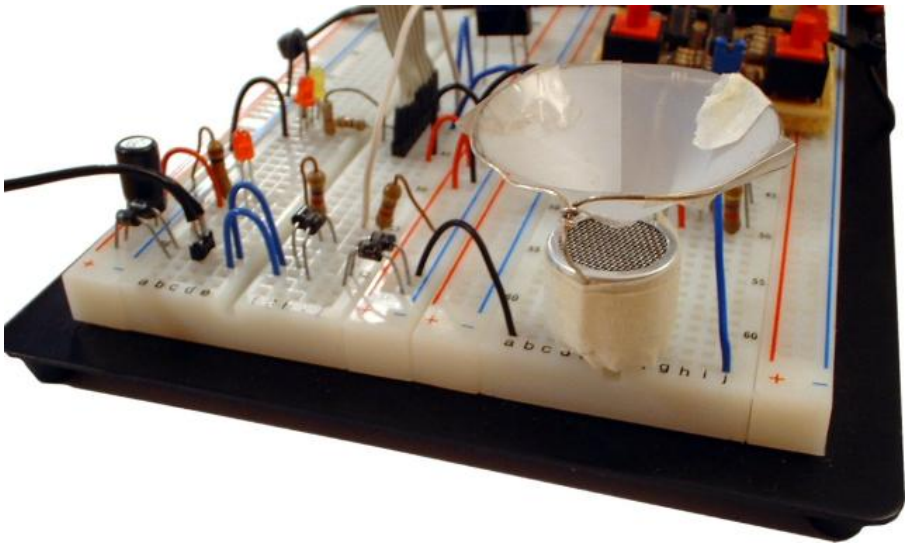
Entwicklung eines Ultraschall-basierten Ortungssystems für Lego Mindstorms Roboter

Studienarbeit von Stephan Höhrmann



Christian-Albrechts-Universität zu Kiel
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme
Prof. Dr. Reinhard von Hanxleden

15. Februar 2005
Betreuer: Jan Lukoschus

Studienarbeit von Stephan Höhrmann

Veröffentlicht am 15. Februar 2005

Mein besonderer Dank gilt allen, die mich in den letzten beiden Jahren unterstützt haben. Reinhard hat mir eine Aufgabenstellung und viele Freiheiten in der Realisierung gegeben, so daß ich eine Menge ausprobieren und lernen konnte. Jan hatte bei technischen Fragen immer ein offenes Ohr und hat sich die Zeit genommen, über verschiedene Ideen zu diskutieren. Xenia hat mir unermüdlich zugehört und viel Zeit investiert, mir bei der Verbesserung dieser Ausarbeitung zu helfen. Außerdem bedanke ich mich bei allen, die viele Ergebnisse ihrer Arbeit im Internet frei zugänglich veröffentlichen, insbesondere in Form von Dokumentation, Programmcode und Erfahrungsberichten. Ohne diese breite Wissensbasis wäre diese Arbeit nicht möglich gewesen.

Für Xenia

Inhaltsverzeichnis

1 Einleitung **9**
1.1 Aufgabenstellung . . . 10
1.2 Vergleichbare Systeme . . . 11
1.3 Gliederung dieser Arbeit . . . 15

I Grundlagen **17**

2 Lego Mindstorms Elektronik **19**
2.1 Robotic Command Explorer . . . 19
2.2 Kabel und Anschlüsse . . . 20
2.3 Motoren und andere Aktoren . . . 21
2.4 Passive Sensoren . . . 25
2.5 Aktive Sensoren . . . 27
2.6 Infrarot-Kommunikation . . . 31
2.7 Abmessungen der Legosteine . . . 36
2.8 Einfluß der Batteriespannung . . . 37

3 Einsatz von Mikrocontrollern **39**
3.1 Auswahl eines Mikrocontrollers . . . 40
3.2 Die PIC-Familie von Microchip . . . 42
3.3 Entwicklung der Steuersoftware . . . 46
3.4 Integration in eine Schaltung . . . 46
3.5 Aufbau von Programmiergeräten . . . 49
3.6 Serielles Debug-Interface . . . 53

4 Ortung mit Ultraschall **55**
4.1 Basistechnik Entfernungsmessung . . . 56
4.2 Erzeugen und Detektieren von Ultraschall . . . 58
4.3 Optimale Geometrie eines Reflektors . . . 60
4.4 Fehlerquellen beim Messen . . . 62
4.5 Berechnung der Koordinaten . . . 63
4.6 Beschreibung verschiedener Szenarien . . . 65

II Realisierung **75**

5 Ein Ortungssystem für Mindstorms **77**
5.1 Anforderungen an das Ortungssystem . . . 77
5.2 Beschreibung des realisierten Systems . . . 80

6 Verwendete Baugruppen **83**
6.1 Einsatz von Mikrocontrollern . . . 83
6.2 Ultraschallsender . . . 84
6.3 Ultraschalldetektor . . . 85
6.4 Synchronisation per Funk . . . 86
6.5 Steuerung der Leuchtfeuer . . . 89
6.6 Stromversorgung . . . 91
6.7 Kommunikation mit Aktoren . . . 93
6.8 Datenübertragung vom Sensor . . . 95
6.9 Ansteuerung des LC-Displays . . . 97

7 Zusammensetzung der Module **99**
7.1 Steuereinheit . . . 99
7.2 Leuchtfeuer . . . 101
7.3 Mobiler Sender . . . 103
7.4 Empfänger . . . 104

8 Einsatz aus Anwendersicht **107**
8.1 Installation des Systems . . . 108
8.2 Ansteuerung mit brickOS . . . 110
8.3 Auswertung der Meßdaten . . . 114
8.4 Ortung ohne Funksignal . . . 119
8.5 Anforderungen an das Sendetiming . . . 126
8.6 Berücksichtigung der Bewegung . . . 127
8.7 Beispielprogramme . . . 127

9 Ergebnisse **129**
9.1 Technische Eckdaten . . . 129
9.2 Einsatz in den Übungen . . . 132
9.3 Mögliche Verbesserungen . . . 134
9.4 Fazit . . . 135

III Anhänge **137**

A Verwendete Schaltungen **139**
A.1 Grundschaltung für Sensoren . . . 139
A.2 Mindstorms Schallsensor . . . 140
A.3 Infrarot-Transceiver für den PC . . . 142
A.4 Prototyp-Platine für PICs mit 8 Pins . . . 143
A.5 Prototyp-Platine für PICs mit 18 Pins . . . 145
A.6 Programmiergerät für PICs . . . 147
A.7 ICSP-Adapter für das Programmiergerät . . . 151
A.8 Serielles Debug-Interface für PICs . . . 153
A.9 Ultraschallreflektoren . . . 155
A.10 Steuereinheit des Ortungssystems . . . 157
A.11 Stationäres Leuchtfeuer . . . 160
A.12 Verkabelung und Terminatoren . . . 162
A.13 Sender für den RCX . . . 162
A.14 Empfänger für den RCX . . . 165
A.15 Vibrationsdämpfer . . . 167

A.16 Timer für UV-Belichter 167
A.17 Joystick mit Infrarotsender 170

B Erstellte Programme **175**
B.1 Programmierung des Mindstorms-Towers 175
B.2 Multiprotokoll-Paketmonitor für Mindstorms 182
B.3 Disassembler für PIC-Programme 182
B.4 Konvertierung von Dateien im Hex-Format 185
B.5 Programmiersoftware für PIC Mikrocontroller 185
B.6 Serielles Debug-Interface für PICs 187
B.7 Numerische Fehlersimulation der Ortung 190
B.8 Serielle Datenübertragung vom Sensor zum RCX 190
B.9 BrickOS-API des Ortungssystems 193
B.10 Algorithmus zum Ziehen von Quadratwurzeln 195

C Meßwerte zu den Abbildungen **197**
C.1 Drehmoment des Lego-Motors 71427 (Abbildung 2.5) 197
C.2 Klemmenspannung am Aktorausgang (Abbildung 2.6) 198
C.3 Sensorwerte im passiven Modus (Abbildung 2.9) 198
C.4 Klemmenspannung bei aktiven Sensoren (Abbildung 2.13) 199
C.5 Ultraschall abhängig von der Treiberspannung (Abbildung 4.4) 200
C.6 Ultraschall abhängig von der Entfernung (Abbildung 4.5) 200
C.7 Fehlersimulation für Koordinatenberechnung 201
C.8 Fehler der Entfernungsmessung (Abbildung 9.1 und 9.6) 201
C.9 Fehler der Koordinatenberechnung (Abbildung 9.2 und 9.3) 203

D Platinenherstellung **205**
D.1 Erstellen der Layoutvorlage 206
D.2 Platinenmaterial und seine Bearbeitung 209
D.3 Fotochemisches Erzeugen der Leiterbahnen 210
D.4 Bohren, Beschriften und Versiegeln 214
D.5 Bestücken der Platine 215
D.6 Herstellung doppelseitiger Platinen 216

Literaturverzeichnis **217**

Inhalt der CD **221**

Quellennachweise **223**

Erklärung **225**

Kapitel 1

Einleitung

Lego Mindstorms ist ein Roboterbaukasten, der auf Lego Technic basiert. Zusätzlich zu den bekannten Bausteinen, Achsen, Zahnrädern und Motoren gibt es einen RCX genannten Baustein, der einen programmierbaren Computer enthält. Dieser basiert auf einem Mikrocontroller und kann mit bis zu drei Sensoren und Aktoren verbunden werden. Außerdem verfügt er über eine Infrarot-Schnittstelle zur Kommunikation, mit der unter anderem Programme von einem PC heruntergeladen werden können. Der RCX ist daher in der Lage, das Verhalten eines Lego-Roboters zu steuern. Ursprünglich war Mindstorms als Spielzeug für Kinder ab 12 Jahren gedacht, der Baukasten wurde aber sehr schnell von Kunden abseits der Zielgruppe entdeckt. So setzen beispielsweise viele Schulen und Universitäten Mindstorms mit Erfolg zur Unterstützung der Lehre und im Übungsbetrieb ein. Auf dieser Plattform können viele Konzepte aus den Bereichen Robotik, Programmierung eingebetteter Systeme, Echtzeitverhalten und Cross-Platform Development ausprobiert werden, ohne daß vorher aufwendige Hardware hergestellt werden muß.

Abbildung 1.1: Robotics Invention System von 1998

Die ersten Bausätze kamen im September 1998 in den USA auf den Markt. Bereits wenige Wochen danach hatte Kekoa Proudfoot, damals Student der Universität Stanford, das zwischen PC und RCX eingesetzte Kommunikationsprotokoll und die Funktionsweise der Software zu großen Teilen entschlüsselt. Seine Ergebnisse stellte er am 7. Oktober in einem Seminar vor, die Folien sind immer noch unter [Proudfoot] zu finden. Damit stand die Möglichkeit offen, andere Programme als den von Lego mitgelieferten Bytecode-Interpreter auf dem RCX zu installieren.

Auf dieser Grundlage entstanden in der Folgezeit viele Programmiersprachen und Entwicklungswerkzeuge für den RCX. Zu den bekanntesten Vertretern gehören pbForth, lejOS, NQC, und brickOS, die Programmierung in den Sprachen Forth, Java und C unterstützen. Bei brickOS handelt es sich um ein

Posix-ähnliches Betriebssystem, das in Assembler und C geschrieben ist. Der Kernel und die Anwendungsprogramme werden mit einem Crosscompiler übersetzt und über die Infrarotschnittstelle auf den RCX übertragen. Die Möglichkeit, komplexere Verhaltensweisen programmieren zu können, bringt den Entwickler aber auch schnell an die Grenzen des Systems, das trotz allem ein Kinderspielzeug bleibt.

Die mit Abstand häufigste Anwendung besteht darin, einen autonomen mobilen Roboter zu bauen, der eine Aufgabe erfüllen soll und sich dafür bewegen muß. Abbildung 1.2 zeigt ein typisches Beispiel hierfür. Der Roboter hat die Aufgabe, auf dem gesamten Feld nach Holzklötzen zu suchen und diese an den Rand vor die Lampe zu schieben. Der Roboter muß seine Position kennen, damit er diese Aufgabe lösen kann. Ohne diese Information kann er das Feld nicht systematisch absuchen oder nach dem Abliefern eines Holzstückes seine Suche an der letzten Fundstelle fortsetzen.

Abbildung 1.2: Roboter beim Einsammen von Holzklötzen

Ein Roboter kann seine Position mit den Bordmitteln von Lego Mindstorms nicht direkt feststellen, er kann nur auf Koppelnavigation zurückgreifen. Dieses Verfahren geht von einer bekannten Startposition aus und addiert alle stattgefundenen Bewegungen des Roboters darauf. Dabei wird üblicherweise die reine Laufzeit der Motoren als Grundlage für die Berechnungen genommen. Im günstigsten Fall helfen Rotationssensoren dabei, die tatsächliche Anzahl der Umdrehungen festzustellen, um die Genauigkeit der Schätzung zu erhöhen. Genauso kann die Position bei Kollisionen mit dem Holzrahmen korrigiert werden, und der Roboter kann mit dem Lichtsensor die Richtung anpeilen, in der die Lampe sich befindet. Diese Korrekturen erfordern einiges an Aufwand, sowohl in der Programmierung als auch in Form von zusätzlichen Fahrbewegungen des Roboters. Trotz allem bleibt das Verfahren sehr fehleranfällig. Im Sortiment der normalen Mindstorms-Sensoren gibt es aber keine bessere Möglichkeit, diese Position zuverlässig und absolut zu bestimmen.

1.1 Aufgabenstellung

Das Ziel dieser Arbeit ist, ein besseres Ortungssystem für Lego Mindstorms zu entwickeln. Mit diesem sollte es möglich sein, die Position eines Roboters innerhalb eines abgesteckten Areals möglichst genau zu bestimmen, und zwar basierend auf aktuellen Meßdaten und nicht auf vorherigen Bewegungen des Roboters. Die Ortung selber soll mit Hilfe von Ultraschallimpulsen erfolgen, es können also Schallimpulse zwischen dem Roboter und festen Basisstationen hin- und hergesendet werden. Dabei wird die Zeit gemessen, die der Schall für das Zurücklegen der jeweiligen Strecke braucht. Aus dieser Zeit und

der Schallgeschwindigkeit wird die Entfernung vom Roboter zur jeweiligen Basisstation berechnet, und aus den Entfernungen zu mehreren Basisstationen schließlich die Position des Roboters.

Zur Erfüllung der Aufgabe muß eine Reihe von implizit enthaltenen Problemen gelöst werden. Beispielsweise müssen Ultraschallimpulse erzeugt und detektiert werden, und es ist nötig, ihre Ausbreitungsrichtung über spezielle Reflektoren zu beeinflussen. Teile des Systems sollen direkt an den RCX angeschlossen werden und müssen daher kompatibel zu dessen Schnittstellen sein, sowohl auf der Ebene der Elektronik als auch bei der Übertragung der Meßwerte. Die Messungen und Steueraufgaben im System haben sich als so aufwendig herausgestellt, daß Mikrocontroller an vielen Stellen des Systems eingesetzt werden müssen. Die Basisstationen müssen über ein Netzwerk verbunden und koordiniert werden, und irgendwo im System werden die Koordinaten des Roboters berechnet. Nicht zuletzt müssen die Geräte auch hergestellt werden, was die Produktion von Platinen und Gehäusen nötig macht. Daher beschäftigen sich Teile dieser Arbeit auch mit diesen Themen.

Die Entwicklung des Ortungssystems endet mit der Implementierung einer Schnittstelle, über die der Benutzer alle Meßdaten des Systems abfragen kann. Die Daten sind umfangreich und lassen genug Spielraum für die Umsetzung verschiedener Lösungsansätze. Kapitel 4 liefert die mathematischen Grundlagen, wie aus den Meßdaten die Koordinaten berechnet werden können. Die Umsetzung in Programmcode erfordert zusätzliche Techniken, die in Kapitel 8 an Beispielen erläutert werden. Es ist aber nicht Thema dieser Arbeit, möglichst effiziente Strategien zur Fehlerkorrektur oder zur Navigation im Allgemeinen zu entwickeln. Diese Aufgabe fällt den späteren Benutzern zu, beispielsweise im Rahmen der Übungen und Praktika am Lehrstuhl Echtzeitsysteme und Eingebettete Systeme.

1.2 Vergleichbare Systeme

Das Problem der Ortsbestimmung mit Ultraschall ist nicht neu, daher gibt es bereits eine Vielzahl anderer Arbeiten zu dem Thema, die überwiegend auf der selben Grundidee aufbauen. Dabei werden in der Umgebung des Roboters Basisstationen an festen Positionen aufgestellt. Durch aktive Messungen wird die Entfernung des Roboters von allen Basisstationen festgestellt. Aus den Entfernungen und den Positionen der Basisstationen können per Trilateration die Koordinaten berechnet werden. Ultraschall eignet sich für Ortsbestimmungen im kleinen Maßstab besonders gut, da die Impulse sich verhältnismäßig einfach erzeugen und detektieren lassen. Die Ausbreitungsgeschwindigkeit ist niedrig genug, so daß die Zeitmessungen nur auf einige Mikrosekunden genau erfolgen müssen. Andererseits ist der Schall schnell genug, so daß eine Ortung relativ zügig erfolgen kann.

Die Messung kann auf zwei verschiedene Weisen erfolgen. Die erste Möglichkeit besteht darin, einen Schallimpuls vom Roboter zu allen Basisstationen zu senden. Diese messen die Laufzeit des Schalls und berechnen daraus die Entfernung und schließlich die Koordinaten des Roboters. Der Vorteil dieses Verfahrens liegt darin, daß nur ein Impuls benötigt wird, was die Ortung sehr schnell macht. Andererseits müssen die Ergebnisse an den Roboter zurückgesendet werden, und die Ortung mehrerer Roboter erfordert den Einsatz unterschiedlicher Frequenzen oder einer Zugriffskontrolle, die gleichzeitiges Senden zweier Roboter verhindert. Bei der zweiten Möglichkeit senden alle Basisstationen nacheinander einen Impuls zum Roboter. Dieser ist jetzt nur passiver Beobachter und kann seine Koordinaten selbst und ohne weitere Interaktion mit den Basisstationen berechnen. Dadurch können im selben Aufbau beliebig viele Roboter gleichzeitig eingesetzt werden. Dafür dauert ein Ortungsvorgang deutlich länger und es muß eine Möglichkeit vorgesehen werden, wie die Roboter die eingehenden Impulse eindeutig einer Quelle zuordnen können.

