit-Fehlercodes und ihre Bedeutung

Compilerfehler treten beim Übersetzen der Quelltext-Datei auf. Sie kennzeichnen Fehler im C-Quelltext. Typische Compilerfehler resultieren aus einer fehlerhaften Syntax oder einem Konflikt zwischen inkompatiblen Datentypen.

Laufzeitfehler treten auf, wenn ein Programm auf dem Prozessor ausgeführt wird. Sie deuten auf Probleme mit einem gerade aktiven C-Programm. Ein einfaches Beispiel für einen Laufzeitfehler ist eine Division durch Null. Ein anderes Beispiel ist ein Stapelüberlauf, der z. B. auftreten kann, wenn eine rekursive Prozedur sich selbst zu oft aufruft.

Diese beiden Fehlertypen werden unterschiedlich behandelt, wie im folgenden beschrieben wird.

7.10.1 Compilerfehler

Wenn ein Compilerfehler auftritt, wird eine Fehlermeldung auf dem Bildschirm angezeigt. Bevor eine Datei zur Platine geladen wird, müssen alle Compilerfehler behoben werden.

7.10.2 Laufzeitfehler

Wenn ein Laufzeitfehler auftritt, wird eine Fehlermeldung auf der LCD-Anzeige angezeigt, welche die Fehlernummer angibt. Wenn der Fehler auftritt, während die Platine an den PC oder Macintosh angeschlossen ist, wird eine ausführlichere Fehlermeldung auf dem Terminal ausgegeben. Abbildung 7.3 zeigt eine Liste der Laufzeit-Fehlercodes.

7.11 Binäre Programme

Bei der Benutzung eines 6811-Assemblerprogrammes erlaubt IC die Verwendung von Maschinensprache-Programmen innerhalb der C-Umgebung. Maschinensprache-Programme können auf zwei Arten eingebunden werden:

1. Programme werden von C aus aufgerufen, als wären sie C-Funktionen.
2. Programme installieren sich selbst in die Interrupt-Struktur des 6811 und laufen ständig oder wenn sie von einem Hardware- oder Software-Interrupt gestartet werden.

Wenn ein Binärprogramm als Funktion arbeitet, ist die Schnittstelle zwischen C und dem Programm eingeschränkt: Ein Binärprogramm erhält eine ganze Zahl als Argument und liefert eine ganze Zahl als Ergebnis. Programme in Binärdateien können eine beliebige Anzahl von globalen Ganzzahl-Variablen deklarieren, die auch in der C-Umgebung verfügbar sind. Zusätzlich kann das Argument eines Binärprogramms als Zeiger auf eine C-Datenstruktur verwendet werden.

7.11.1 Die binäre Quelldatei

In der Assembler-Quelldatei (oder dem Assembler-Modul) werden besondere Schlüsselwörter verwendet, um dem Binärprogramm die folgenden Eigenschaften zu verleihen:

Einsprungpunkt
Für jedes Programm, das in der Binärdatei definiert ist, wird ein Einsprungpunkt festgelegt.

Einsprungpunkt für die Initialisierung
Jede Datei kann eine Routine haben, die bei Auftreten einer Neustart-Bedingung automatisch aufgerufen wird. (Die Neustart-Bedingungen werden in Abschnitt 7.3.3 erklärt, wo die Initialisierung globaler Variablen beschrieben ist.) Diese Initialisierungsroutine ist insbesondere für Programme sinnvoll, die als Interrupt-Routine dienen.

C-Variablendefinition
In einer Binärdatei kann eine beliebige Anzahl von zwei Byte großen C-Ganzzahl-Variablen deklariert werden. Wenn das Modul in IC geladen wird, werden diese Variablen als globale C-Variablen definiert.

Abbildung 7.4 zeigt ein Beispiel einer Quelldatei für ein IC-Binärprogramm, an dem die oben aufgeführten Eigenschaften erklärt werden sollen.

Die erste Anweisung in der Datei (»`ORG␣MAIN_START`«) deklariert den Anfang des Binärprogramms. Diese Zeile muß vor dem eigentlichen Code stehen.

Der Einsprungpunkt für einen Programmaufruf aus C heraus wird über die besondere Form deklariert, die mit dem Text `subroutine_` beginnt. In diesem Fall

```
/* Beispiel icb-Datei */
/* Anfang des Moduls und der Variablen */
ORG     MAIN_START
/* Programm, das als Ergebnis das Doppelte des als Parameter übergebenen */
/* Wertes liefert */
subroutine_double:
ASLD
RTS
/* Deklaration für die Variable "foo" */
variable_foo:
FDB     55
/* Programm, um die C-Variable "foo" zu setzen */
subroutine_set_foo:
STD     variable_foo
RTS
/* Programm, um die Variable "foo" zu lesen */
subroutine_get_foo:
LDD     variable_foo
RTS
/* Code, der bei Auftreten einer Neustart-Bedingung ausgeführt wird */
subroutine_initialize_module:
LDD     #69
STD     variable_foo
RTS
```

Abbildung 7.4: Beispiel eines Quelltextes für ein IC-Binärprogramm: `testicb.asm`.

lautet der Name des Binärprogramms `double`. Deshalb trägt die Marke den Namen `subroutine_double`. Wie dem Kommentar zu entnehmen ist, verdoppelt dieses Programm den Wert, der als Parameter übergeben wird.