In beiden Fällen steht der Empfänger vor dem Problem, den genauen Sendezeitpunkt herausfinden zu müssen, damit er die Laufzeit berechnen kann. Dazu können die Uhren aller Sender und Empfänger synchronisiert und die Sendezeitpunkte über ein bekanntes Zeitschema vorgegeben werden. Wesentlich

einfacher ist es, unmittelbar vor dem Senden des Ultraschalls ein Synchronisationssignal zu übertragen, wofür Funk- und Infrarotsignale sich besonders gut eignen. Dadurch steht der Sendezeitpunkt fest und das zusätzliche Signal kann Informationen wie eine Identifikation des Senders tragen. Zwei Schallimpulse dürfen auch nicht zu schnell nacheinander gesendet werden, vorher muß das Echo des ersten Impulses sich stark genug abgeschwächt haben. Dieser Faktor begrenzt zusammen mit der relativ niedrigen Schallgeschwindigkeit jedes System auf wenige Ortungsvorgänge pro Sekunde.

Neben diesem Schema gibt es noch andere Ansätze für eine Ortung mit Hilfe von Ultraschall. Der Roboter kann beispielsweise einen Impuls aussenden, das Echo mit mehreren Sensoren auffangen und daraus ein Bild der Umgebung berechnen. Erkennt er darin markante Punkte, schließt er aus deren Koordinaten auf seine eigene Position. Diese Methode ist allerdings sehr schwierig und fehleranfällig, so daß sie in der Praxis wenig eingesetzt wird. Sie findet in vereinfachter Form eine Anwendung bei Einparkhilfen für PKWs. Bei der Ortsbestimmung für Roboter mit Ultraschall wird fast ausschließlich auf die Laufzeitmessung zurückgegriffen, folglich gibt es viele Variationen davon. Eine Arbeitsgruppe an der Universität von Kobe hat ein Ortungssystem entwickelt, das einen mobilen Haushaltsroboter in einem Zimmer lokalisieren kann. Dazu sendet der Roboter gleichzeitig einen Ultraschall- und einen Infrarotimpuls in Richtung der Zimmerdecke. Dort befinden sich mehrere miteinander verbundene Empfänger, die nach jedem Impuls die Position des Roboters auf 5 cm genau bestimmen können (siehe [Ghidary]).

Den entgegengesetzten Ansatz verwendet die Firma IS Robotics für die Positionsbestimmung ihrer Roboter aus der Genghis-Serie (vergleiche [Borenstein]). Hier sind die Rollen von Sender und Empfänger vertauscht, zwei Basisstationen stehen am Rand des abzudeckenden Feldes in festem Abstand und senden abwechselnd Ultraschallimpulse. Ein Funksignal wird für die Synchronisation benutzt, das System führt zwei Ortsbestimmungen pro Sekunde durch und erreicht in einem Areal von 9,1 m x 9,1 m eine Genauigkeit von 12,7 mm. Das System wird für 10.000 Dollar kommerziell vertrieben.

Andere Arbeiten beschäftigen sich mit Koordinatenberechnung und Fehlerminimierung. [Mahajan] formalisiert die Berechnung der Koordinaten und gibt ein allgemeingültiges Verfahren hierfür an, das vielfach benutzt wird. [Kleeman] beschreibt, wie die Ultraschallmessung mit Koppelnavigation kombiniert werden kann, um eine höhere Genauigkeit zu erreichen. Dabei wird auch mathematische Filterung eingesetzt, unter anderem in Form von Kalman-Filtern.

Abbildung 1.3: Roboter mit Ortungssystem aus [Navarro]

Die Basisstationen müssen auch nicht zwangsläufig an festen Positionen stehen. [Navarro] geht von einem Team von Robotern aus, von denen einige dynamisch zu Basisstationen gemacht werden. Diese Roboter bleiben stehen und dienen den anderen zur Orientierung. Der Rest kann sich in der Umgebung frei bewegen, bis das Team entscheidet, die Rollenverteilung zu ändern. Dadurch lassen sich auch größere Bereiche abdecken, als dies mit festen Basisstationen möglich wäre.

Es ist eine Herausforderung, die genannten Konzepte auf Mindstorms zu übertragen, vor allem weil der RCX nicht für den Anschluß komplizierter Peripherie gedacht ist. Ein Sensor muß mit sehr wenig Strom

auskommen und kann lediglich einen Analogwert an den RCX weitergeben, der in eher unregelmäßigen Abständen ausgelesen wird. Außerdem verfügt der RCX nur über relativ wenig Rechenleistung, so daß aufwendige Berechnungen sehr schwierig sind. Eine Projektgruppe eines luxemburgischen Internats (Convict Episcopal de Luxembourg, [Restena]) hat eine Lösung entwickelt, die direkt auf Mindstorms aufbaut und das System um Sender und Empfänger erweitert.

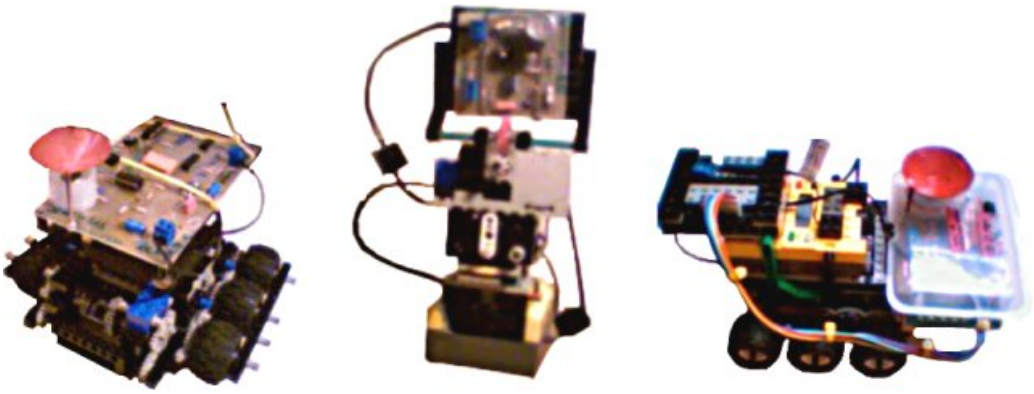


Abbildung 1.4: Ortungssystem des Convict Episcopal de Luxembourg

In der Mitte der Abbildung 1.4 ist eine sendende Basisstation zu sehen, wovon insgesamt drei benötigt werden. Die Roboter links und rechts davon tragen den dazu passenden Empfänger. Eine Ortsbestimmung beginnt damit, daß der Roboter die erste Basis auffordert, einen Impuls zu senden. Diese dreht ihren Sender in die vermutete Richtung des Roboters und strahlt gleichzeitig einen Infrarot- und einen Ultraschallimpuls ab. Der Empfänger auf dem Roboter registriert beide Impulse und mißt den Laufzeitunterschied mit Hilfe eines Mikrocontrollers. Das Ergebnis wird über einen Digital-Analog-Wandler in einen vom RCX lesbaren Sensorwert umgewandelt. Der Vorgang wiederholt sich für alle Basisstationen, dann berechnet der Roboter seine Koordinaten. Sie werden an die Stationen in der Umgebung übermittelt, damit diese sich im nächsten Durchlauf gezielter ausrichten können. Das System zeichnet sich durch einige positive Eigenschaften aus.

- Alle zusätzlichen Bauteile sind als Sensoren und Aktoren ausgelegt und integrieren sich so nahtlos in das Mindstorms-System.
- Der Ultraschallempfänger ist durch den kegelförmigen Reflektor richtungsunabhängig. Horizontal ankommende Schallwellen aus jeder Richtung werden durch ihn in die Detektorkapsel umgeleitet.
- Das Ausrichten der Basisstationen auf den Roboter demonstriert sehr elegant die Funktion des Systems, die reinen Koordinaten sind schließlich nicht sichtbar.

Daneben gibt es auch einige konstruktionsbedingte Schwächen, die sich negativ bemerkbar machen.

- Für einen Minimalaufbau werden vier RCX benötigt.
- Das System ist für den Einsatz mit der Lego-Software entwickelt. Diese unterstützt aber nur Ganzzahlen mit eingeschränktem Wertebereich, was zu deutlichen Rundungsfehlern führt.
- Das Ausrichten der Sender ist mechanisch aufwendig und fehleranfällig.
- Die Infrarotübertragungen von RCX und dem Ortungssystem finden bei sehr ähnlichen Frequenzen statt (38 kHz und 40 kHz) und können sich daher stören. Außerdem verlangsamt das Aktivieren per Infrarot den Vorgang unnötig.
- Die gleichzeitige Ortung mehrerer Roboter ist nur eingeschränkt möglich.

- Die Elektronik enthält einige Designfehler. Sie ist unnötig kompliziert, störanfällig und verbraucht relativ viel Strom. Die Ultraschallmodule reizen ihr Potential nur zu einem kleinen Teil aus, es werden weder die maximal mögliche Sendestärke noch eine hohe Empfindlichkeit erreicht.
- Die gesamte Konstruktion ist zu aufwendig und verbraucht sehr viel Platz.
- Bei der Datenübertragung zum RCX wird ein erheblicher Aufwand betrieben, trotzdem können nicht genug Daten für eine präzise Messung mit hoher Reichweite übertragen werden. Die Genauigkeit der einfachen Entfernungsmessung liegt bei 4,4 cm, was eine schlechte Grundlage für eine präzise Koordinatenberechnung darstellt.
- Eine vollständige Ortsbestimmung dauert mindestens eine halbe Sekunde.

Das System funktioniert, allerdings summieren sich die Wirkungen einiger Schwächen auf, so daß die berechneten Koordinaten mit einem erheblichen Fehler behaftet ist. Abbildung 1.5 zeigt die berechnete Spur eines Roboters, der ein Quadrat mit einer Kantenlänge von 80 cm abgefahren ist. Die Darstellung basiert auf Daten von [Restena].

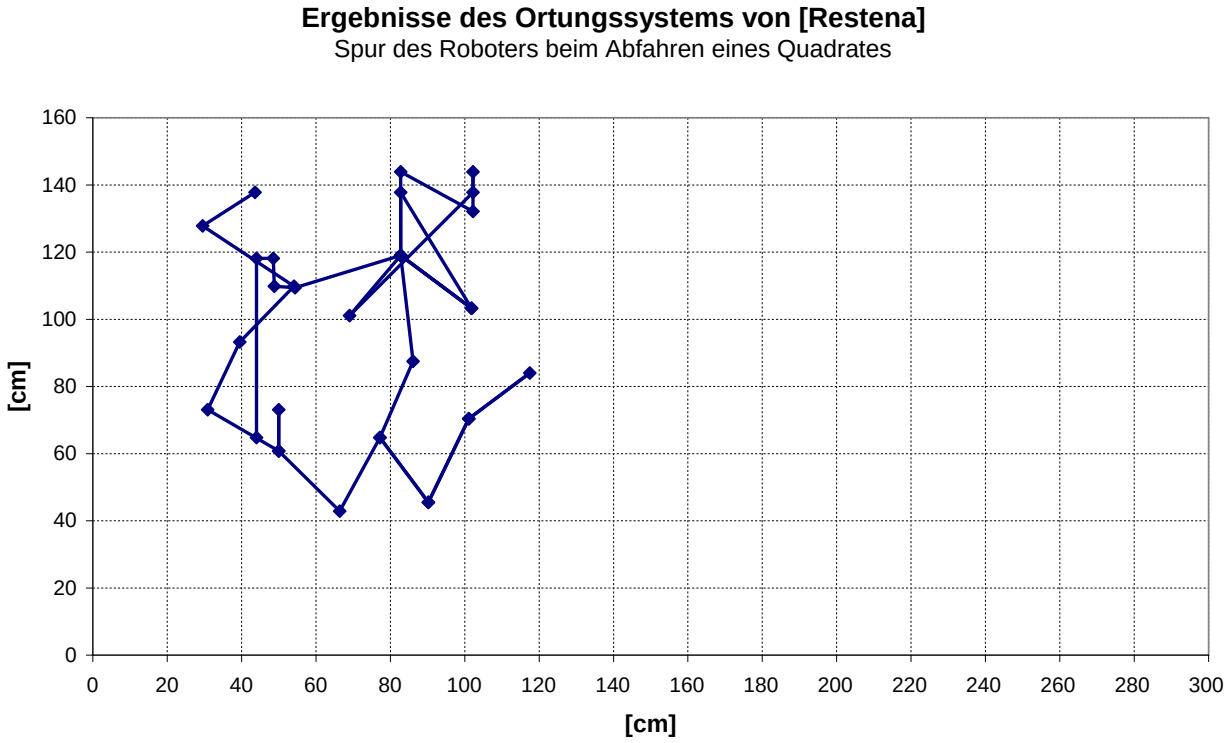


Abbildung 1.5: Ergebnisse des luxemburgischen Ortungssystems aus [Restena]

Diese Arbeit greift einige Ideen des luxemburgischen Systems auf und verbessert sie weiter. Die technischen Realisierungen haben aber sehr wenig gemein und das Design unterscheidet sich an vielen Schlüsselstellen deutlich. Die Schwerpunkte bei der Entwicklung waren:

- Hohe Genauigkeit und große Reichweite
- Robustheit, sowohl in der Handhabung als auch beim Auftreten von Fehlern
- Geschwindigkeit, ein kompletter Ortungsvorgang muß möglichst schnell beendet sein.
- Konfigurierbarkeit, das System soll auf möglichst viele Arten einsetzbar sein.
- Einfachheit, ein Benutzer soll das System ohne große Einarbeitung benutzen können.

All diese Ziele konnten erreicht werden. Im Laufe der Arbeit ist ein gut funktionierendes System mit ausreichenden technischen Daten für die meisten denkbaren Aufgaben entstanden. Auf dem Weg dahin sind aber auch viele Informationen zusammengetragen und Entwicklungswerkzeuge hergestellt worden, mit denen sich auch einige der folgenden Abschnitte beschäftigen werden.

1.3 Gliederung dieser Arbeit

Der Grundlagenteil am Anfang dieser Arbeit beschäftigt sich mit allen technischen Voraussetzungen für die Realisierung des Ortungssystems. Kapitel 2 enthält eine der vielleicht umfassendsten Übersichten zur Entwicklung von Mindstorms Sensoren und Aktoren sowie zur Funktionsweise der Infrarotkommunikation. Der Leser soll durch das Kapitel in die Lage versetzt werden, beliebige Peripherie für Mindstorms zu entwickeln.

Kapitel 3 beschäftigt sich mit dem Einsatz vom Mikrocontrollern, speziell aus der PIC-Serie von Mikrochip. Diese integrierten Schaltungen sind für viele Funktionen des Ortungssystems unverzichtbar. In dem Kapitel wird ein komplettes Entwicklungssystem vorgestellt, das viele wichtige Werkzeuge und Schaltungen zum Programmieren, Testen und zur Fehlersuche enthält.

Mit der Entfernungsmessung per Ultraschall und der Koordinatenberechnung beschäftigt sich Kapitel 4. Neben den grundlegenden Techniken werden Methoden und Formeln zur Bestimmung der Schallgeschwindigkeit und zur Berechnung der Reflektorgeometrie angegeben. Die Formeln für die Koordinatenberechnung werden hergeleitet, eine numerische Fehlersimulation zeigt die Stärken und Schwächen verschiedener Ansätze.

Die Kapitel 5, 6 und 7 stellen das eigentliche Ortungssystem hierarchisch vor, erklären seinen Aufbau und begründen die wichtigsten Designentscheidungen. Das darauf folgende Kapitel 8 faßt alle Informationen zusammen, die ein Benutzer des Systems braucht, um damit arbeiten zu können. Ein Schwerpunkt liegt dabei auf der Implementierung von Algorithmen zur Koordinatenberechnung. Am Ende werden in Kapitel 9 die technischen Daten zusammengefaßt und bewertet.

Die Anhänge bündeln einen Großteil der Informationen, die zum Nachbau der Hardware und zum Einsatz der Software benötigt werden. In Anhang A sind alle Schaltungen aus der Arbeit mit den zum Nachbau nötigen Angaben aufgeführt. Unter anderem sind dies sämtliche Peripheriebausteine für Mindstorms, das PIC Entwicklungssystem mit Programmiergerät, Prototyp-Platinen und Debugging-Interface sowie die verschiedenen Module des Ortungsystems.

Anhang B enthält die Interfacebeschreibungen und kurze Bedienungsanleitungen für alle neu entwickelten Programme. Dazu gehört auch `lnphost`, eine Bibliothek zur Ansteuerung des Mindstorms-Towers, die den fehlerhaften und veralteten `lnpd` ersetzt.

Die Meßwerte zu den wichtigsten Diagrammen sind in Anhang C aufgeführt. Anhang D widmet sich der Herstellung gedruckter Platinen und beschreibt den gesamten Produktionsprozeß. Der Rest der Arbeit enthält das Literaturverzeichnis, Quellennachweise und Ähnliches.

Teil I

Grundlagen

Kapitel 2

Lego Mindstorms Elektronik

Lego Mindstorms ist ein auf Lego Technic aufbauendes System zur Konstruktion von Robotern. Zusätzlich zu den bekannten Bausteinen und Motoren gibt es bei Lego Mindstorms einen frei programmierbaren Computer (Robotic Command Explorer, kurz RCX) und eine Reihe von Sensoren, die es erlauben, autonome Roboter mit komplexem Verhalten zu bauen und zu programmieren. Darüber hinaus ist es relativ einfach möglich, eigene Sensoren und Aktoren zu entwickeln, um die Fähigkeiten des Roboters zu erweitern. Dabei müssen aber eine Reihe von Grundlagen beachtet werden, damit das fertige Bauteil auch mechanisch und elektronisch kompatibel zu Lego Mindstorms ist. Dieses Kapitel stellt diese Grundlagen dar und faßt dabei auch Informationen aus [Proudfoot], [Gasperi] und [Nelson] zusammen. Der Schwerpunkt liegt auf der Funktionsweise und dem Zusammenspiel der Elektronik von RCX und Peripherie, daneben werden auch Themen wie Infrarotkommunikation, Energieversorgung und Abmessungen der Legosteine behandelt.