Wenn das Binärprogramm von C aus aufgerufen wird, erhält es eine ganze Zahl als Argument. Dieses Argument wird im Register D des 6811 (dem »Doppelakkumulator«) gespeichert, bevor das Binärprogramm aufgerufen wird.

Das Programm `double` verdoppelt die Zahl im Register D. Die Anweisung `ASLD` (»Arithmetic Shift Left Double«) entspricht einer Multiplikation mit 2; hiermit wird also die Zahl im Register D verdoppelt.

Die `RTS`-Anweisung ist ein »Return from Subroutine«. Jedes Binärprogramm muß mit dieser Anweisung abgeschlossen werden. Bei Beendigung des Binärprogramms entspricht der im Register D enthaltene Wert dem Ergebnis der Funktion in C. Das `double`-Programm verdoppelt also sein C-Argument und liefert das Ergebnis an C zurück.

Deklaration von Variablen in Binärdateien

Die Marke `variable_foo` ist ein Beispiel für eine besondere Form, über die der Name und der Ort einer Variablen deklariert wird, auf die auch von C aus zugegriffen werden kann. Auf das Präfix »`variable_`« folgt der Name der Variablen – in diesem Fall »`foo`«. Auf diese Marke muß die Anweisung `FDB␣<number>` unmittelbar folgen.

Dies ist eine Anweisung an den Assembler, einen zwei Byte großen Wert zu erzeugen (der den Anfangswert der Variablen darstellt).

Die Variablen, die das Binärprogramm verwendet, müssen in der Binärdatei deklariert sein. Sie werden zu globalen C-Variablen, wenn die Binärdatei in C geladen wird.

Das nächste Binärprogramm in der Datei heißt »`set_foo`«. Es setzt den Wert der Variablen `foo`, indem es den Inhalt von Register D in dem Speicherbereich ablegt, der für `foo` reserviert wurde. Anschließend wird das Programm beendet.

Das folgende Binärprogramm heißt »`get_foo`«. Es lädt den Inhalt des für `foo` reservierten Speicherbereiches in das Register D und kehrt dann zurück.

Deklaration eines Initialisierungsprogramms

Die Marke `subroutine_initialize_module` ist eine besondere Form, die den Einsprungpunkt für den Code angibt, der zur Initialisierung des Binärprogrammes ausgeführt werden soll. Dieser Code wird bei Auftreten einer Neustart-Bedingung ausgeführt: beim Laden eines Programms auf die Platine, bei einem Hardware-Reset oder bei Ausführung der Funktion `main()`.

In dem Beispiel speichert der Initialisierungscode den Wert 69 in dem für die Variable `foo` reservierten Speicherbereich. Es wird also die 55 überschrieben, die normalerweise der Anfangswert für diese Variable wäre.

Globale Variablen, die in einem Binärmodul definiert sind, werden auf andere Weise initialisiert als globale Variablen, die in C definiert sind. In einem Binärmodul werden die Globalen nur dann auf ihren durch die `FDB`-Anweisung angegebenen Startwert initialisiert, wenn der Code zu der 6811-Platine hinuntergeladen wird (nicht bei einem Reset oder bei Aufruf von `main`, wie andere Globale). Die Initialisierungsroutine wird bei den üblichen Neustart-Bedingungen ausgeführt und kann zur Initialisierung der globalen Variablen verwendet werden, wie in dem Beispiel gezeigt wurde.

7.11.2 Interruptgesteuerte Binärprogramme

Interruptgesteuerte Binärprogramme benutzen die Initialisierungsroutine des Binärmoduls, um ein Stück Code in die Interruptstruktur des 6811 zu integrieren.

Der 6811 verfügt über mehrere Interrupts, die sich zum größten Teil mit der im Chip eingebauten Hardware beschäftigen, wie z. B. mit den Zeitgebern und Zählern. Einer dieser Interrupts wird von der Systemsoftware verwendet, um Zeitmessungs- und andere periodische Funktionen (wie z. B. die LCD-Anzeigeverwaltung) zu implementieren. Dieser Interrupt wird »System-Interrupt« genannt und läuft mit 1000 Hertz.

Anstatt einen anderen 6811-Interrupt zu verwenden, um ein benutzerdefiniertes Binärprogramm auszuführen, können zusätzliche Programme (die mit 1000 Hz

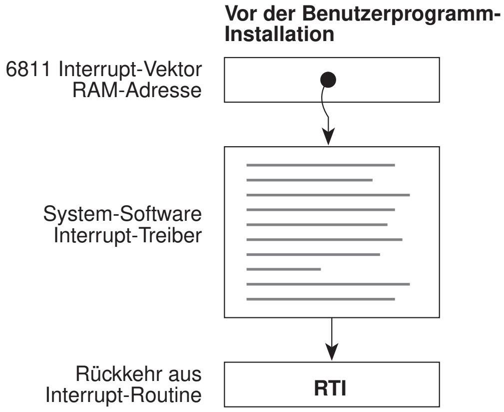

Abbildung 7.5: Interrupt-Struktur vor der Installierung eines Benutzerprogramms

oder weniger laufen müssen) sich selbst in den System-Interrupt installieren. Die Benutzerprogramme werden dann zu einem Teil der 1000 Hz Interrupt-Sequenz.

Um dies zu erreichen, läßt man das Benutzerprogramm den Original 6811-Interrupt Vektor »abfangen«, der auf den systemeigenen Interrupt-Code zeigt. Dieser Vektor wird so verändert, daß er auf das Benutzerprogramm zeigt. Wenn das Benutzerprogramm beendet wird, springt es an den Anfang des systemeigenen Interrupt-Codes.

Abbildung 7.5 zeigt die Interrupt-Struktur, bevor das Benutzerprogramm sich installiert hat. Der 6811-Vektor zeigt auf den systemeigenen Interrupt-Code, der mit einer »return from interrupt«-Anweisung endet.

Abbildung 7.6 stellt dar, wie die Interrupt-Struktur aussieht, nachdem das Benutzerprogramm installiert ist. Der 6811-Vektor zeigt auf das Benutzerprogramm, das mit einem Sprung zum systemeigenen Software-Treiber endet. Dieser Treiber endet wie zuvor mit der RTI-Anweisung.

Es können mehrere Benutzerprogramme auf diese Weise installiert werden. Jedes Programm installiert sich vor den bereits vorhandenen Programmen.

In Abbildung 7.7 ist ein Beispiel für ein Programm zu sehen, daß sich selbst in den System-Interrupt installiert. Jedesmal, wenn das Programm ausgeführt wird, schaltet es die Signalleitung um, die den Piezo-Summer ansteuert. Da der System-Interrupt mit 1000 Hz läuft, erzeugt dieses Programm einen anhaltenden Ton bei 500 Hz.

Die erste Zeile hinter dem aus Kommentaren bestehenden Kopf bindet eine Datei »`6811regs.asm`« ein. Diese Datei enthält Entsprechungen für alle 6811-Register und Interrupt-Vektoren, von denen die meisten Binärprogramme wenigstens ein paar benötigen. Es ist am einfachsten, sie alle in einer Datei aufzubewahren, so daß sie leicht eingebunden werden können.

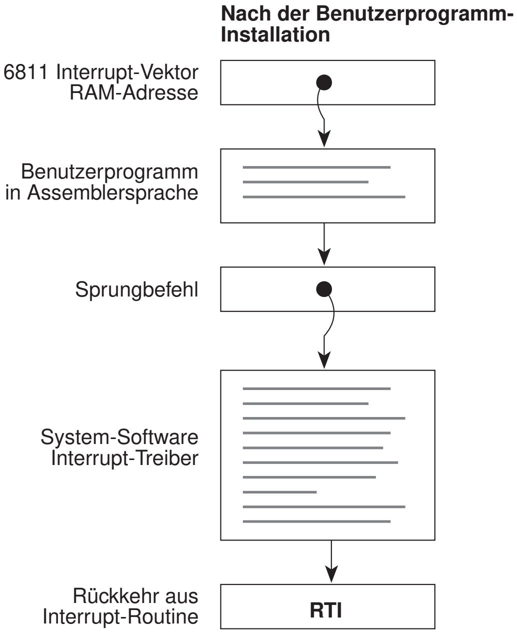

Abbildung 7.6: Interrupt-Struktur nach der Installierung eines Benutzerprogramms

Der Initialisierungsteil des Programms beginnt mit der Deklaration `subroutine_initialize_module`. Als erstes wird die Datei »`ldxibase.asm`« eingebunden. Diese Datei enthält einige Zeilen 6811-Assemblercode, der die Aufgabe hat, den Basiszeiger auf den 6811-Interruptvektorbereich zu ermitteln. Der Zeiger wird in das X-Register des 6811 geladen.

Die nächsten 4 Codezeilen installieren die Interruptroutine (die mit der Marke `interrupt_code_start` beginnt) entsprechend der Methode, die in Abbildung 7.6 dargestellt ist. Zunächst wird der bestehende Interrupt-Zeiger geholt. Wie der Kommentar angibt, wird der TOC4-Zeitgeber des 6811 benutzt, um den System-Interrupt zu implementieren. Der Vektor wird in die JMP-Anweisung geschrieben, mit der die Interrupt-Routine beendet wird. Als nächstes wird der 6811-Interruptzeiger durch einen Zeiger auf die neue Interrupt-Routine ersetzt. Mit diesen beiden Schritten ist die Initialisierung abgeschlossen.