2.1 Robotic Command Explorer

Der wichtigste Legostein im Mindstorms-System ist der RCX. Er ist das zentrale Bauteil eines jeden Roboters, das die angeschlossenen Sensoren und Aktoren ansteuert und ein vom Benutzer geschriebenes Programm ausführt.

Abbildung 2.1: Der RCX

Der RCX bezieht seine Energie aus 6 Batterien (Mignon-Zellen, Typ AA) oder einem Steckernetzteil und kann sich selbst sowie den ganzen Roboter mit Strom versorgen. An die Eingänge 1, 2 und 3 können Sensoren angeschlossen werden, während die Ausgänge A, B und C für den Anschluß von

Aktoren gedacht sind. In der Mitte des Gerätes befindet sich ein Display, das Informationen über den Zustand des RCX anzeigen kann. Dazu dienen fünf Ziffernanzeigen und eine ganze Reihe von Symbolen, die je nach Programm unterschiedlich genutzt werden. Um das Display herum sind vier Knöpfe angeordnet, mit denen der Benutzer den RCX einschalten, Programme auswählen und starten kann. An der Stirnseite befinden sich hinter geschwärztem Kunststoff je ein Sender und ein Empfänger zur Infrarot-Kommunikation. Auf diese Weise können Programme von einem PC auf den RCX übertragen sowie beliebige Daten mit anderen Geräten ausgetauscht werden. Neben all diesen nach außen sichtbaren Baugruppen verfügt der RCX noch über einen Lautsprecher, der Piepstöne in unterschiedlichen Frequenzen erzeugen kann.

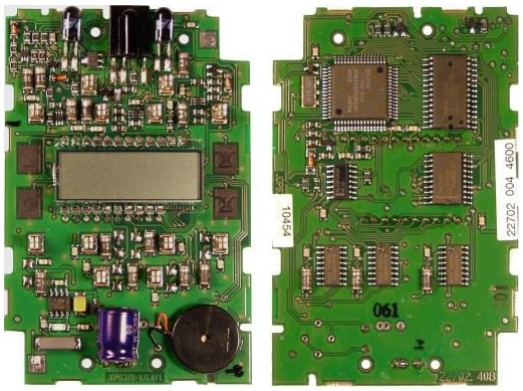


Abbildung 2.2: Platine im Inneren des RCX

Herzstück des RCX ist ein Hitachi H8/3292 Mikrocontroller, der mit 16 MHz getaktet wird und über 32 KByte ROM sowie 32 KByte RAM verfügt. Im ROM ist bereits eine umfangreiche Firmware vorhanden, die viele Interfaces zur Steuerung der Peripherie bereitstellt. Sogar einige einfache Steuerprogramme für Roboter gehören zum Funktionsumfang des Codes. Daneben kann der Programmcode auch ein Betriebssystem über die Infrarotschnittstelle in den RAM nachladen. Die Firma Lego liefert den RCX mit einem Bytecode-Interpreter als Betriebssystem aus, der mit einer visuellen Programmiersprache geschriebene Programme ausführen kann. Die freie C-ähnliche Sprache [NQC] von David Baum setzt ebenfalls auf diesem Interpreter auf. Daneben existiert mit [BrickOS] auch ein Betriebssystem, das unabhängig von dem Interpreter arbeitet und dem Benutzer vollen Zugriff auf alle Ressourcen des RCX erlaubt. Sowohl brickOS als auch der Lego-Interpreter nutzen die Interfaces der Firmware zur Steuerung der Hardware. Die Entwicklung von Programmen für den RCX und die Ansteuerung der Sensoren und Aktoren ist beispielsweise in [Legokiel] beschrieben, eine anschauliche Zusammenfassung der mechanischen Grundlagen von Lego Technic finden sich in [Martin].

2.2 Kabel und Anschlüsse

Die elektrischen Verbindungen zwischen dem RCX und seinen Peripheriegeräten werden über zweiadrige Kabel hergestellt, an deren Enden spezielle Legosteine als Stecker sitzen. Die Kabel werden durch einfaches Aufstecken angeschlossen, wobei Metallteile in den Steckern die nötigen Kontakte herstellen. Die Stecker sind so trickreich aufgebaut, daß es nahezu egal ist, wie herum sie angeschlossen werden. Es dreht sich lediglich die Polarität um, wenn ein Stecker um 180 Grad gedreht wird. Außerdem sind Kurzschlüsse ausgeschlossen, solange keine Schleifen in der Verkabelung enthalten sind.

Abbildung 2.3 zeigt die Metallkontakte in einem transparent gezeichneten Stecker. Jeder der Knöpfe auf der Oberseite enthält einen Kontakt, und auf der Unterseite befinden sich 4 Metallfedern an der

Innenwand des Steins. Alle vier Anschlüsse auf der einen Seite (A) sind mit einer Leitung des Kabels verbunden, alle Kontakte der anderen Seite (B) mit der anderen Leitung.

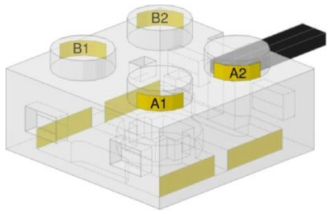


Abbildung 2.3: Aufbau eines Legosteckers

Der Nachbau dieses Steckersystems für Eigenentwicklungen ist nicht ganz einfach. Es besteht beispielsweise die Möglichkeit, zwei diagonal gegenüberliegende Knöpfe eines Legosteins durch passende Schrauben zu ersetzen oder dünne Metallfolie auf den Innenwänden eines Legosteins zu befestigen. Am einfachsten ist jedoch, ein Originalkabel in der Mitte durchzuschneiden und die beiden Hälften zum Anschluß des Bauteils zu benutzen.

2.3 Motoren und andere Aktoren

Der Begriff Aktor bezeichnet all diejenigen Peripheriegeräte, die Einfluß auf die Umwelt des Roboters nehmen statt sie wie Sensoren nur passiv zu beobachten. Bei Lego Mindstorms gehören insbesondere die Antriebsmotoren zu den Aktoren, daneben gibt es noch Lämpchen, Sirenen und eine Vielzahl von Eigenentwicklungen von Drittanbietern. Dennoch sind die Motoren die wichtigsten Aktoren, an denen sich viele der im Folgenden benutzten Begriffe orientieren. Der RCX verfügt über Ausgänge für drei Aktoren, die unabhängig voneinander gesteuert werden können. Jeder Ausgang kann von dem Treiberbaustein des RCX in einen von vier Modi geschaltet werden.

- **Vorwärts**: Der Aktor wird mit Gleichstrom versorgt.
- **Rückwärts**: Wie vorwärts, allerdings mit entgegengesetzter Polarität.
- **Aus**: Der Anschluß wird hochohmig geschaltet.
- **Bremsen**: Der Treiberbaustein schließt den Stromkreis mit dem Aktor kurz.

Der Bremsen-Modus ist nötig, um einen laufenden Motor möglichst schnell zum Stillstand zu bringen. Wird der Motor einfach ausgeschaltet, läuft er durch die mechanische Trägheit noch eine Weile nach, bis er schließlich stehen bleibt. Im Bremsen-Modus hingegen arbeitet der Motor als Generator. Durch seine Eigendrehung erzeugt er Strom, der durch den Kurzschluß in Wärme umgewandelt wird. Der Energieverlust bremst den Motor stark ab, so daß er praktisch sofort stillsteht und seine Achse sich nur mit einigem Kraftaufwand drehen läßt.

Neben der Drehrichtung des Motors kann auch dessen Geschwindigkeit per Pulsweitenmodulation eingestellt werden. Bei dieser Methode wird die Spannung am Ausgang in schnellem Wechsel ein- und ausgeschaltet. Je länger der Motor im zeitlichen Mittel mit Strom versorgt wird, desto schneller dreht er sich. Der Vorteil dieses Verfahrens liegt darin, daß weder die Spannung am Ausgang noch der fließende Strom verändert werden müssen, was die nötige Elektronik auf ein Minimum beschränkt. Statt dessen wird die gesamte Steuerung von der Software übernommen. Der RCX löst dazu jede Millisekunde einen Interrupt aus, der dann alle drei Aktorausgänge entsprechend schaltet. Der dabei zum Einsatz kommende Algorithmus hängt vom Betriebssystem des RCX ab.

2.3.1 Algorithmen für die Pulsweitenmodulation

Der Bytecode-Interpreter von Lego benutzt ein sehr einfaches Verfahren zur Pulsweitenmodulation. Das Anwendungsprogramm gibt den Zustand des Ausgangs (aus, bremsen, vorwärts, oder rückwärts) und die Geschwindigkeit n vor, wobei n zwischen 0 (langsam) und 7 (schnell) liegt. Basierend auf diesen Parametern wird der Ausgang wie folgt geschaltet.

- Ist der Ausgang im Zustand „aus“ oder „bremsen“, wird er direkt entsprechend geschaltet.
- In den anderen beiden Fällen wird die Pulsweitenmodulation benutzt. Grundlage ist ein Zeitfenster mit einer Länge von 8 ms, das sich periodisch wiederholt. Der Motor wird die ersten (n+1) ms des Zeitfensters mit Gleichstrom der entsprechenden Polarität versorgt, dir restliche Zeit ist der Ausgang auf „aus“ geschaltet.

Dieses Verfahren erlaubt nur wenige Geschwindigkeitsstufen und eine Steigerung ist nur schlecht möglich. In dem Fall würde das Zeitfenster zu breit und die Pulsweitenmodulation würde sich als Stottern des Motors bemerkbar machen.

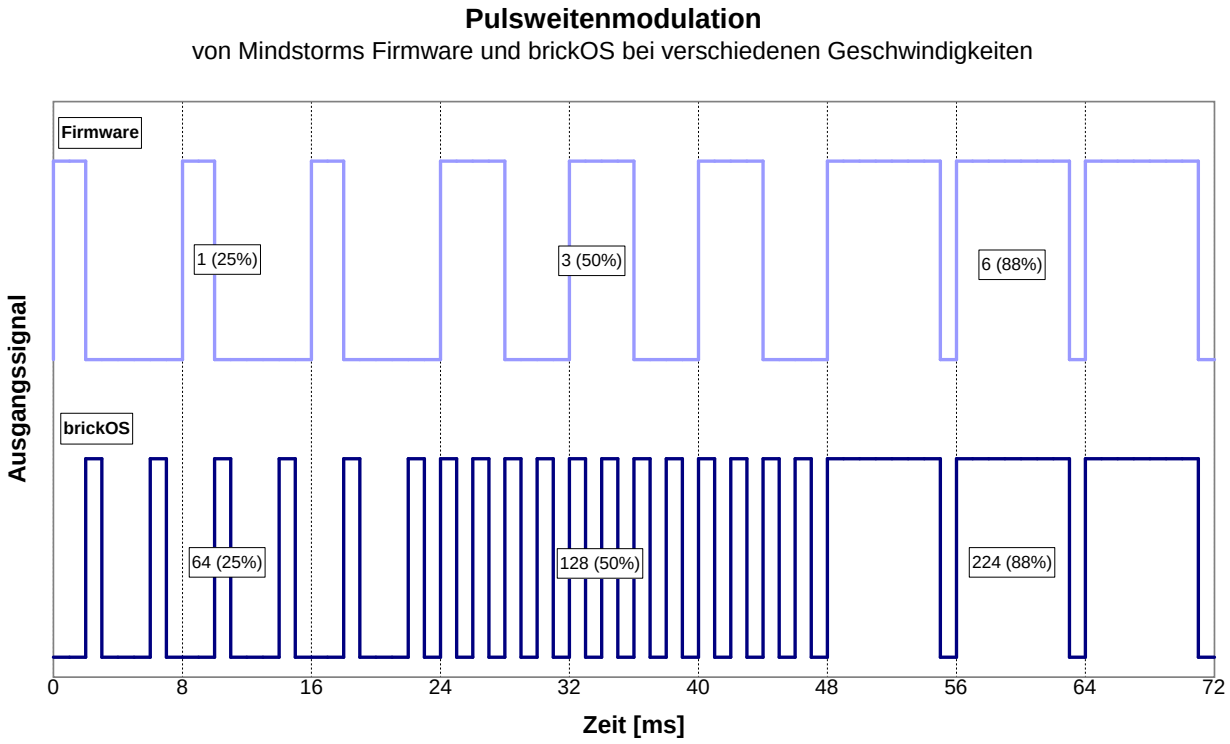


Abbildung 2.4: Schematische Darstellung der Pulsweitenmodulation

BrickOS benutzt einen anderes, auf dem Bresenham-Algorithmus basierendes Verfahren, das in der Kernel-Dokumentation von [Nielsson] erläutert ist. Dabei sind der Zustand des Ausgangs und die Geschwindigkeit von 0 (langsam) bis 255 (schnell) vorgegeben. Zusätzlich benutzt der Algorithmus intern eine Byte-Variable als Zähler, die mit 0 initialisiert wird.

- Jede Millisekunde wird (n+1) auf die Zählervariable addiert.
- Kommt es dabei zum Überlauf, wird der Ausgang in den gewünschten Zustand geschaltet (vorwärts, rückwärts, aus oder bremsen). Anderenfalls wird er auf „aus“ geschaltet.

Dieses Verfahren wird unter brickOS als Coast Mode PWM bezeichnet und ist standardmäßig aktiviert. Alternativ dazu kann auch Hold Mode PWM gewählt werden, wobei der Ausgang in den Pausen statt auf „aus“ auf „bremsen“ geschaltet wird. In jedem Fall erlaubt brickOS wesentlich mehr Abstufungen, dennoch macht sich ein Stottern des Motors nur bei sehr niedriger Geschwindigkeit bemerkbar. Außerdem ist es möglich, auch die Intensität des Bremsens einzustellen. Abbildung 2.4 stellt die simulierten Signalmuster der Pulsweitenmodulation beider Algorithmen für unterschiedliche Geschwindigkeiten gegenüber.

2.3.2 Drehmoment und Geschwindigkeit

Für das mechanische Design eines Roboters spielen die genauen Eigenschaften der Motoren eine große Rolle. Zum Teil handelt es sich dabei um das maximale Drehmoment und die Leerlaufdrehzahl, die in [Hurbain] für eine Vielzahl von Lego-Motoren untersucht und verglichen werden. Vor allem ist aber interessant, wie schnell sich ein Motor dreht, wenn die Belastung und die per Pulsweitenmodulation eingestellte Geschwindigkeit bekannt sind. Diese Größe ist beispielsweise wichtig, um das Beschleunigungsverhalten eines Roboters modellieren zu können. [Janz] untersucht den Lego-Motor vom Typ 71427 unter diesen Gesichtspunkten.

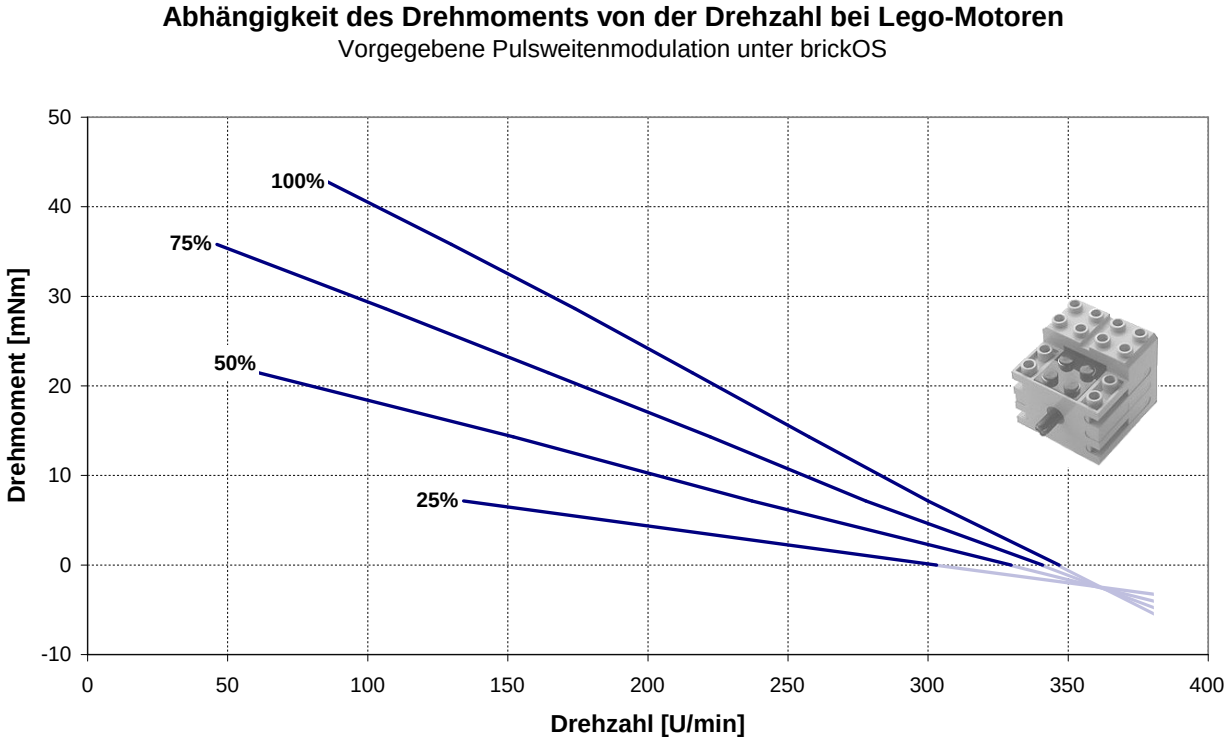


Abbildung 2.5: Drehmoment des Lego-Motors 71427 bei verschiedenen Geschwindigkeiten

Die Abbildung 2.5 zeigt die Ergebnisse dieser Untersuchung. Der Motor wurde an einen unter brickOS laufenden RCX angeschlossen und nacheinander mit verschiedenen Geschwindigkeiten angesteuert. Für jede Einstellung wurde die Abhängigkeit des Drehmoments von der Drehzahl ermittelt. Daraus resultiert eine Geradenschar, die sich in einem Punkt mit negativem Drehmoment schneiden. Dieser Wert entspricht genau dem Reibungsverlust des Getriebes im Motor.

2.3.3 Grenzen der Treiberbausteine

Die Aktorausgänge des RCX sind so ausgelegt, daß sie ausreichend hohe Ströme für den Betrieb der mitgelieferten Motoren liefern können (beispielsweise bis zu 360 mA für den 71427). Dennoch gibt es einige Schutzschaltungen, die den RCX und die Motoren vor Überbelastung schützen sollen. So sind die Ausgänge der Motortreiber-ICs auf maximal 500 mA begrenzt, wie die in Abbildung 2.6 dargestellte Messung eines Ausgangs zeigt.

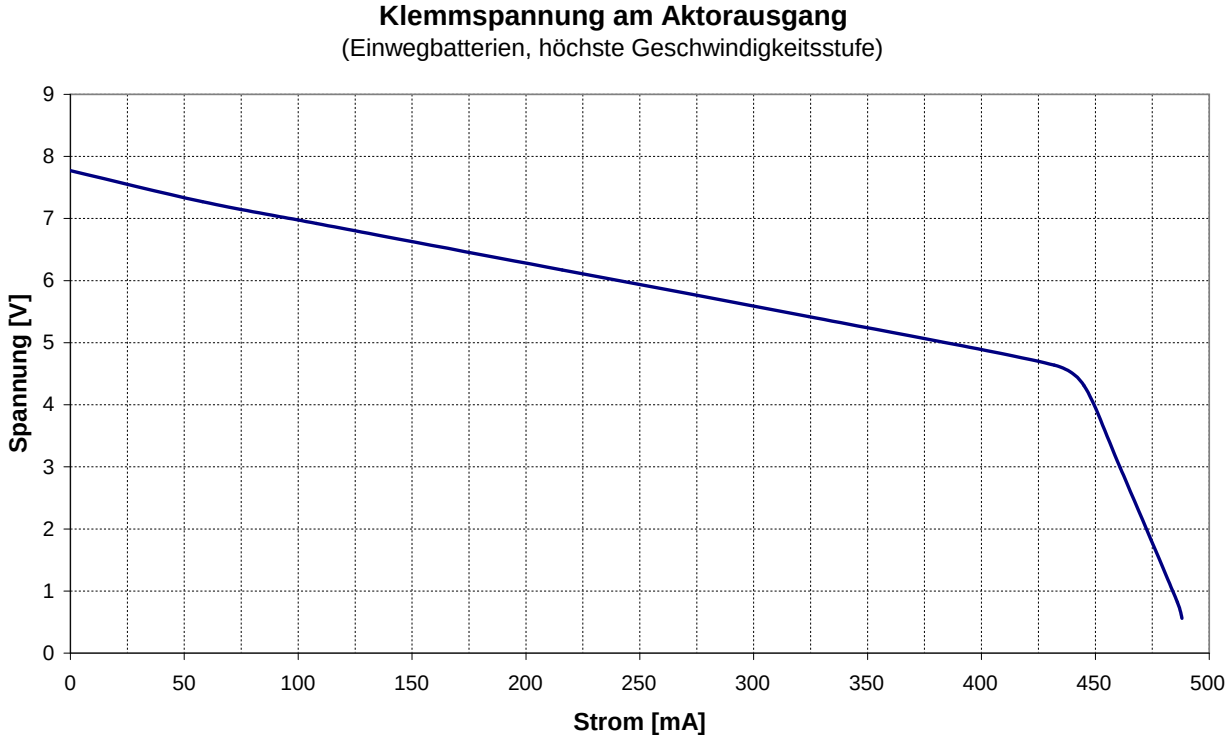


Abbildung 2.6: Verhalten eines Aktorausgangs bei Belastung

Solange der entnommene Strom niedrig genug ist, erscheint der Aktorausgang wie eine Spannungsquelle mit einem Innenwiderstand von unter 8 Ω. Bei 430 mA greift die Begrenzung und reduziert die Spannung so stark, daß es nicht möglich ist, mehr als 500 mA zu entnehmen. Dementsprechend ist der RCX selber mit einer internen Sicherung von 1,5 A abgesichert.

Ein Schutz vor Überhitzung ist in allen Motortreibern und in den Motoren selbst eingebaut. Der Motortreiber schaltet ab, sobald er zu heiß wird, und aktiviert sich nach dem Abkühlen wieder von selbst. Die Motoren sind mit einem PTC-Widerstand abgesichert, der bei steigender Innentemperatur den fließenden Strom effektiv reduziert.

2.3.4 Entwicklung eigener Aktoren

Es ist sehr einfach, eigene Aktoren zu entwickeln, da sie in der Regel den RCX nur als Stromquelle benutzen. So können zum Beispiel einfache Bauteile wie Lämpchen, Elektromagneten und Motoren direkt oder über einen Vorwiderstand angeschlossen werden. Etwas schwieriger ist es, wenn der Aktor eine stabilisierte Spannungsversorgung benötigt. Dann machen sich drei Probleme bemerkbar.

- Die Polarität der Versorgungsspannung hängt davon ab, wie herum die Stecker des Verbindungskabels aufgesteckt sind. Wenn die Schaltung also auf eine bestimmte Polarität angewiesen ist, muß ein Brückengleichrichter diese sicherstellen.

- Die Pulsweitenmodulation kann dafür sorgen, daß die Versorgungsspannung immer wieder für einige Millisekunden ausfällt. Je niedriger die Geschwindigkeit, desto häufiger und länger sind diese Unterbrechungen. Daher muß ein Kondensator im Aktor eingesetzt werden, der genügend Energie speichern kann, um die Pausen zu überbrücken. Abhängig von dem Strombedarf des Aktors, der maximalen Unterbrechungszeit und den tolerierbaren Spannungsschwankungen kann durchaus eine Kapazität von einigen hundert oder tausend Mikrofarad erforderlich sein. Formel (2.1) liefert eine gute Näherung für die Mindestgröße des Kondensators.

$$C \geq \frac{I \cdot \Delta t}{\Delta U} \tag{2.1}$$

  Dabei ist I die Stromaufnahme der Schaltung, Δt die zu überbrückende Zeit und ΔU der maximal tolerierbare Spannungsabfall während der Versorgungslücke.

- Eventuell muß die Spannung in einem letzten Schritt noch begrenzt oder auf einen bestimmten Wert stabilisiert werden. Dies ist für alle Schaltungen nötig, die nur bei bestimmten Schaltungen wie gewünscht funktionieren.

Bei der Entwicklung eines Aktors für Mindstorms stellt sich in erster Linie die Frage, ob Spannung und Strom aus dem RCX für den sicheren Betrieb der Schaltung ausreichen. Die Spannung ist insbesondere bei fast leeren Batterien schon relativ gering. Davon müssen noch der Spannungsabfall über dem Gleichrichter (bis zu 1,4 V) und dem Spannungsregler der Schaltung (z.B. 2 V bei einem 78L05) abgezogen werden. Der Rest muß für einen stabilen Betrieb des Aktors ausreichen.

Ist dies nicht der Fall, kann der Spannungsregler durch einen effizienteren Typen (Low Power, Low Dropout, Low Quiescent Current) ersetzt werden. Beispielsweise arbeitet der LP2950 im Gegensatz zum 78L05 mit einer Spannungsdifferenz von 20 mV noch stabil. Bei niedrigem Stromverbrauch lohnt es sich auch, den Gleichrichter aus Schottky-Dioden aufzubauen, was den Spannungsverlust über dem Gleichrichter auf 0,6 V verringern kann. Die Schaltung in Abbildung 2.7 zeigt eine vielseitig verwendbare Grundschaltung für den Aufbau von Aktoren. Sie ist zum Beispiel geeignet, einen Mikrocontroller mit Peripherie zu versorgen.

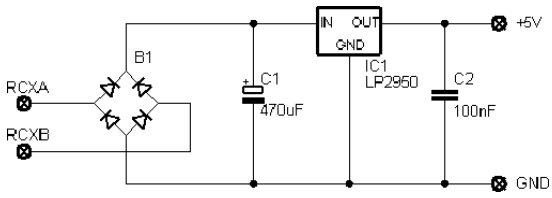


Abbildung 2.7: Grundschaltung für Aktoren

Wenn der RCX im Display anzeigt, daß die Batterien fast leer sind, schafft diese Schaltung es mit Standarddioden immer noch, einen Strom von 30 mA bei stabilisierten 5 V zu liefern. Erst kurz vor der Selbstabschaltung des RCX funktioniert die Regelung nicht mehr.

2.4 Passive Sensoren

Im RCX existieren vier Anschlüsse für Sensoren, die regelmäßig reihum abgefragt werden. Drei der Anschlüsse sind nach außen geführt und können mit Mindstorms-Sensoren versehen werden, während der vierte die Batteriespannung mißt. Die externen Sensoren fallen in zwei Kategorien. Dies sind zum einen die passiven Sensoren, die veränderliche elektrische Widerstände darstellen und keine zusätzliche Versorgungsspannung brauchen. Dazu zählen der Taster und der Temperatursensor von Lego, genauso

funktionieren zum Beispiel Fotowiderstände, PTCs oder Potentiometer. Auf der anderen Seite stehen die aktiven Sensoren, die vom RCX mit Strom versorgt werden müssen. Das bekannteste Beispiel ist der Lichtsensor, dessen Leuchtdiode mit Strom versorgt werden muß. Genauso benötigt die Platine im Inneren des Rotationssensors Strom, um die Drehbewegungen mit Hilfe von Lichtschranken erfassen und auswerten zu können. Viele Sensoren von Drittanbietern sind ebenfalls auf eine Stromversorgung angewiesen. Jeder Sensoreingang muß passend zum Typ des angeschlossenen Sensors für den aktiven oder passiven Modus konfiguriert sein.

Der RCX steuert passive Sensoren nach dem in Abbildung 2.8 dargestellten Schema an (vergleiche [Gasperi]). Aus einer Spannungsquelle von 5 V fließt der Strom I erst durch einen Widerstand von 10 kΩ und dann durch den eigentlichen Sensor. Ein Analog-Digital-Wandler mißt die über dem Sensor abfallende Spannung U_{ADC}.

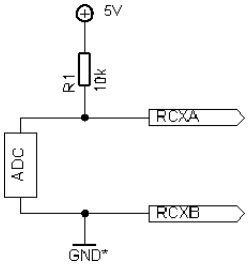


Abbildung 2.8: Ansteuerung von passiven Sensoren im RCX

Der Querstrom durch den Wandler ist klein genug, daß er vernachlässigt werden kann. Damit kann I aus den ohmschen Widerständen berechnet werden.

$$I = \frac{5\text{V}}{10\text{k}\Omega + R_{\text{Sensor}}} \tag{2.2}$$

Die Spannung über dem Analog-Digital-Wandler entspricht 5 Volt minus dem Spannungsabfall über dem 10 kΩ-Widerstand im Inneren des RCX.

$$\begin{aligned} U_{\text{ADC}} &= 5\text{V} - I \cdot 10\text{k}\Omega \\ &= 5\text{V} - \frac{5\text{V} \cdot 10\text{k}\Omega}{10\text{k}\Omega + R_{\text{Sensor}}} \\ U_{\text{ADC}} &= \frac{5\text{V} \cdot R_{\text{Sensor}}}{10\text{k}\Omega + R_{\text{Sensor}}} \end{aligned} \tag{2.3}$$

Der Analog-Digital-Wandler mißt U_{ADC} mit einer Genauigkeit von 10 Bit. Sein Ausgabewert N ergibt sich durch eine lineare Skalierung von U_{ADC}, so daß 5 Volt auf 1023 abgebildet wird.

$$N = 1023 \cdot \frac{U_{\text{ADC}}}{5\text{V}}$$

$$N = 1023 \cdot \frac{R_{\text{Sensor}}}{10\text{k}\Omega + R_{\text{Sensor}}} \tag{2.4}$$

$$R_{sensor} = \frac{N}{1023 - N} \cdot 10\text{k}\Omega. \tag{2.5}$$

Der Sensorwert ist demnach proportional zu Spannung und Strom, die am Sensoreingang gemessen werden, nicht jedoch zum Widerstand des Sensors. Dieser Punkt ist beim Design von Sensoren, die analoge Werte liefern sollen, wichtig. Ein Sensor muß den fließenden Strom linear verändern, um lineare Änderungen der Meßwerte im RCX hervorzurufen.

Abbildung 2.9 stellt in zwei Kurven den Zusammenhang zwischen Sensorwert, Spannung und Strom dar. Die Daten basieren auf einer Vergleichsmessung und bestätigen die hergeleiteten Formeln.

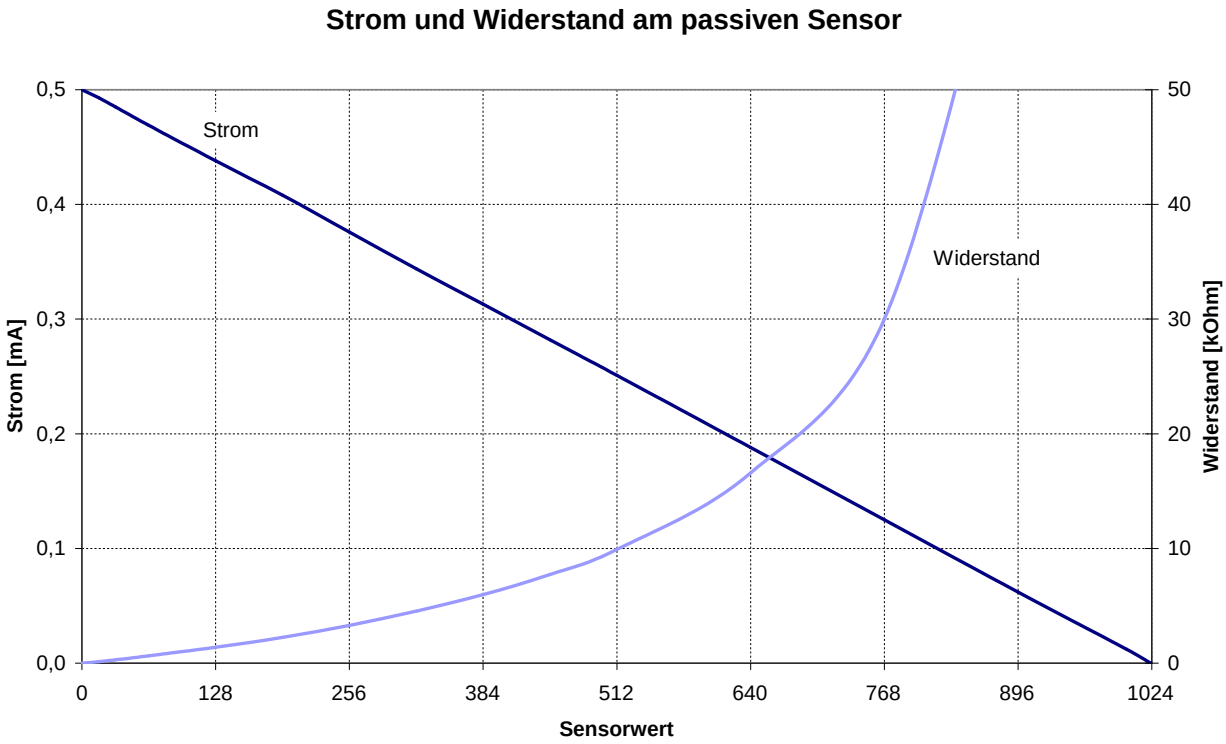


Abbildung 2.9: Sensorwerte, Strom und Widerstand im passiven Modus

2.5 Aktive Sensoren

Die passiven Sensoren stoßen schnell an ihre Grenzen. Für viele Meßaufgaben ist eine zusätzliche Stromversorgung nötig, um beispielsweise einen Verstärker, Detektor oder andere Baugruppen betreiben zu können. Diese Aufgaben lassen sich mit aktiven Sensoren realisieren, die vom RCX mit Strom versorgt werden. Lego hat entschieden, für diese Sensoren keinen neuen Kabeltyp einzuführen, so daß die vorhandenen zweiadrigen Kabel und Stecker per Multiplexing abwechselnd zur Stromversorgung und zum Abfragen des Sensorwertes genutzt werden.

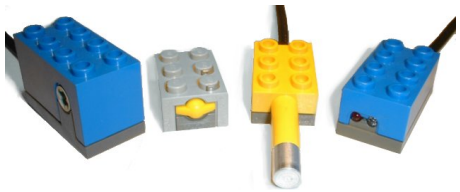


Abbildung 2.10: Standard Mindstorms-Sensoren

Diese Tatsache sorgt dafür, daß die Sensorschaltung über einen geeigneten Demultiplexer mit dem RCX verbunden werden muß. Glücklicherweise ist dieser relativ einfach zu bauen und besteht im Wesentlichen nur aus wenigen Dioden und einem Kondensator.

2.5.1 Ansteuerung durch den RCX

Aktive Sensoren werden vom RCX fast genau wie passive Sensoren angesteuert, es kommt lediglich noch eine schaltbare Spannungsquelle hinzu, wie das Ersatzschaltbild in Abbildung 2.11 zeigt.

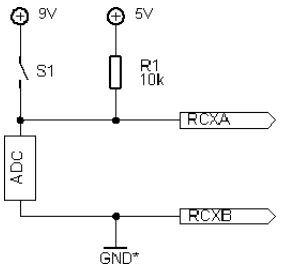


Abbildung 2.11: Ansteuerung von aktiven Sensoren im RCX

Zum Auslesen des Sensors wird Schalter S_1 geöffnet. Die Messung des Sensorwertes funktioniert genau wie bei passiven Sensoren, die im vorigen Abschnitt genannten Formeln gelten also auch für aktive Sensoren. In der übrigen Zeit wird der Sensor mit Strom versorgt, wozu der Schalter S_1 geschlossen wird. Der RCX wechselt ständig schnell zwischen Versorgungsmodus und Auslesemodus hin und her.

Im Sensor muß eine Interfaceschaltung wie in Abbildung 2.12 diese beiden Funktionen wieder trennen.