Die eigentliche Interrupt-Routine ist ziemlich kurz. Sie invertiert Bit 3 vom PORTA-Registers des 6811. Das PORTA-Register steuert die acht Pins von Port A, an das externe Hardware angeschlossen ist. Bit 3 ist an den Piezo-Summer angeschlossen. Der Interrupt-Code endet mit einer Sprunganweisung. Das Argument für diesen Sprung wurde vom Initialisierungsprogramm überschrieben.

Diese Methode erlaubt es einer beliebigen Anzahl von Programmen in getrennten Dateien sich selbst in den System-Interrupt zu installieren. Da die Dateien von der C-

```
* icb-Datei:  "sysibeep.asm"
*
*   Beispiel für ein Programm, das sich selbst in den
*   SystemInt 1000 Hz Interrupt installiert
*
*   Fred Martin
*   Thu Oct 10 21:12:13 1991
*
#include <6811regs.asm>
ORG     MAIN_START
subroutine_initialize_module:

#include <ldxibase.asm>
* X enthält nun den Basis-Zeiger auf die Interrupt-Vektoren
* ($FF00 oder $BF00)

* aktuellen Vektor ermitteln; den Wert so in dieses Programm schreiben,
* daß im Anschluß dorthin verzweigt wird
LDD     TOC4INT,X            ; SystemInt auf TOC4
STD     interrupt_code_exit+1

* dieses Programm in den Interruptvektor installieren
LDD     #interrupt_code_start
STD     TOC4INT,X
RTS

* die Interrupt-Routine startet hier
interrupt_code_start:
* bei jedem Aufruf den Beeper ansteuern
LDAA    PORTA
EORA    #%00001000     ; beeper bit
STAA    PORTA

interrupt_code_exit:
JMP     $0000    ; dieser Wert wird bei der Initialisierung überschrieben
```

Abbildung 7.7: `sysibeep.asm`: Ein Binärprogramm, das sich in den System-Interrupt installiert.

```
S116802005390037FD802239FC802239CC0045FD8022393C
S9030000FC
S116872B05390037FD872D39FC872D39CC0045FD872D39F4
S9030000FC
6811 assembler version 2.1  10-Aug-91
   please send bugs to Randy Sargent (rsargent@athena.mit.edu)
   original program by Motorola.
subroutine_double 872b *0007
subroutine_get_foo 8733 *0021
subroutine_initialize_module 8737  *0026
subroutine_set_foo 872f  *0016
variable_foo 872d *0012 0017 0022 0028
```

Abbildung 7.8: Ein Beispiel für eine binäre IC-Objektdatei: `testicb.icb`

Umgebung geladen werden können, bietet dieses Vorgehen dem Benutzer maximale Flexibilität in Verbindung mit einer hohen Code-Effizienz.

7.11.3 Die binäre Objektdatei

Der Name eine Quelldatei für ein Binärprogramm muß das Suffix `.asm` tragen. Wenn eine `.asm`-Datei angelegt wurde, wird eine spezielle Version des 6811-Assemblerprogramms benutzt, um den binären Objektcode aufzubauen. Dieses Programm erzeugt eine Datei, die den assemblierten Maschinencode und die Definition der Marken für Einsprungpunkte und C-Variablen enthält.

Um den Quellcode zu assemblieren und eine binäre Objektdatei zu erzeugen, wird das Programm `as11_ic` benutzt. Als Argument wird der Dateiname der Quelldatei angegeben. Die erzeugte Objektdatei erhält automatisch das Suffix `.icb` (für »IC Binary«). Abbildung 7.8 zeigt eine binäre Objektdatei, die aus der Beispieldatei `testicb.asm` erzeugt wurde.

7.11.4 Laden einer `icb`-Datei

Sobald eine `.icb`-Datei erzeugt wurde, kann sie wie jede andere C-Datei in IC geladen werden. Wenn in Zusammenhang mit einem binären Programm C-Funktionen verwendet werden sollen, ist es üblich, sie in eine Datei zu schreiben, die denselben Namen trägt wie die `.icb`-Datei, und eine `.lis`-Datei zu benutzen, um die beiden Dateien gemeinsam zu laden.

7.11.5 Feldzeiger an ein Binärprogramm übergeben

Ein Zeiger auf ein Feld ist eine ganzzahlige 16-Bit Adresse. Um einen Feldzeiger in eine Ganzzahl umzuwandeln, kann die folgende Form benutzt werden:

```
feld_zgr = (int) feld;
```

Dabei ist `feld_zgr` eine ganze Zahl und `feld` ist ein Feld.

Beim Kompilieren eines Programms, das diese Art der Zeigerumwandlung benutzt, muß IC in einem besonderen Modus betrieben werden. Normalerweise verbietet IC bestimmte Arten der Zeiger-Manipulation, durch die das System abstürzen könnte. Um diese Art von Code trotzdem zu übersetzen, muß IC durch den folgendem Aufruf gestartet werden:

```
ic -wizard
```

In der internen Repräsentation besteht ein Feld aus einem zwei Byte langen Wert, gefolgt vom eigentlichen Feldinhalt.

7.12 Format und Verwaltung von IC-Dateien

Dieser Abschnitt beschreibt, wie IC mehrere Quelldateien handhabt.

7.12.1 C-Programme

Alle Dateien, die C-Code enthalten, müssen einen Namen haben, der das Suffix `.c` hat.

Um Funktionen aus mehr als einer C-Datei zu laden, kann in der IC-Kommandozeile für jede Datei ein Befehl zum Laden eingegeben werden. Um z. B. die C-Dateien `da.c` und `tei.c` zu laden, kann folgende Eingabe dienen:

```
C> load da.c
C> load tei.c
```

Alternativ können die Dateien mit einem einzelnen Befehl geladen werden:

```
C> load da.c tei.c
```

Wenn die zu ladenden Dateien Abhängigkeiten enthalten (z. B., wenn eine Datei eine Funktion enthält, die auf eine Variable oder Funktion in der anderen Datei zugreift), muß entweder die zweite Methode angewendet (mehrere Dateinamen in einem `load`-Befehl) oder der folgende Ansatz verfolgt werden.

7.12.2 List-Dateien

Wenn das Programm in mehrere Dateien aufgeteilt ist, die stets zusammen geladen werden, kann eine »List-Datei« angelegt werden. Diese Datei weist IC an, eine Reihe von angegebenen Dateien zu laden. Um das oben verwendete Beispiel fortzuführen, kann eine Datei `projekt.lis` angelegt werden:

Inhalt von `projekt.lis`:

```
da.c
tei.c
```

Wird anschließend der Befehl `load␣projekt.lis` in der C-Kommandozeile eingegeben, werden sowohl `da.c` als auch `tei.c` geladen.

7.12.3 Datei- und Funktionsverwaltung

Dateien entladen

Wenn Dateien in IC geladen werden, bleiben sie geladen, bis sie ausdrücklich entladen werden. Dies entspricht normalerweise der gewünschten Funktionalität. Wenn an einer der Programmdateien gearbeitet wurde, bleiben die übrigen im Speicher, so daß sie nicht jedesmal ausdrücklich neu geladen werden müssen, wenn die in Entwicklung befindliche Datei wieder geladen wird.

Angenommen, die Datei `da.c` wird geladen und die Datei enthält eine Definition für die Funktion `main`. Als nächstes wird die Datei `tei.c` geladen, die ebenfalls eine Definition der Funktion `main` enthält. Dies führt zu einer Fehlermeldung, da beide Dateien ein `main` enthalten. Aufgrund des Fehlers wird IC die Datei `tei.c` wieder entladen und die Datei `da.c` sowie alle anderen zur Zeit geladenen Dateien neu hinunterladen.

Die Lösung besteht darin, zunächst die Datei mit dem unerwünschten `main` zu entladen und dann die Datei mit dem neuen `main` zu laden:

```
C> unload da.c
C> load tei.c
```

7.13 IC konfigurieren

IC verfügt über eine große Anzahl von Kommandozeilen-Schaltern, die dem Benutzer die Kontrolle über verschiedene Dinge ermöglichen. Wenn der Befehl »`ic␣-help`« eingegeben wird, gibt IC Erklärungen zu den Schaltern aus.

IC speichert intern den Suchpfad und die Namen der Bibliotheksdateien. Diese Einstellungen können mit dem Befehl »`ic␣-config`« geändert werden. Wenn dieser Befehl ausgeführt wird, fragt IC nach einem neuen Pfad und dem Namen einer Bibliotheksdatei und erzeugt eine neue ausführbare Kopie von sich selbst, die diese Änderungen enthält.

Anhang A

Die IC-Bibliotheksdatei

Die Bibliotheksdateien enthalten Standard C-Funktionen, die eine Kommunikation zwischen der Hardware und der Steuerungsplatine des Roboters ermöglichen. Diese Funktionen liegen entweder in C oder Assemblersprache vor. Die Bibliotheksdateien dienen zur Steuerung der Motoren, zur Erzeugung von Tönen und zur Erfassung der Sensordaten.

Das IC-Programm lädt die Bibliotheksdatei automatisch, sobald sie aufgerufen wird. Eventuell müssen Sie die Konfiguration von IC ändern, um andere als die Standard-Bibliotheksdateien zu laden.

Die in diesem Abschnitt beschriebenen Funktionen beziehen sich auf die Programmversion 2.0 und höher.

A.1 Taktbefehle

Der Systemcode protokolliert die abgelaufenen Zeit in Millisekunden. Die zeitangebenden Variablen sind vom Typ *long integer* (lange ganze Zahlen). Die Standardfunktionen ermöglichen die Verwendung von Fließkommavariablen in den Zeitgeberfunktionen.

`void reset_system_time()`
Setzt den Zählwert der Systemzeit auf 0 Millisekunden zurück.

`long mseconds()`
Gibt den Wert der Systemzeit in Millisekunden aus. Er wird durch einen Hardware-Reset (d. h. durch Drücken der RESET-Taste auf der Platine oder die Funktion `reset_system_time()` zurückgesetzt. `mseconds()` liegt als C-Befehl vor (nicht als Bibliotheksfunktion).