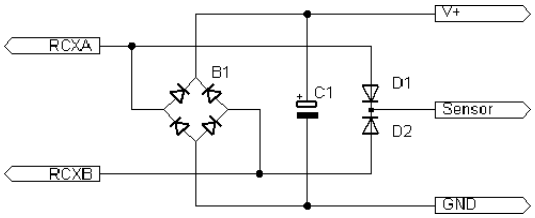


Abbildung 2.12: Grundschaltung für einen aktiven Sensor

Während der Versorgungsphase fließt der Strom über den Brückengleichrichter, lädt den Kondensator C_1 auf und versorgt den Rest des Sensors. Beginnt der RCX dann mit dem Auslesen, schaltet er die Versorgungsspannung ab, wodurch die Spannung am Sensor auf maximal 5 V absinkt. Damit sperren die oberen beiden Dioden im Brückengleichrichter, weil das Potential hinter ihnen durch den Kondensator auf über 5 V gehalten wird. Der Auslesestrom kann jetzt nur durch eine der Dioden D_1 oder D_2 fließen, von wo er über den Sensorwiderstand zum Masseanschluß und damit zurück in den RCX geleitet wird. Der Sensorwert wird also ausschließlich von diesem Teil der Schaltung bestimmt. Im Versorgungsmodus stellt der Sensoranschluß natürlich einen zweiten Pluspol in der Schaltung dar. Dies ist für die Funktion nicht entscheidend, beeinflußt aber den Stromverbrauch.

Wieviel Zeit der RCX jeweils mit Versorgung und Auslesen verbringt, wird komplett durch die Software gesteuert. Laut [Gasperi] liest die Firmware im ROM nach 3 ms im Versorgungsmodus den Sensor 0,1 ms lang aus. BrickOS hingegen benutzt ein sehr viel schnelleres Timing und fragt den Sensor so schnell ab, wie es der A/D-Wandler, der dazugehörige Interrupthandler und die übrige Software erlauben. Die Dauer eines Auslesevorgangs erreicht Zeiten in der Größenordnung von 40 Mikrosekunden, was aber stark von der laufenden Software abhängt. Die Versorgungsphase eines Sensors ist dementsprechend so lang, wie das Auslesen der anderen drei Sensoren dauert. Details zum Verfahren beschreibt die Kerneldokumentation von [Nielsson].

2.5.2 Elektrische Grenzen der Anschlüsse

Genau wie bei den Aktoren ist auch jeder Sensoranschluß am RCX durch einen Strombegrenzer geschützt. Dadurch muß jeder Sensor mit nur wenigen Milliampere Strom auskommen, was eine erhebliche Anforderung an das Design darstellt. Umso erstaunlicher ist es, daß sich in der Literatur fast keine brauchbaren Informationen über den Strombegrenzer finden. Die im Anhang C.4 beschriebene Messung war also nötig, um die fehlenden Daten zu ermitteln.

Zunächst wurde eine Meßreihe aufgenommen, bei der auf dem RCX nur die Firmware lief. Ihre Auswertung in Abbildung 2.13 zeigt, daß sich im RCX ein Strombegrenzer befindet, der bei kleinen Strömen einen Innenwiderstand von etwa 70 Ω hat und ab 12 mA den durchfließenden Strom stark abschwächt.

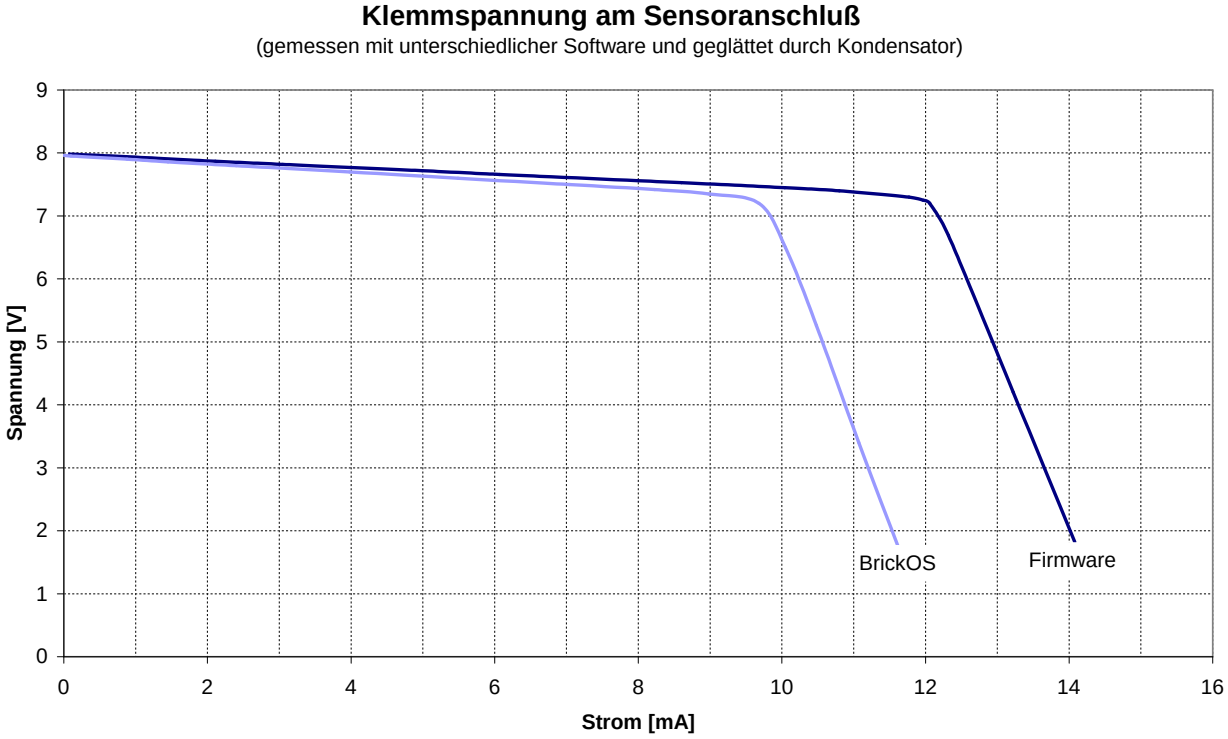


Abbildung 2.13: Klemmenspannung an einem aktiven Sensor

Eine zweite Meßreihe unter brickOS ergibt, daß der maximal entnehmbare Strom mit dieser Software geringer ist. Ursache dafür ist der schnelle Ausleserythmus, bei dem der Sensor etwa 25 % der Zeit nicht mit Strom versorgt werden kann. In der übrigen Zeit fließt ein höherer Strom, um den Kondensator wieder aufzuladen, und daher bricht die Spannung deutlich früher ein.

Wenn der Sensor eine stabilisierte Spannungsversorgung braucht, gelten hier natürlich die selben Richtlinien wie bei den Aktoren. Der Regler muß mit einer möglichst geringen Spannung auskommen und darf selber fast keinen Strom verbrauchen. Es kann sinnvoll sein, den Gleichrichter aus Schottky-Dioden aufzubauen, und der Kondensator muß ausreichend groß sein. Die Mindestkapazität liegt bei einem Mikrofarad pro Milliampere Stromverbrauch.

2.5.3 Interpretation der Sensorwerte

Beim Auslesen eines Sensors mißt die Hardware den Sensorwert als Zahl zwischen 0 und 1023. Die Software kann je nach angeschlossenem Sensortyp daraus einfacher zu verarbeitende Darstellungen

wie die Helligkeit in Prozent oder die Temperatur in Grad Celsius berechnen. [Gasperi] gibt die von Lego benutzten Formeln für die Standardsensoren an und stellt die verschiedenen Darstellungsformen in der folgenden Tabelle gegenüber.

Spannung	Rohwert	Widerstand	Licht	Temperatur	Taster
0,0 V	0	0 Ω	-	-	1
1,1 V	225	2816 Ω	-	70,0°C	1
1,6 V	322	4587 Ω	100	57,9°C	1
2,2 V	450	7840 Ω	82	41,9°C	1
2,8 V	565	12309 Ω	65	27,5°C	0
3,8 V	785	32845 Ω	34	0.0°C	0
4,6 V	945	119620 Ω	11	-20,0°C	0
5,0 V	1023	∞	0	-	0

Tabelle 2.1: Umrechnungstabelle für Sensorwerte

Die einzelnen Zeilen stellen genau diejenigen Rohwerte da, die für die Umrechnungsformeln Schlüsselwerte darstellen. Im einzelnen sind das der größte und kleinste Sensorwert, die Meßwerte bei den Temperaturen -20°C und 70°C, der größte Helligkeitswert von 100% und die beiden Punkte, an deren die Hysteresefunktion für den Tastsensor umschaltet.

2.5.4 Entwicklung eigener Sensoren

Die Schaltung in Abbildung 2.12 stellt den Grundbaustein eines jeden Mindstorms-Sensors dar. Wie sein passives Gegenstück muß auch jeder aktive Sensor den Strom verändern, der vom Sensoranschluß nach Masse fließt. Häufig muß der Sensorstromkreis vom übrigen Teil der Schaltung entkoppelt werden, damit dieser nicht den Sensorwert beeinflußt. [Gasperi] gibt für diese Ausgabe eine Schaltung mit einem Operationsverstärker an, die häufig eingesetzt wird und in Abbildung 2.14 zu sehen ist.

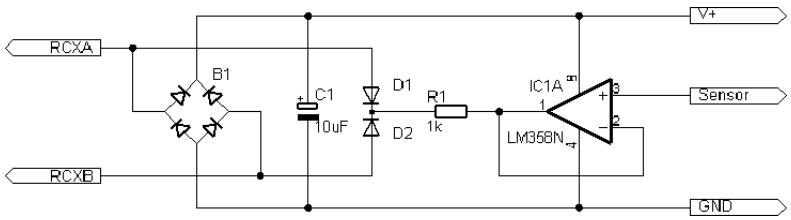


Abbildung 2.14: Einfaches Interface für analoge Sensoren

Der Auslesestrom des RCX fließt über R_1 und den Ausgang des Operationsverstärkers nach Masse ab. Dabei wird der Strom so geregelt, daß die Spannung hinter R_1 gleich der Eingangsspannung des Operationsverstärkers ist. Diese Schaltung liefert gute Ergebnisse, allerdings ist die Übertragungsfunktion durch die Kennlinie der Dioden im Stromkreis nicht linear und der Stromverbrauch des Interfaces kann auf bis zu 8 mA anwachsen, wenn die Eingangsspannung des OPs bei 0 V liegt. Dennoch eignet sie sich gut für einfache Sensoren. Im Anhang A.2 wird ein kompletter Geräuschsensor für Mindstorms vorgestellt, der auf diesem Interface basiert.

Wenn ein Sensor auf eine lineare Übertragungsfunktion angewiesen ist, bietet sich das von [Mientki] weiterentwickelte Interface an. Es braucht nur einen Transistor mehr als das von Michael Gasperi und

hat dafür den Vorteil, daß es über eine lineare Charakteristik verfügt. Dieses Interface ist daher eine vielseitig einsetzbare Basis für analoge Sensoren.

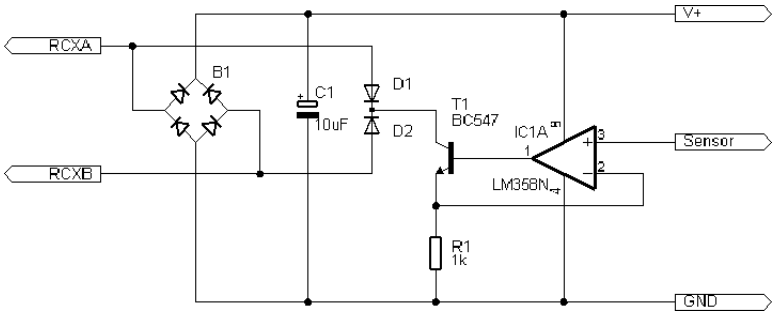


Abbildung 2.15: Lineares Interface für analoge Sensoren

Wenn die Spannung U an den offenen Eingang des Operationsverstärkers gelegt wird, öffnet dieser den Transistor soweit, daß die Spannungen an seinen beiden Eingängen gleich sind. Der Strom I im Sensorkreis läßt sich damit über R_1 berechnen.

$$I \cdot R_1 \quad = \quad U \tag{2.6}$$

Der vom RCX gemessene Wert ist proportional zu I und damit auch zu U. Sinnvolle Werte für U liegen zwischen 0 V (Sensorwert 1023) und 0,5 V (Sensorwert 0).

2.6 Infrarot-Kommunikation

An der Vorderseite des RCX ist ein Infrarot-Transceiver angebracht, über den der RCX mit der Umwelt kommunizieren kann. Für den PC gibt mit dem Mindstorms Tower ein passendes Gegenstück, das an die serielle Schnittstelle oder einen USB-Port angeschlossen werden kann. Die Transceiver in RCX und PC können eine Verbindung herstellen, über die Programme in den RCX übertragen werden können. Prinzipiell ist aber jede Art von Kommunikation zwischen mehreren RCX und Towern möglich, beispielsweise das Senden von Telemetriedaten zum PC, das Fernsteuern eines Roboters oder die Kommunikation zwischen Robotern in einer Gruppe.

Abbildung 2.16: Infrarot-Transceiver von Tower und RCX

Trotzdem hat die Infrarot-Verbindung auch einige Nachteile. Die Verbindung verfügt mit 3,8 kBit/s nur über eine geringe Bandbreite. Die Übertragungen finden in einem Broadcast-Medium statt, das nur ein Teilnehmer zur Zeit nutzen kann und in dem es beim Senden zu Kollisionen kommen kann. Und schließlich brauchen die Transceiver Sichtkontakt, wenn sie nicht in einer Umgebung mit genügend stark reflektierenden Wänden betrieben werden.

2.6.1 Funktionsweise des Transceivers

Der Transceiver besteht aus einem Sender und einem Empfänger, die jeweils mit Infrarotsignalen auf einer Trägerfrequenz von 38 kHz arbeiten. Die Datenübertragung erfolgt seriell, dazu wird das Trägersignal im Takt des seriellen Datenstroms ein- und ausgeschaltet.

Der Sender besteht aus dem Signalgenerator und zwei in Reihe geschalteten IR-Sendedioden vom Typ TSAL6200. Die Reichweite kann in Stufen variiert werden, die sich nur darin unterscheiden, welcher Strom durch die Sendedioden fließt. Im Nahbereich reicht ein Strom von 10 mA aus, zum Überbrücken größerer Strecken fließen die maximal erlaubten 100 mA durch die Dioden. Das Umschalten erfolgt am seriellen Tower über einen Schalter, beim RCX und USB-Tower per Software.

Beim Empfänger handelt es sich um ein handelsübliches Modell vom Typ TSOP1138, das alle erforderlichen Komponenten in einem vergossenen Gehäuse enthält. Dieses besteht aus einem speziellen Kunststoff, der das einfallende Licht filtert und fast nur den Infrarotanteil durchläßt. An seinem Ausgang liefert das Modul den demodulierten seriellen Datenstrom. Zum sicheren Erkennen des Signals sind 10 aufeinanderfolgende Rechteckimpulse des Infrarot-Signals erforderlich, kürzere Impulsfolgen sind nur unter bestimmten Bedingungen möglich. Dadurch erklärt sich die Bandbreitenbegrenzung auf ein Zehntel der Trägerfrequenz, also 3800 Bit/s.

Die eigentlichen Nutzdaten werden nach dem RS232-Standard mit einem Startbit, 8 Datenbits, ungerader Parität und einem Stopbit übertragen. Zum Senden einer logischen Null wird das Trägersignal für die Dauer des Bits eingeschaltet, zum Senden der logischen Eins bleibt der Sender entsprechend lange ausgeschaltet. Auf diese Weise sendet der Transceiver im Ruhezustand kein Signal, da die serielle Datenleitung dann auf Eins liegt. Abbildung 2.17 zeigt als Beispiel das Datensignal für die Übertragung des Wertes 141, eine genaue Beschreibung des RS232-Standards befindet sich in [Strangio].

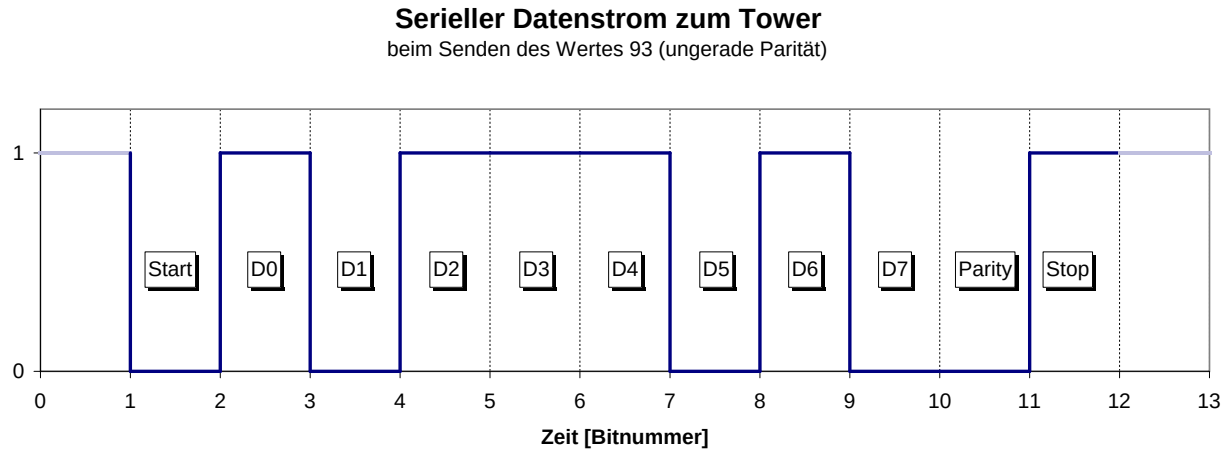


Abbildung 2.17: Beispiel für ein per Infrarot übertragenes Datensignal

Die Lego-Software überträgt Daten mit einer Geschwindigkeit von 2400 Baud, brickOS bietet daneben auch einen Modus mit 4800 Baud an, was abhängig von den Daten aber die Spezifikation des Empfängermoduls verletzen kann. Dennoch funktioniert der Modus meistens zuverlässig genug, so daß er unter brickOS standardmäßig zum Übertragen von Kernel und Anwendungsprogrammen benutzt wird. Die Benutzerprogramme arbeiten jedoch unter allen Plattformen einheitlich mit 2400 Baud.

Auf dem beschriebenen Physical Layer setzen die Lego-Software und brickOS eigene Data Link Layer auf, die sich stark in ihrem Aufbau und Einsatzgebiet unterscheiden und auch nicht zueinander kompatibel sind.

2.6.2 Data Link Layer der Lego Software

Die Lego-Software benutzt ein Protokoll, das die größtmögliche Zuverlässigkeit erreichen soll. Die Grundidee hierbei ist, daß im zeitlichen Mittel gleiche Anzahlen von Bits mit den Werten 0 und 1 übertragen werden und daß keine langen Sequenzen gleicher Bits auftreten können. Dadurch kann der Empfänger sich optimal auf das Signal einstellen und empfängt auch schwache Signale noch zuverlässig. Der Preis dafür ist eine auf weniger als die Hälfte reduzierte Bandbreite. Abbildung 2.18 zeigt den Aufbau eines Paketes, mit dem die n Datenbytes d_1 bis d_n übertragen werden.

Abbildung 2.18: Aufbau eines Lego Datenpaketes

Die ersten 3 Bytes stellen den Header dar. Ihre Signalmuster sollen dem Empfänger Gelegenheit geben, sich auf das Infrarotsignal einzustellen. Die eigentlichen Daten des Headers werden von der Software nicht ausgewertet und müssen auch nicht dem gezeigten Schema entsprechen. Darauf folgen die Nutzdaten, wobei jedes Byte einmal direkt und einmal invertiert übertragen wird. Das Paket endet mit einer Checksumme, die wieder direkt und invertiert gesendet wird.

$$\overline{x} = x \text{ xor 0xFF} \tag{2.7}$$

$$c = \left(\sum_{i=1}^{n} d_i\right) \text{ and 0xFF} \tag{2.8}$$

Der PC sendet Datenpakete an den RCX, die einzelne Befehle eines Bytecode-Programms enthalten. Der Interpreter führt sie aus und sendet gegebenenfalls eine Antwort an den PC zurück. Dadurch lassen sich viele Funktionen, die einer Applikation auf dem RCX zur Verfügung stehen, auch vom PC aus aktivieren. Daneben gibt es Befehle, die nur von außen funktionieren (z.B. Download eines Programms) oder nur von Applikationen genutzt werden können (z.B. Steuerung des Kontrollflusses). Bei Übertragungsproblemen wiederholt der PC den Befehl so lange, bis er eine gültige Antwort erhält. Der RCX führt den ersten gültigen Befehl aus und schickt eine passende Antwort zurück an den PC. Empfängt er unmittelbar danach den selben Befehl nochmals, geht er davon aus, daß seine Antwort den PC nicht erreicht hat und wiederholt diese, ohne den Befehl nochmals auszuführen. Für den Fall, daß der selbe Befehl mehrmals nacheinander ausgeführt werden soll, gibt es von jedem Befehl und jeder Antwort zwei Versionen, zwischen denen PC und RCX ständig alternieren müssen.

Das von Lego entwickelte Protokoll ist gut auf die physikalischen Eigenschaften der Transceiver abgestimmt und bietet durch die vielen Ebenen der Fehlererkennung (Paritätsbit, Inverse und Checksumme) eine hohe Zuverlässigkeit. Es ist relativ langsam und dient im Wesentlichen zum Fernsteuern eines RCX. Daneben ist es nur möglich, ein einzelnes Byte als „message“ an alle Stationen in Reichweite zu senden. Ein unabhängiger Beobachter kann den Inhalt eines Paketes nicht eindeutig interpretieren, wenn er nicht weiß, ob es vom PC oder RCX gesendet wurde. Damit ist das Protokoll für aufwendigere Kommunikation nicht geeignet, dennoch ist es unter Sprachen wie NQC das einzig verfügbare Protokoll. Eine detaillierte Beschreibung aller Befehle und Antworten ist im Lego Mindstorms SDK enthalten (zu finden unter [Legores]).

2.6.3 Data Link Layer von brickOS

BrickOS setzt auf den Physical Layer zwei einfacher gestrickte Protokolle auf, die hauptsächlich der Kommunikation zwischen Applikationen dienen. Dabei spielt es keine Rolle, ob die Applikationen auf einem RCX oder einem PC laufen.

Bei dem ersten Protokoll handelt es sich um den Integrity Layer. Er erlaubt es, Datenpakete mit einer Nutzlast von bis zu 255 Bytes an alle Empfänger in Reichweite zu senden. Die Pakete werden dabei mit einer Checksumme abgesichert. Abbildung 2.19 zeigt den Aufbau eines Paketes.

Abbildung 2.19: Aufbau eines Datenpaketes im brickOS Integrity Layer

Der Header besteht lediglich aus einer Konstanten und der Länge des Paketes, direkt daran schließen sich die eigentlichen Daten und die Checksumme an.

$$\text{len} = n \tag{2.9}$$

$$c = \left(\text{0xF0} + \text{len} + \sum_{i=1}^{n} d_i + \text{0xFF} \right) \text{ and 0xFF} \tag{2.10}$$

Über den Integrity Layer lassen sich Nachrichten nur als Broadcast versenden. Viele Anwendungen brauchen hingegen eine Möglichkeit, einzelne Stationen adressieren zu können. Diese Funktionalität bietet der Addressing Layer, bei dem Sender und Empfänger über Adressen mit einer Breite von 8 Bit angesprochen werden können. Diese sind nach dem Vorbild von UNIX in eine Hostadresse und eine Portnummer aufteilt. Standardmäßig sind die oberen 4 Bits für den Host reserviert, wobei der RCX die Nummer 0 und der PC die Nummer 8 bekommen. Die unteren 4 Bits erlauben es, einzelne Ports auf dem Host anzusprechen, von denen nur Port 0 für den Programmdownload reserviert ist. Diese Parameter können natürlich auch in der Konfiguration geändert werden.

Abbildung 2.20: Aufbau eines Datenpaketes im brickOS Addressing Layer

Abbildung 2.20 zeigt den Aufbau eines Paketes im Addressing Layer. Gegenüber dem Integrity Layer sind im Wesentlichen Felder zur Speicherung der Adressen von Sender und Empfänger hinzugekommen, dafür ist die maximale Paketgröße auf 253 Bytes reduziert.

$$\text{len} = n + 2 \tag{2.11}$$

$$c = \left(\text{0xF1} + \text{len} + \text{dest} + \text{src} + \sum_{i=1}^{n} d_i + \text{0xFF} \right) \text{ and 0xFF} \tag{2.12}$$

Die für brickOS geschriebenen Anwendungsprogramme können für den Integrity Layer und für jeden Port des Addressing Layers eine eigene Callback-Funktion setzen, die beim Empfang eines gültigen Paketes aufgerufen werden und einen Zeiger auf die Daten erhalten. Auf der PC-Seite ist die Unterstützung für das Infrarot-Protokoll natürlich abhängig vom Betriebssystem. Der für diese Aufgabe entwickelte Lego Network Protocol Daemon (lnpd) funktioniert unter aktuellen Versionen von brickOS nicht mehr. Für diese Arbeit mußte daher eine Alternative entwickelt werden, die in Abschnitt B.1 vorgestellt wird.

2.6.4 Entwicklung eigener Transceiver

Wenn man Eigenentwicklungen mit einem Mindstorms-kompatiblen Transceiver ausstattet, eröffnet das eine Vielzahl von Anwendungsgebieten. Es ist dann möglich, Geräte zu bauen, die einem Roboter

Befehle oder Informationen schicken können oder die vom Roboter aus gesteuert werden können. Dazu ist nur eine Handvoll handelsüblicher Bauteile erforderlich. In Abbildung 2.21 ist der Aufbau einer Grundschaltung zu sehen. Die seriellen Daten werden über TX eingespeist bzw. an RX gelesen. Dabei werden im Gegensatz zu RS232 TTL-Pegel benutzt, eine logische 1 entspricht also einer Spannung von 5 V, während die logische 0 durch 0 V dargestellt wird.

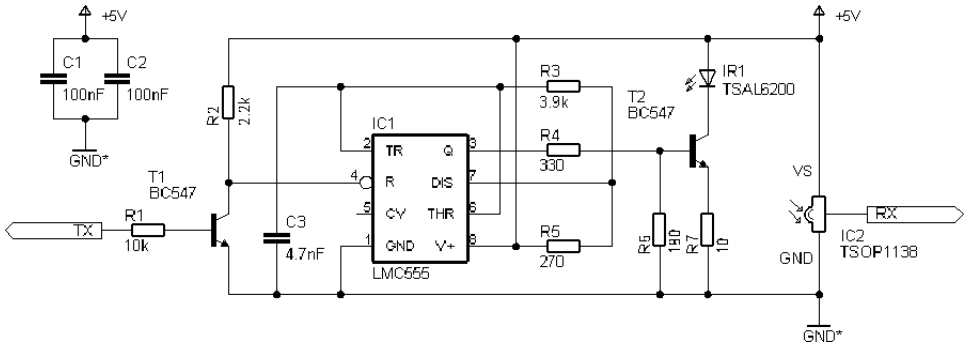


Abbildung 2.21: Grundschaltung für Infrarot-Transceiver

Der LMC555 bildet zusammen mit C_3, R_3 und R_5 einen 38 kHz-Oszillator, der über den Reset-Eingang gesteuert wird. Eine logische 1 an TX entspricht dem Ruhezustand und soll kein Infrarot-Signal erzeugen. Sie muß daher von dem aus T_1, R_1 und R_2 bestehenden Inverter in eine logische 0 umgewandelt werden, um den LMC555 abzuschalten. Das Ausgangssignal des ICs steuert den Treiber für die Infrarot-Sendediode IR_1, bestehend aus T_2, R_4, R_6 und R_7. Der Treiber stellt sicher, daß beim Senden ein Strom von 100 mA durch die Sendediode fließt. Der Empfänger IC_2 braucht keine weitere Beschaltung, er liefert direkt das demodulierte Ausgangssignal und gibt im Ruhezustand wie gefordert eine logische 1 aus. Die Kondensatoren C_1 und C_2 sind Pufferkondensatoren für IC_1 und IC_2 und müssen demnach möglichst dicht an diesen Bauteilen plaziert werden.

Die Schaltung kann natürlich auch abhängig von den Erfordernissen angepaßt werden.

- Wenn der Transceiver an einem Mikrocontroller betrieben werden soll, kann dieser auch die Trägerfrequenz erzeugen. R_4 und RX werden in so einem Fall direkt an den Mikrocontroller angeschlossen, der LMC555 mit Beschaltung und der Inverter an TX fallen dann komplett weg.
- Die Reichweite des Senders kann erhöht werden, indem mehrere Sendedioden in Reihe geschaltet werden. Diese müssen dann aber von einer höheren Spannung als 5 V gespeist werden.
- Das Empfangsmodul TSOP1138 kann durch den Vergleichstyp TSOP1738 ersetzt werden. Durch die strengeren Toleranzgrenzen des Bausteins sind Geschwindigkeiten von mehr als 3800 Baud allerdings völlig ausgeschlossen.
- Mit Hilfe eines Pegelwandlers wie dem MAX232 von Maxim können die Pegel an RX und TX so angepaßt werden, daß der Transceiver direkt an RS232-Schnittstellen betrieben werden kann.

Auf der Basis dieser Schaltung ist es möglich, einen Tower zu bauen, der voll kompatibel zu dem seriellen Mindstorms-Tower ist. Anhang A.3 zeigt die dafür nötigen Änderungen der Schaltung.

2.6.5 Der Lego Mindstorms Tower

Der ältere serielle Tower und das neuere USB-Modell von Lego funktionieren natürlich beide nach dem im vorigen Abschnitt dargestellten Prinzip, sie unterscheiden sich aber deutlich in ihren Eigenschaften und ihrer Programmierung.

Der serielle Tower soll möglichst lange mit einer 9 V-Batterie funktionieren und setzt deswegen rigorose Maßnahmen zum Stromsparen ein. Er ist standardmäßig ausgeschaltet und aktiviert sich erst, wenn tatsächlich Daten gesendet werden. Danach bleibt er für etwa 4-5 Sekunden eingeschaltet. Dadurch ist gewährleistet, daß der Tower nie unbeabsichtigt Strom verbraucht, andererseits muß etwa alle 4 Sekunden mindestens ein Byte gesendet werden, damit der Tower ständig empfangsbereit bleibt. Diese Funktion findet sich unter dem Namen Keepalive-Mechanismus in den Steuerprogrammen für den PC. Außerdem empfängt der Tower wie jeder Infrarot-Transceiver die Reflexion seines eigenen Sendesignals und damit alle von ihm selbst gesendeten Daten. Diese Tatsache ließe sich zum Verifizieren der gesendeten Daten und damit zum Erkennen von Kollisionen ausnutzen, was aber an technischen Eigenschaften der seriellen Schnittstelle im PC scheitern kann (mehr dazu in Anhang B.1).

Im seriellen Datenkabel zwischen Tower und PC sind nur die Leitungen GND, RXD, TXD, RTS und CTS belegt. RTS ist direkt mit CTS verbunden und dient dazu, den angeschlossenen Tower erkennen zu können. Die restlichen Leitungen werden für die serielle Datenübertragung in beide Richtungen gebraucht. Der von John Barnes rekonstruierte Schaltplan findet sich unter [Mientki].

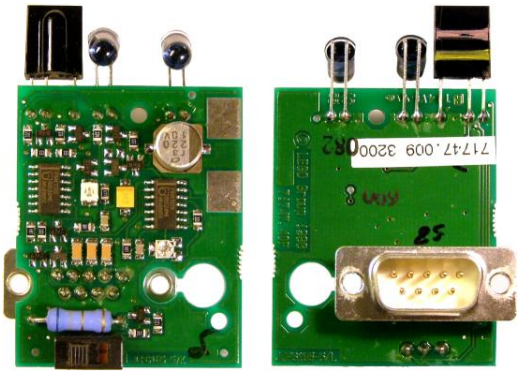


Abbildung 2.22: Platine im Inneren des seriellen Towers

Bei dem USB-Tower sind einige technische Verbesserungen realisiert worden. Der Tower bezieht seinen Strom aus dem Rechner und kann daher auf Trickschaltungen zum Stromsparen verzichten. Dadurch ist er ständig empfangsbereit und ein Keepalive-Mechanismus ist nicht mehr nötig. Die Vermeidung von Kollisionen und die Verifikation der gesendeten Daten anhand des Infrarot-Echos sind auf den Tower verlegt worden und funktionieren so besser als bei dem seriellen Gegenstück. Es werden auch keine Echo-Daten mehr zum PC weitergeleitet. Beim Design des Towers wurde eine Reihe von Funktionen vorgesehen, die allerdings noch nicht implementiert sind oder nicht zusammen mit den aktuellen RCX-Modellen funktionieren. Dazu gehören höhere Geschwindigkeiten bis 19200 Baud, ein konfigurierbares Trägersignal, andere Modulationsverfahren und sogar die Möglichkeit, die Daten per Funk statt per Infrarot zu übertragen.

Eine Bibliothek zur Steuerung des Towers aus eigenen Programmen heraus befindet sich im Anhang B.1. Sie ist als Ersatz für den Lego Network Protocol Daemon (lnpd) gedacht, der in vielen Szenarien nicht zuverlässig funktioniert und außerdem weder neuere BrickOS-Versionen noch den USB-Tower unterstützt.

2.7 Abmessungen der Legosteine

Ein wichtiger Faktor bei der Entwicklung von Mindstorms-Peripherie ist die Wahl des Gehäuses. Idealerweise sollte das Gehäuse die Abmessungen eines Legosteins haben, damit das Bauteil problemlos an einen Roboter angebaut werden kann und nahtlos in das Gesamtbild paßt. Im günstigsten Fall besteht

die Möglichkeit, einen vorhandenen Legostein auszuhöhlen und die Schaltung darin einzubauen, wie es im Anhang A.2 für den Schallsensor vorgestellt wird.

Die genauen Abmessungen von Legosteinen haben in jedem Fall einen Einfluß auf den Aufbau der Schaltung. Alle relevanten Größen sind gut dokumentiert, sie finden sich zum Beispiel in [Pfeifer]. Die Abbildung 2.23 stellt die wichtigsten Maße am Beispiel eines Legosteins der Größe 2x4 dar.

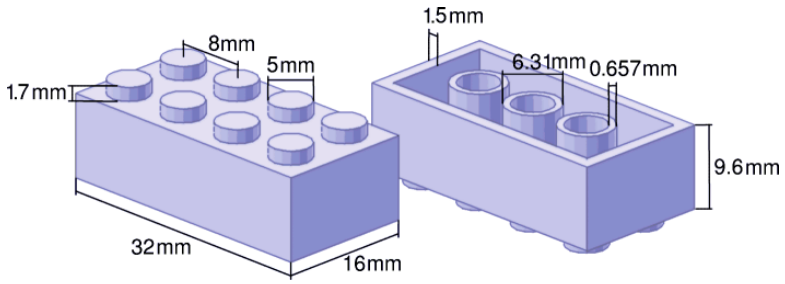


Abbildung 2.23: Abmessungen eines Legosteins

Einige Quellen wie [Knudsen] geben diese Längen in Zoll umgerechnet an. Dabei handelt es sich aber um gerundete Werte, die leicht von den wirklichen Abmessungen abweichen.

Abmessung	**Größe**
Kantenlänge eines Bausteins	8 mm pro Knopf
Höhe eines Bausteins	9,6 mm
Höhe einer Platte	3,2 mm
Baustein Wanddicke	1,5 mm
Knopf Durchmesser	5 mm
Knopf Höhe	1,7 mm
Knopf Abstand	8 mm
Zylinder Durchmesser	6,31 mm
Zylinder Wanddicke	0,657 mm

Tabelle 2.2: Abmessungen eines Legosteins

Die Legosteine selbst bestehen aus dem Kunstharz Acrylnitril-Butadien-Styrol, das sich durch hohe Festigkeit, Schlagfestigkeit und Oberflächenhärte auszeichnet. Die Steine lassen sich sehr gut mit Feinwerkzeug bearbeiten, solange das Material nicht zu warm wird. Bohren, Schleifen und Fräsen sind problemlos möglich. Damit eignen sie sich gut als Gehäuse für Eigenentwicklungen, solange der Platz in dem Stein ausreicht, um die Schaltung aufzunehmen.