`float seconds()`
Gibt den Wert der Systemzeit als Fließkommazahl in Sekunden aus. Die Schrittweite beträgt eine Millisekunde.

`void sleep(float sec)`
Wartet eine bestimmte Zeitspanne, die gleich oder größer `sec` ist. `sec` ist eine Fließkommazahl. Beispiel:

```
/*  1,5 Sekunden lang warten  */
sleep(1.5);
```

`void msleep(long msec)`
Wartet eine bestimmte Zeitspanne, die gleich oder größer `msec` Millisekunde ist. `msec` ist ein long integer. Beispiel:

```
/*  1,5 Sekunden lang warten */
msleep(1500L);
```

A.2 Tonerzeugung

Die vom Piezo-Summer erzeugten Töne werden durch mehrere Befehle gesteuert.

`void beep()`
Erzeugt für eine Zeitspanne von 0,3 Sekunden einen Ton von 500 Hz.

`void tone(float frequency, float length)`
Gibt für eine Zeitspanne von `length` Sekunden einen Ton von `frequency` Hz aus. Sowohl `frequency` als auch `length` sind Fließkommazahlen.

`void set_beeper_pitch(float frequency)`
Setzt den Summerton auf `frequency` Hz. Zum Einschalten des Summers wird die Funktion `beeper_on()` verwendet. Wenn der Summerton geändert wird, während der Summer an ist, werden »gewobbelte« Sinustöne erzeugt.

`void beeper_on()`
Schaltet den Summer mit der zuletzt gewählten `set_beeper_pitch`-Frequenz an.

`void beeper_off()`
Schaltet den Summer aus.

A.3 Sensordatenerfassung

`int digital(int nth)`
Gibt den Status des $n$ten Eingabebits des digitalen I/O-Ports A aus. Die Bits 1 und 2 sind nicht belegte Eingänge. Die Bits 0 und 7 sind Eingänge, die mit dem linken bzw. rechten Rad-Encoder verbunden sind. Die Bits 3, 4, 5 und 6 sind Ausgänge. Durch den Aufruf `digital(0)` sollte beispielsweise der gleiche Wert wie für `left_shaft()` ausgegeben werden.

`int analog(int p)`
Gibt den Wert des Sensor-Ports mit der Nummer `p` aus. Das Ergebnis ist eine ganze Zahl zwischen 0 und 255.

Analog-Ports sind im Port E des 6811 abgebildet. Analog(0) entspricht `PE0`, Analog(1) `PE1` usw. Die Leitungen `PE6` und `PE7` stehen auf dem Erweiterungsstecker für Benutzerspezifikationen bereit. Wurde der optionale Wärmesensor nicht eingebaut, kann außerdem `PE5` (verfügbar auf dem Stecker für den Wärmesesor) verwendet werden.

`int analog(photo_right | photo_left)`
Gibt den Wert der rechten bzw. linken Photozelle aus. Je niedriger die Zahl, desto heller das Licht.

`int analog(microphone)`
Gibt den augenblicklichen Wert des A/D-Eingangs aus, der mit dem Mikrophon verbunden ist. Wurde kein Geräusch erfaßt, gibt `analog(microphone)` einen Wert von etwa 128 aus. Hat das Mikrophon ein Geräusch erfaßt, kann der Wert niedriger oder höher als 128 sein. Um einen aussagekräftigen Wert für den Geräuschpegel zu erhalten, muß das Mikrophon häufig abgetastet und die Werte gemittelt werden.

`int analog(pyro)`
Gibt den Wert des optionalen Wärmesensors aus. Sofern keine Wärmequelle den Wärmesensor passiert, sollte der Wert einigermaßen konstant sein.

`int bumper()`
Gibt einen 3-Bit Wert aus, der die Anzahl der geschlossenen Kollisionsschalter angibt. Wenn die Kollisionsschalter mit A, B und C gekennzeichnet sind, dann entspricht Bit 0 dem geschlossenen Schalter A, Bit 1 dem geschlossenen Schalter B und Bit 3 dem geschlossenen Schalter C. Gibt `bumper()` den Wert ‘3’ (binär 011) aus, bedeutet dies also, daß Schalter A und B geschlossen sind.

`int ir_detect()`
Gibt einen 2-Bit-Wert mit folgender Bedeutung aus: 0 – kein Hindernis erfaßt, 1 – Hindernis auf der rechten Seite erfaßt, 2 – Hindernis auf der linken Seite erfaßt, 3 – Hindernis auf beiden Seiten erfaßt. Dieser Test benötigt mindestens 2 Millisekunden.

`int leds(int n)`
Setzt die Korrektur-LEDs auf den Binärwert der vier niederwertigen Bits von `n`. Sind die LEDs, von der rechten Seite der Platine beginnend, von 0 bis 3 gekennzeichnet, dann entspricht Bit 0 von n der LED 0, Bit 1 der LED 1 usw. Beachten Sie, daß die Korrektur-LEDs mit dem Motorrichtungsbit und den IR-Sendedioden parallel geschaltet sind. Die LEDs können also nicht ohne Aktivierung dieser anderen Funktionen aktiviert werden und umgekehrt.