2.8 Einfluß der Batteriespannung

Zum Abschluß des Kapitels lohnt sich noch ein Blick auf die Batterien, aus denen der RCX und damit auch alle Sensoren und Aktoren ihren Strom beziehen. Ihre Spannung hängt einerseits stark vom Ladestand ab, andererseits sind RCX und Peripherie darauf angewiesen, daß die Spannung bestimmte Werte nicht unterschreitet. In der Praxis hält ein Satz handelsüblicher Alkaline-Batterien einige Stunden Dauerbetrieb aus, bevor die Zellen leer sind. Die genaue Zeit ist natürlich von der Belastung (Auslastung der CPU, Leistung der Motoren, Stromaufnahme der aktiven Sensoren, Aktivität und Signalstärke des IR-Transceivers) abhängig. Wenn die Klemmenspannung des Batteriepaketes unter

Belastung unter 6,6 V sinkt, leuchtet ein Symbol im Display auf, daß die Batterien bald ersetzt werden müssen. Beim Erreichen von 4,3 V kommt es zu einer Notabschaltung des RCX, wobei er auch den gesamten RAM-Inhalt verliert.

Die schwächer werdenden Batterien machen sich aber schon früher bemerkbar. Die Klemmenspannungen von Sensoren und Aktoren folgen der Batteriespannung im Abstand von mindestens 0,5 V, bei wachsender Stromaufnahme kann der Abstand auch deutlich größer werden. Bei fast vollständig entladenen Batterien kann es dazu kommen, daß die Ausgänge des RCX auf unter 4 V absinken, auch wenn die Stromaufnahme eines Sensors oder Aktors sehr gering ist. Peripheriebausteine müssen mit dieser Situation zurechtkommen, sich also entweder anpassen oder rechtzeitig die Funktion einstellen, bevor sie Fehler verursachen können. In Abbildung 2.24 zu sehen, wie die Klemmenspannung einer typischen Alkaline-Zelle (Einwegbatterie) während der Entladung absinkt.

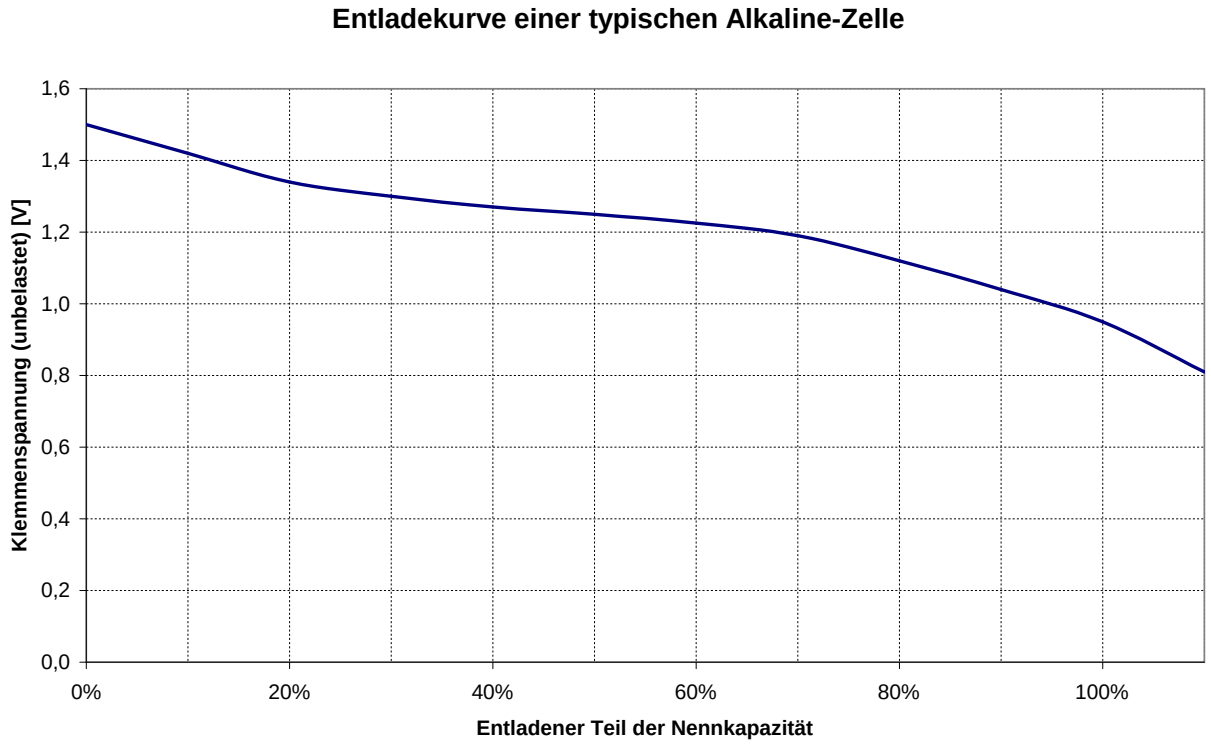


Abbildung 2.24: Klemmenspannung einer typischen Alkaline-Batterie nach [HorHill]

Läuft eine Schaltung gar nicht unter den Einschränkungen des RCX, kann sie notfalls auch von einer externen Spannungsquelle versorgt werden. Besser ist es natürlich, den Stromverbrauch so weit zu reduzieren, daß dies nicht nötig ist. [HorHill] und Application Note 606 von [Microchip] geben einige Tips für mögliche Einsparungen.

Kapitel 3

Einsatz von Mikrocontrollern

Das Kapitel 2 hat gezeigt, wie der Mindstorms Baukasten um zusätzliche Sensoren und Aktoren erweitert werden kann. Einfache Funktionen lassen sich häufig mit einfachen Schaltungen und wenigen Teilen realisieren. Sollen die Sensoren und Aktoren jedoch komplexere Aufgaben übernehmen, kann die Entwicklung relativ schnell an Grenzen stoßen.

Mit der Schwierigkeit der Aufgabe wächst in der Regel auch die Anzahl der Bauteile, die für die Realisierung gebraucht werden. Die Schaltung braucht mehr Platz und mehr Strom, und beides wird in einem Mindstorms-Roboter schnell zu einem Problem. Außerdem kann bei Sensoren die Datenübertragung zum RCX einen Engpaß darstellen. Der Sensor wird höchstens einige tausend Mal pro Sekunde abgefragt und kann jedesmal nur einen Analogwert übermitteln. Bei der Beobachtung schneller oder komplexer Vorgänge ist es daher nötig, die Meßdaten im Sensor erst aufzubereiten, bevor sie an den RCX geschickt werden. Das erhöht wiederum die Komplexität des Sensors. Ein möglicher Ausweg aus dieser Situation ist, einen Mikrocontroller als Steuerelement in die Schaltung einzubauen.

Mikrocontroller sind integrierte Bausteine, die einen kompletten Computer auf einem Chip vereinigen und so den Aufbau einer aufwendigen Steuerung mit sehr wenig Hardware erlauben. Große Teile der Steuerung werden dann von der auf dem Mikrocontroller laufenden Software übernommen und lassen sich so schnell und einfach austauschen, ohne daß die zugrunde liegende Schaltung geändert werden muß. Mikrocontroller sind in vielen Geräten des Alltags zu finden und werden in großen Stückzahlen zu niedrigsten Preisen produziert.

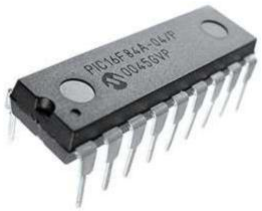


Abbildung 3.1: Ein Mikrocontroller aus der PIC-Familie

Die niedrigen Kosten ermöglichen es, Mikrocontroller speziell für eine Aufgabe zu entwickeln, so daß es eine Vielzahl verschiedenster Typen und Ausstattungen auf dem Markt gibt. Den meisten Controllern ist gemeinsam, daß sie auf geringen Stromverbrauch, geringe Größe und einen niedrigen Preis, nicht aber auf Leistungsfähigkeit optimiert sind. Die Ressourcen sind meistens so begrenzt, daß der Entwickler sie effektiv nutzen muß, um nicht auf ein größeres Modell mit höherem Preis und höherem Stromverbrauch umsteigen zu müssen.

In jedem Mikrocontroller ist eine Reihe von Peripheriegeräten integriert, die für die Programmausführung gebraucht werden oder die den Controller mit anderen Teilen der Schaltung verbinden. Häufig zu finden sind die folgenden Baugruppen:

- Integrierter Speicher für Programmcode (Flash oder EPROM) sowie Daten (RAM)
- Ansteuerung für einen externen Quarz, möglicherweise ein interner Taktgenerator
- Digitale und analoge Ein- und Ausgänge
- Analog-Digital- und Digital-Analog-Wandler
- Spannungskomparatoren mit programmierbarer Referenzspannungsquelle
- Schnittstellen wie UART, USB, I^2C, CAN, SPI
- Erzeugung einer Pulsweitenmodulation
- Timer zur Zeitmessung und Ablaufsteuerung
- Watchdog Timer zum Erkennen von Programmabstürzen
- Anschlußmöglichkeiten für einen Debugger, JTAG-Interface
- Interfaces für bestimmte Peripheriebausteine wie Displays

Die Vorteile beim Einsatz von Mikrocontrollern liegen auf der Hand. Zum einen vereinfacht sich das Design der Hardware ganz erheblich, weil viel weniger diskrete Komponenten benutzt werden müssen. Zum anderen wird die Funktionalität zu einem großen Teil von der Software des Mikrocontrollers gebildet, wodurch Änderungen und Fehlerkorrekturen schnell und problemlos möglich sind. Dadurch wird es möglich, leistungsfähige Sensoren und Aktoren für Mindstorms zu entwickeln.

3.1 Auswahl eines Mikrocontrollers

Wenn in einem Projekt ein Mikrocontroller eingesetzt werden soll, muß zunächst ein passender Typ ausgewählt werden. Die Entscheidung ist nicht einfach, weil eine ganze Reihe von Faktoren gegeneinander abgewogen werden müssen.

- Wie einfach ist der Controller zu beschaffen, was kostet er und wie viele zusätzliche Bauteile sind für den Betrieb erforderlich?
- In welchen Grenzen liegen Betriebsspannung und Stromverbrauch?
- Was für Anschlüsse, Peripherie-Interfaces und interne Funktionsgruppen müssen vorhanden sein, wieviel Speicher wird für Programmcode und Daten gebraucht?
- Ist bei Bedarf der Umstieg auf einen größeren Mikrocontroller aus der selben Familie möglich? Mit wieviel Portierungsaufwand ist dies verbunden?
- Wie viel Informationsmaterial existiert für den Controller? Gibt es fertige Codebausteine für Standardaufgaben, die sich in das eigene Projekt integrieren lassen?
- Was für Entwicklungswerkzeuge sind verfügbar, wie gut sind sie und was kosten sie? Interessant sind insbesondere Assembler, Debugger, Simulatoren, Hochsprachencompiler und gegebenfalls Backends für Modellierungswerkzeuge.
- Was soll das spätere Steuerprogramm leisten, soll es in Assembler oder einer Hochsprache geschrieben werden? Wie einfach läßt sich das Programm in der Architektur des Mikrocontrollers realisieren (Speicher, Instruktionen, Registersatz)?

- Falls die Ausführungsgeschwindigkeit eine Rolle spielt: Wie viele Instruktionen können pro Sekunde ausgeführt werden? Dies ist nicht mit der Taktfrequenz zu verwechseln, denn oft braucht eine Instruktion mehrere Takte.
- Welchen Aufwand bedeutet es, ein neues Programm in den Mikrocontroller zu laden, wie oft und wie schnell ist das problemlos möglich?
- Kann der Controller in seiner Schaltung programmiert werden (In-System oder In-Circuit Programming), oder muß er dafür ausgebaut werden?

Legt man diese Kriterien zugrunde, eignen sich für den Einsatz mit Lego Mindstorms zwei Familien von Mikrocontrollern besonders gut: AVR von [Atmel] und die Mid-Range PICs von [Microchip]. Beide gibt es in zahlreichen Ausstattungsvarianten günstig zu kaufen. Sie wurden bereits in vielen Projekten eingesetzt, so daß eine breite Unterstützung in Form von Entwicklungswerkzeugen und Dokumentation kostenlos zu bekommen ist. Der Programmcode wird in Flash-Speicher auf dem Controller gespeichert und kann so einfach geändert werden. Dies ist bei beiden Familien auch möglich, wenn der Controller bereits in einer Schaltung eingebaut ist.

Die Mikrocontroller aus der AVR-Reihe verfügen über eine Architektur, die speziell als Zielplatform für Hochsprachencompiler wie C ausgelegt ist. Es gibt 32 Register und einen Befehlssatz von etwa 130 Instruktionen, der auch komplexere Operationen wie die Multiplikation abdeckt. Die Ausführung eines Befehls dauert 1-2 Taktzyklen, und der Controller verfügt über relativ viel Speicher (1-128kB Programmcode, 128-4096 Bytes EEPROM und 0-4kB RAM).

Abbildung 3.2: AVR Mikrocontroller von Atmel

Die ersten PICs wurden 1976 als I/O-Controller für größere Prozessoren entwickelt und waren nicht dafür gedacht, aufwendige Berechnungen durchzuführen oder in einer Hochsprache programmiert zu werden. Trotz aller Weiterentwicklungen ist diese grundsätzliche Eigenschaft bei den meisten PICs erhalten geblieben. Es gibt bei ihnen nur ein Register, und die meisten der 35 Instruktionen beschäftigen sich mit Bitmanipulation oder einfacher Arithmetik, ihre Ausführung dauert 4-8 Taktzyklen.

Viele der PICs enthalten relativ wenig Speicher (0.5-8k Programmcode, 0-256 Bytes EEPROM, 25-368 Bytes RAM), und ihre Programmierung ist durch die fehlenden Register und die wenigen Instruktionen komplizierter als bei den AVRs. Erschwerend kommen noch die Eigenheiten der Architektur hinzu, beispielsweise bietet der Stack nur Platz für 8 Rücksprungadressen und die Speicherbereiche für Programmcode und Daten sind in Bänken organisiert, zwischen denen häufig umgeschaltet werden muß. Die Stärke der PICs ist eindeutig die Steuerung von einfachen I/O-Vorgängen, wofür sie über entsprechend leistungsfähige Peripheriebausteine verfügen.

Zusammenfassend läßt sich sagen, daß die Mikrocontroller von Atmel denen von Microchip in Bezug auf die Programmierbarkeit überlegen sind. Für die Entwicklung des Ortungssystems ist dies aber nicht ausschlaggebend, weil der Sensor nur verhältnismäßig einfache Aufgaben erledigen muß. Hier ist der Nutzen der leistungsfähigen und weitgehend konfigurierbaren integrierten Peripherie der PICs größer. Letztendlich fällt die Entscheidung zwischen AVR und PIC aber an einer ganz anderen Stelle: Der Stromverbrauch der PICs ist geringer. In einem Legosensor, dem insgesamt maximal 10 mA zur Verfügung stehen, ist dies ein so wichtiger Faktor, daß er den Ausschlag zugunsten des PICs gibt.

3.2 Die PIC-Familie von Microchip

Für die Umsetzung dieser Arbeit wurden zwei Modelle aus der Reihe der Mid-Range PICs ausgewählt, und zwar der 12F675 als leistungsfähiger und sparsamer Prozessor mit nur 8 Pins sowie der 16F628 für Aufgaben, bei denen mehr Pins benötigt werden. Die folgenden beiden Abschnitte beschreiben kurz die wichtigsten Eigenschaften der ausgewählten PICs, um einen Überblick über die Möglichkeiten des Mikrocontrollers zu geben. Genauere Dokumentation gibt es in den Datenblättern und Application Notes von [Microchip] sowie auf den unter [Piclinks] genannten Internetseiten.

3.2.1 Beschreibung des PIC12F675

Der 12F675 ist ein vollwertiger Mid-Range PIC mit nur 8 Pins, von denen 6 zum Anschluß von Peripheriegeräten dienen und weitgehend konfigurierbar sind. Die Versorgungsspannung muß zwischen 2,0 V und 5,5 V liegen, der Stromverbrauch bei 5 V und 4 MHz beträgt maximal 0,6 mA.

Abbildung 3.3: PIC12F765

Eine ganze Reihe von Baugruppen in dem Mikrocontroller werden für die Programmausführung oder die Verbindungen zur umgebenden Schaltung gebraucht.

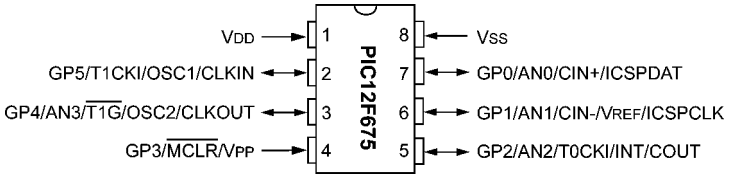


Abbildung 3.4: Pinbelegung des PIC12F765

- Der **Programmspeicher** faßt 1024 Instruktionen, dazu kommen 128 Bytes **RAM**.
- Der PIC unterstützt eine Reihe von **Taktgeneratoren** mit Frequenzen von 32 kHz bis 20 MHz. In erster Linie sind dies Quarze (OSC1, OSC2), es werden aber auch RC-Oszillatoren und externe Taktquellen (CLKIN) unterstützt. Für Anwendungen, in denen die Pins knapp sind oder nur wenige externe Bauteile benutzt werden sollen, enthält der Controller einen internen 4 MHz-Oszillator, der per Software justiert werden kann. Je vier Impulse des Taktsignals sind nötig, um eine Instruktion auszuführen. Ein Sprungbefehl benötigt zwei dieser Instruktionstakte, alle anderen Befehle nur einen.
- Bis zu sechs **I/O-Pins** sind vom Benutzer für die verschiedenen Aufgaben konfigurierbar (GP0 bis GP5), zum Beispiel als digitale Eingänge oder Ausgänge (GP3 nur als Eingang). Für den Betrieb als Eingang können einzeln interne Pullup-Widerstände dazugeschaltet werden, als Ausgang kann jeder Pin Ströme bis 25 mA abgeben oder aufnehmen.
- Der **10-Bit A/D-Wandler** kann Spannungen auf jedem der 4 Eingänge AN0..AN3 messen. Als Referenzspannung kann die Betriebsspannung oder eine externe Quelle (V_{REF}) dienen.