A.4 Motorfunktionen

Die Antriebsmotoren des Rug Warrior sind geschwindigkeitsgesteuert. Das heißt, daß es möglich ist, nicht nur festzulegen, ob der Motor an- oder ausgeschaltet ist, sondern auch mit welcher Geschwindigkeit er laufen soll. Die nachfolgend beschriebenen Befehle steuern nur einen offenen Regelkreis. Dennoch ist es möglich, mit diesen Befehlen auch einen geschlossenen Regelkreis zur Geschwindigkeitssteuerung zu erstellen.

`int init_motors()`
Diese Funktion initialisiert mehrere Register, die eine Steuerung der Geschwindigkeit der Motoren ermöglichen. Bei einem RESET wird sie automatisch aufgerufen. Sie muß jedoch eventuell erneut aufgerufen werden, wenn die Register des MC68HC11 (OC1M, TCTL1, TOC1 oder DDRD) geändert wurden.

`void motor(int index, int speed)`
Befehl zur Steuerung der Motorgeschwindigkeit. Index $= 0$ beschleunigt den linken Motor, Index $= 1$ beschleunigt den rechten Motor. Die Geschwindigkeit wird als ganze Zahl zwischen -100 und $+100$ angegeben und zwar als Prozentangabe der Höchstgeschwindigkeit, bei welcher der Motor arbeitet. *In älteren Versionen als 2.0 wird `speed` als Fließkommazahl angegeben.*

`void drive(int trans_vel, int rot_vel)`
Funktion zur Steuerung der Robotergeschwindigkeit und Fahrtrichtung. `trans_vel` ist eine ganze Zahl zwischen -100 und $+100$, welche die Geschwindigkeit des Roboters als Prozentwert der Höchstgeschwindigkeit angibt (negative Zahlen veranlassen den Roboter, rückwärts zu fahren). `rot_vel` ist eine ganze Zahl zwischen -100 und $+100$, welche die Umdrehungsgeschwindigkeit des Roboters als Prozentangabe der maximalen Geschwindigkeit angibt. Positive Zahlen entsprechen einer Umdrehung gegen den Uhrzeigersinn. Diese Funktion ist nur für Geschwindidgkeiten in offenen Regelkreisen ausgelegt. *In älteren Versionen als 2.0 sind `trans_vel` und `rot_vel` als Fließkommazahlen angegeben.*

A.5 Rad-Encoder

Im Gegensatz zu den anderen Bibliotheksroutinen werden die meisten Funktionen, die zur Überwachung der Rad-Encoder dienen, nicht automatisch geladen. Sie müssen durch Laden von `shaft.lis` explizit aufgerufen werden. Diese Datei lädt `speed.icb` und `shaft.c`. Die erste Datei startet eine Interrupt-Routine, die den mit `PA0` verbundenen Rad-Encoder überwacht. Die zweite Datei stellt eine maschinennahe Benutzerschnittstelle in Form der Funktionen `get_left_clicks()` und `get_right_clicks()` zur Verfügung. Jedesmal wenn eine dieser Routinen aufgerufen wird, gibt sie die Anzahl der Rad-Encoderimpulse seit dem letzten Aufruf aus. Diese Zahl ist proportional zum Weg, den das Rad während dieser Zeitspanne zurückgelegt hat.

Um die Geschwindigkeit des Roboters zu messen, sollten Sie dafür sorgen, daß die get_clicks-Funktionen in regelmäßigen Abständen aufgerufen werden. Dies könnte z. B. durch ein Unterprogramm geschehen, das eine get_clicks-Funktion aufruft, den Wert einer Variablen zuweist und anschließend eine bestimmte Zeitspanne wartet. Wie häufig eine get_clicks-Funktion aufgerufen werden sollte, hängt von der Anzahl der schwarz-auf-weiß Übergänge der Kodierscheibe auf dem Rad ab (beim Mechanik-Bausatz sind es 16) sowie der benötigten Geschwindigkeitsauflösung. Die Rad-Encoder unterscheiden nicht die Motorlaufrichtung. Die get_clicks-Funktionen registrieren dagegen, ob das Rad sich in oder gegen den Uhrzeigersinn dreht.

`void init_velocity()`
Initialisiert die Routinen, die eine Geschwindigkeitsüberwachung ermöglichen. Diese Funktion muß abgerufen werden, damit die Ergebnisse der Funktionen `get_left_clicks()` und `get_right_clicks()` eine Bedeutung erhalten.

`int get_left_clicks()`
Greift auf eine Variable in einer maschinennahen Interrupt-Routine zu (definiert in `speed.asm`). Die Interrupt-Routine ist dem Pin `PA0` des 6811 zugeordnet. Der zurückgegebene Wert gibt die Anzahl der Zählimpulse seit dem letzten Aufruf von `get_left_clicks` an.

`int get_right_clicks()`
Greift auf das Impulszähler-Register zu, das mit dem Pin `PA7` des 6811 verbunden ist. Der ausgegebene Wert gibt die Anzahl der Zählimpulse seit dem letzten Aufruf von `get_right_clicks` an.

Zur Erleichterung der Programmierung enthält die Rug Warrior-Bibliothek zwei weitere Funktionen.

`left_shaft()`
Gibt den aktellen Status des linken Rad-Encoders an: 1 bei Erfassung eines weißen Streifens, ansonsten 0.

`right_shaft()`
Gibt den aktuellen Status des rechte Rad-Encoders an: 1 bei Erfassung eines weißen Streifens, ansonsten 0.

Anhang B

Rechneranschluß und technische Daten

B.1 Serielle Verbindung

Abbildung B.1 wurde zur zusätzlichen Information hinzugefügt. Sie kann hilfreich sein, wenn Sie selbst ein Kabel für die direkte Verbindung zum Rug Warrior konstruieren wollen, entweder mit einem Macintosh DIN-8 Stecker oder einem IBM PC 9-Pin Stecker.

Die Zeichnung zeigt mehrere gebräuchliche Host-Computer Stecker mit den wesentlichsten Anschlüssen. Verdrahten Sie Ihr Kabel so, daß Gnd, TxD und RxD an Ihrem Host-Computer mit den entsprechenden Signalen an der Rug Warrior-Buchse für den seriellen Anschluß verbunden sind. Je nach Schnittstellenkarte kann es notwendig sein, die Pins 5, 6 und 20 des IBM 25-Pin Steckers, bzw. die Pins 4, 6 und 8 des IBM 9-Pin Steckers zusammen zu verdrahten. Beim seriellen Stecker des Macintosh müssen die Pins 4 und 8 verbunden werden.

B.2 Ausgewählte technische Daten des Rug Warrior

Die Tabelle in Abbildung B.2 gibt die typischen Daten der Rug Warrior-Roboter an, die mit dem erweiterten Bausatz konstruiert werden.

Der Erfassungsbereich der IR-Detektoren geht von einem flachen weißen Hindernis und einer optimalen Oszillatoreinstellung aus.

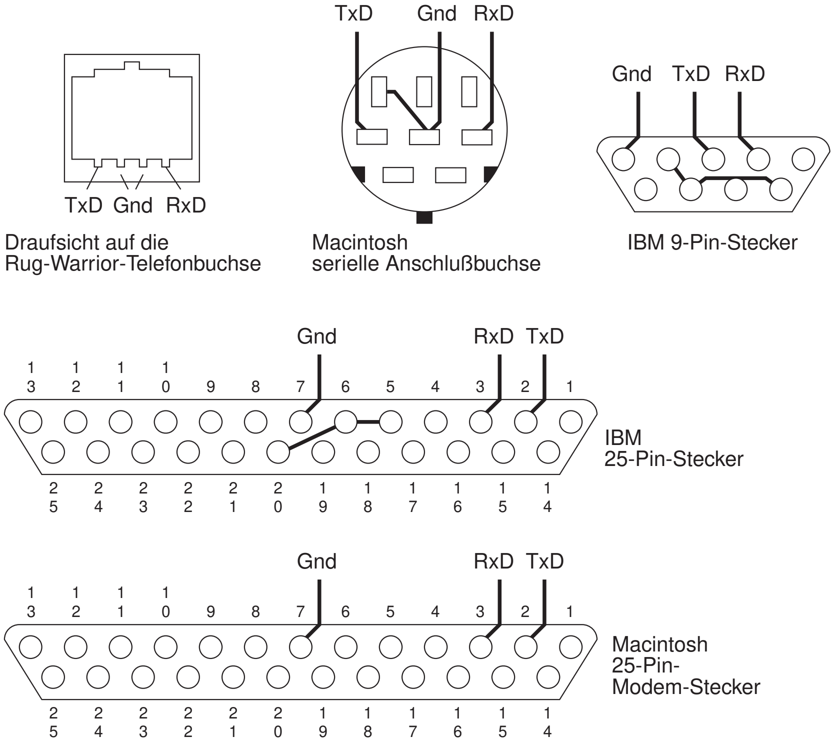

Abbildung B.1: Der Rug Warrior Stecker-benötigt nur drei Signale: Masse (Gnd), Senden (TxD) und Empfangen (RxD).

Bauteil	Typisch	Min	Max	Einheiten
Spannungsversorgung Schaltlogik	5,0	4,6	7,0	Volt
Spannungsversorgung Motor	5,0	4,0	9,0	Volt
Mikroprozessor Taktfrequenz	2,0			MHz
IR Osc. Freq.	40	38	42	kHz
Hinderniserfassungsreichweite	ca. 15–20			cm
Robotergeschwindigkeit	ca. 0,45			m/s
Zählimpulse pro Umdrehung	16			Einheiten
Robotergewicht (mit Batterien)	ca. 900			g
Motorstrom (pro Motor)			1,0	Ampere
Durchmesser der Räder	ca. 6			cm
Durchmesser des Roboters	ca. 16			cm
Höhe des Roboters	ca. 10			cm

Abbildung B.2: Typische Daten der Rug Warrior-Roboter.