- Ein **Komparator** vergleicht Spannungen miteinander und entscheidet, welche größer ist. Alle 3 Anschlüsse des Komparators können mit Pins verbunden werden (CIN+, CIN-, COUT), ein Eingang kann auch mit der internen **Referenzspannungsquelle** verbunden werden, die in 16 Stufen jeweils in einem groben oder einem feinen Bereich einstellbar ist.
- Ein integrierter **8-Bit Timer** wird von dem Instruktionstakt oder einem externen Signal (am Eingang T0CKI) getrieben, so daß er alle 1, 2, 4, 8, 16, 32, 64 oder 128 Impulse inkrementiert wird. Externe Taktquellen werden dafür mit dem Prozessortakt synchronisiert.
- Der **16-Bit Timer/Counter** kann genauso vom Instruktionstakt oder von einer externen Quelle (T1CKI/T1G) angesteuert werden., wobei die Frequenz vorher durch 1, 2, 4 oder 8 geteilt wird. Dies funktioniert auch asynchron zum Prozessortakt und im Sleep-Modus.
- Für Daten, die auch nach dem Abschalten der Stromversorgung erhalten bleiben sollen, stehen 128 Bytes **EEPROM** zur Verfügung. Der Speicher liegt nicht im Adreßraum des Prozessors, kann aber programmgesteuert gelesen und geschrieben werden. Eine Vorinitialisierung beim Programmieren ist ebenfalls möglich.
- Ein kleiner Speicherbereich bietet Platz für eine **Seriennummer** von 4x7 Bit.
- Im PIC können eine Reihe von Funktionen einen gemeinsamen **Interrupt** auslösen, dessen Handler die Quelle ermitteln und entsprechend darauf reagieren kann. Auslösende Ereignisse können zum Beispiel ein Zustandswechsel an einem der Pins GP0 bis GP5, der Interrupt-Eingang (INT), eine Zustandsänderung am Komparator, der Überlauf eines Timers sowie das Ende eines EEPROM-Zugriffs oder einer A/D-Wandlung sein.
- Beim Erkennen einer Fehlfunktion kann der **Watchdog-Timer** helfen. Er wird von einem eigenen Oszillator angetrieben und muß von dem Programm in regelmäßigen Abständen zurückgesetzt werden. Passiert dies nicht, wird ein Reset ausgelöst, der das Programm neu startet.
- Codespeicher und EEPROM können vor Lesezugriffen von außen geschützt werden, um Reverse Engineering zu erschweren. Diese Einstellung wird bei der Programmierung festgelegt und kann später nicht mehr entfernt werden. Der **Leseschutz** läßt sich nur deaktivieren, indem der PIC komplett gelöscht wird, wobei mit dem Schutz auch der Speicherinhalt verloren geht.
- Die drei Leitungen ICSPCLK, ICSPDAT, und VPP bilden das **serielle Programmierinterface**. Bei entsprechender Beschaltung ist mit ihnen auch ein Programmieren des PICs in der Anwendungsschaltung möglich.

Der 12F675 eignet sich sehr gut für Anwendungen, in denen nur wenige Pins benötigt werden und der Stromverbrauch oder die Gehäusegröße eine wichtige Rolle spielt. Dennoch ist er mit einigen Peripherieeinheiten ausgestattet und eröffnet so viele Einsatzmöglichkeiten.

3.2.2 Beschreibung des PIC16F628

Der 16F628 gehört genau wie der 12F675 zu den Mid-Range PICs und ist diesem in vielen Belangen sehr ähnlich. Er verfügt allerdings über 18 Pins und mehr Speicher, dafür ist auch sein Stromverbrauch etwas höher und kann bei 5 V und 4 MHz 2 mA erreichen.

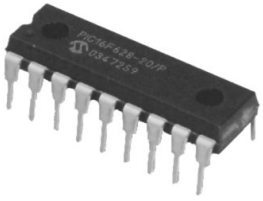


Abbildung 3.5: PIC16F628

Die folgende Liste zählt wesentliche Merkmale des 16F628 auf, die ihm von dem 12F675 unterscheiden.

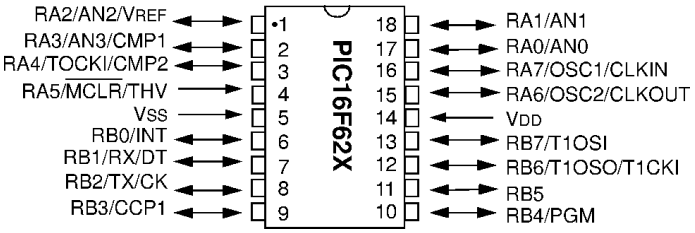


Abbildung 3.6: Pinbelegung des PIC16F628

- Der Flash-**Programmspeicher** bietet Platz für 2048 Befehle, dazu kommen 224 Bytes **RAM**.
- Der 16F628 unterstützt die selben Taktgeneratoren wie der 12F675 (OSC1, OSC2, CLKIN, CLKOUT), allerdings kann der interne Oszillator nicht justiert werden und weicht so bis zu 10 Prozent von seiner Zielfrequenz von 4 MHz ab. Unter bestimmten Bedingungen kann der PIC im Betrieb die Taktfrequenz auf 37 kHz absenken, um Energie zu sparen.
- Die bis zu 16 **I/O-Pins** teilen sich auf die Ports A und B. Alle Pins von Port A (RA0..7) sind mit Schmitt-Triggern ausgestattet und sind so für den Anschluß analoger Signale geeignet. Port B (RB0..7) ist für den Anschluß digitaler Bauteile spezialisiert, wobei alle Pins mit einem schaltbaren internen Pullup-Widerstand versehen sind. Als Ausgänge können die meisten Pins Ströme von 25 mA liefern oder aufnehmen, RA4 ist ein Open Drain-Ausgang und RA5 nur als Eingang benutzbar.
- Im Gegensatz zum 12F675 verfügt dieser PIC über keinen **Analog-Digital-Wandler**.
- Zwei **Komparatoren** können in einer von 8 Konfigurationen benutzt werden, um externe Spannungen untereinander oder mit der programmierbaren **Referenzspannungsquelle** zu vergleichen (AN0..3, VREF, CMP1..2).
- Bei einem zusätzlichen **8-Bit Timer/Zähler** sind Periode, Prescaler und Postscaler einstellbar.
- Das **Capture Compare PWM (CCP)-Modul** bietet eine Reihe von Funktionen an. Im Capture Mode kann der Wert des 16-Bit Timers im Moment einer Veränderung am Pin CCP1 für eine spätere Auswertung kopiert werden. Im Compare Mode wird CCP1 geschaltet, sobald der 16-Bit Timer einen vordefinierten Wert erreicht. Der PWM-Modus ermöglicht es, ein pulsweitenmoduliertes Rechtecksignal zu erzeugen, bei dem die Periode und die anteilige Einschaltzeit programmierbar sind.
- Die **serielle Schnittstelle** (USART, Pins TX und RX) kann in verschiedenen synchronen und asynchronen Modi Daten senden und empfangen. Dabei werden unter anderem alle Einstellungen unterstützt, die auch mit einer seriellen Schnittstelle im PC möglich sind.
- Der Interrupt-Mechanismus unterstützt die neu hinzugekommenen Module und kann so Interrupts bei Aktivität des CCP-Moduls oder nach dem Senden und Empfangen von Daten auslösen.
- Der 16F628 ist auch über ein **serielles Programmierinterface** in der Schaltung programmierbar, dieses besteht aus den Pins MCLR, RB6 und RB7, im sogenannten Low Voltage Programmiermodus wird zusätzlich noch der Pin PGM exklusiv für die Programmierung reserviert.

Dieser PIC eignet sich gut für Schaltungen, in denen viele externe Bauteile angeschlossen werden sollen. Da beide PICs den selben Kern enthalten, ist es relativ einfach, Programme von einem Mikrocontroller auf den anderen zu portieren. Häufig muß nur die Ansteuerung der Hardware und die Aufteilung des Speichers geändert werden.

3.2.3 Organisation des Speichers

Alle Mikrocontroller der PIC-Familie verfügen über eine Harvard-Architektur. Die Speicher für Programmcode und Daten sind physikalisch getrennt und liegen auch nicht im selben Adreßraum.

Der Programmspeicher besteht aus einer Anzahl von Speicherzellen mit einer Breite von 14 Bit, die jeweils genau eine Instruktion enthalten. Dabei handelt es sich um Flash-Speicher, der von einem externen Programmiergerät beschrieben werden muß, um Software in den PIC zu laden. Das laufende Programm kann weder lesend noch schreibend auf den Speicher zugreifen. Ein 13 Bit breiter Programmzähler wird zur Adressierung des Speichers benutzt, wobei einige der Adressen eine feste Bedeutung haben.

- 0x0000: Das Programm startet nach dem Einschalten und nach dem Reset an dieser Adresse.
- 0x0004: Beim Auftreten eines Interrupts wird diese Adresse angesprungen.
- 0x03ff: Bei dem 12F675 liegt am Ende des Programmspeichers eine retlw-Instruktion, die den Kalibrierwert für den internen Oszillator enthält. Das Programm kann diese mit einem call-Befehl aufrufen und bekommt so den Kalibrierwert im Prozessorregister übergeben. Der Wert kann zusammen mit dem übrigen Code geändert werden.

Das bei PICs vorgesehene Umschalten zwischen den Speicherbänken im Programmspeicher ist bei den beiden hier vorgestellten Modellen nicht nötig, da nur eine Bank vorhanden ist.

Der Datenspeicher, in der Terminologie der PICs File Register genannt, besteht aus SRAM und ist in 2 oder 4 Bänken von jeweils 128 Bytes organisiert. Die Speicherzellen einer Bank können frei vom Programm gelesen und geschrieben werden. Einige dieser Zellen dienen zur Steuerung der Hardware (Special Function Registers), der Rest steht dem Benutzerprogramm als RAM zur Verfügung (General Function Registers) oder ist physikalisch nicht vorhanden. Bestimmte Speicherstellen tauchen in mehreren Bänken auf, dazu gehört insbesondere das Statusregister des Prozessors, das in allen Bänken zu finden ist und unter anderem den Umschalter für die Bänke enthält. Die Adressierung einer Speicherstelle in der aktiven Bank erfolgt über einen 7 Bit breiten Offset.

Der Zugriff auf den EEPROM erfolgt über einen Controller, der über Special Function Registers gesteuert wird. Er liegt damit auch nicht im Adreßraum des Prozessors.

Bei dem Stack handelt es sich um einen Ringpuffer mit 8 Einträgen von je 13 Bit, die der Prozessor beim Aufruf von Unterprogrammen oder beim Auslösen eines Interrupts selbst verwaltet. Ein Zugriff durch den Benutzer ist nicht möglich, lokale Variablen und Parameterübergabe über den Stack im Stil einer Hochsprache sind damit genauso ausgeschlossen wie präemptives Multitasking.

Zusätzlich zu den genannten Speichern steht dem Programmierer noch das eine CPU-Register namens W offen. Es ist als Akkumulator ausgelegt und wird bei vielen arithmetischen Operationen gebraucht.

3.2.4 Konfiguration

Die Konfiguration des PICs und der Peripherieeinheiten erfolgt größtenteils zur Laufzeit per Software. Einige Einstellungen müssen bereits vorher durchgeführt werden, damit der PIC überhaupt starten kann. Dazu gehört beispielsweise die Auswahl der Taktquelle. Dazu dienen die sogenannten Configuration Fuses, eine Sequenz von 14 Steuerbits, die beim Programmieren gesetzt werden. Sie erlauben die Konfiguration der folgenden Funktionen:

- Auswahl und Einstellungen der Taktquelle
- Konfiguration des Pins MCLR für I/O oder als Reset-Eingang
- Konfiguration des Pins LVP für I/O oder das Programmierinterface

- Aktivierung des Watchdog-Timers
- Aktivierung Leseschutzes für Codespeicher und/oder EEPROM
- Aktivierung der Timer, die den Startvorgang steuern
- Aktivierung der Brown-Out Detection

Diese Liste enthält die häufigsten Einstellmöglichkeiten. Bei einem konkreten PIC müssen nicht alle dieser Optionen vorhanden sein, es können auch weitere hinzukommen.

3.3 Entwicklung der Steuersoftware

Es gibt eine Vielzahl von Entwicklungssystemen, mit denen Software für die PIC Mikrocontroller geschrieben werden kann. Neben dem natürlich vorhandenen Assembler gibt es einige wenige Compiler für Hochsprachen wie C und sogar einige grafische Modellierungswerkzeuge. Insgesamt sind die Hochsprachen aber wenig verbreitet, da die Architektur des PICs nicht kompatibel zu den Backends der Standardcompiler ist.

Assembler ist im Allgemeinen deutlich besser zum Programmieren geeignet. Der erzeugte Code ist kleiner, schneller und hat die größtmögliche Kontrolle über die Hardware. Außerdem ist der Befehlssatz mir nur 35 einfachen Instruktionen sehr überschaubar. Ein weiterer entscheidender Vorteil ist, daß das zeitliche Verhalten des Programms in Assembler exakt festgelegt werden kann, was bei den meisten Hochsprachen nicht möglich ist. Diese Eigenschaft ist aber für Echtzeitanwendungen unverzichtbar.

Beim Entwickeln in Assembler ist die prinzipielle Vorgehensweise immer die selbe. Zunächst wird der Quellcode in einem Editor eingegeben und dann compiliert. Der Assembler erzeugt ein Speicherabbild für den Mikrocontroller, in dem der Code, Daten für den EEPROM und die Configuration Fuses enthalten sind. Dieses wird in der Regel in einer Variante des Intel Hexadecimal Object Formates gespeichert (siehe [Intelhex]). Die Programmiersoftware liest das Speicherabbild ein und steuert ein an den Rechner angeschlossenes Programmiergerät an, um die Daten in den Speicher des Controllers zu brennen. Anschließend ist der PIC einsatzbereit und kann in seiner Zielschaltung den Betrieb aufnehmen. Alternativ kann das Speicherabbild auch am PC in Simulationssoftware geladen und getestet werden.

Die nötigen Programme gibt es separat und als integrierte Entwicklungsumgebungen für Linux und Windows, [Piclinks] zählt einige Quellen und weiterführende Informationen auf. Die Quellcodes in dieser Arbeit wurden für den Assembler aus den Kommandozeilenutilities der GNU PIC Utilities (siehe [Gputils]) erstellt. Als Programmiergerät kam eine Eigenentwicklung zum Einsatz, die in 3.5 vorgestellt wird und sich besonders gut dazu eignet, den PIC auch in der Schaltung zu programmieren.

3.4 Integration in eine Schaltung

Wenn die Entscheidung für einen bestimmten PIC getroffen ist, muß der Rest der Schaltung um den PIC herum entworfen werden. Dabei treten unabhängig vom PIC immer wieder die selben Baugruppen auf, die hier anhand eines Beispiels mit dem 12F675 vorgestellt werden sollen.

Für den Betrieb benötigt der PIC zunächst eine stabilisierte Stromversorgung von beispielsweise 5 V und den bei CMOS-ICs üblichen Bypasskondensator von 100 nF. Als Taktquelle wird ein 4 MHz-Quarz benutzt, dessen Anschlüsse jeweils mit dem PIC und über einen sehr kleinen Kondensator mit Masse verbunden sein müssen. Richtwerte für die Größe des Kondensators sind im Datenblatt des PICs angegeben, bei 4 MHz benötigt der 12F675 zwischen 15 und 68 pF. Der MCLR-Pin steuert die Reset-Funktion des PICs. Wird der Pin auf Masse gezogen, wechselt der PIC in den Reset-Zustand.

Für den normalen Betrieb muß ein positiver Pegel an MCLR anliegen. Diese Funktionen übernehmen ein Pullup-Widerstand und ein Taster. Schließlich wird noch eine Leuchtdiode mit passendem Vorwiderstand an den PIC angeschlossen. Sie dient als einfaches Ausgabegerät für den PIC.

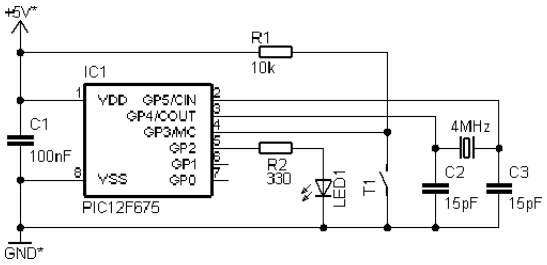


Abbildung 3.7: Grundschaltung mit dem PIC 12F675

Ein Steuerprogramm für diese Schaltung muß den Aufbau der Hardware berücksichtigen, wir das folgende Beispiel zeigt. Nach der Auswahl des Prozessors werden mit der `CONFIG`-Direktive die Configuration Fuses gesetzt. Als Oszillator wird ein Quarz von bis zu 4 MHz ausgewählt (`XT_OSC`), die Reset-Logik am MCLR-Pin wird aktiviert (`MCLRE_ON`) und der Watchdog-Timer ausgeschaltet (`WDT_OFF`). Nach der Deklaration einer Variablen beginnt das Hauptprogramm.

```
        ;; blinkled.asm, Stephan Höhrmann, 12.01.2004

        processor p12f675
        include   "p12f675.inc"
        __CONFIG _XT_OSC & _MCLRE_ON & _WDT_OFF

        ;; Variablen deklarieren
        cblock 0x20
               MEMDLY
        endc


        ;; Start des Programms
        org     0x0000
        goto    start
        org     0x0004

        ;; Pin GP2 als digitalen Ausgang konfigurieren
start   bcf     STATUS,RP0  ; Register File Bank 0 aktivieren
        clrf    GPIO        ; Alle Ausgänge vorab auf Low schalten
        movlw   B'00000111' ; Komparator ausschalten
        movwf   CMCON
        bsf     STATUS,RP0  ; Register File Bank 1 aktivieren
        clrf    ANSEL       ; Keine analogen Eingänge benutzen
        clrf    TRISIO      ; Alle Pins werden Ausgänge
        bcf     STATUS,RP0  ; Register File Bank 0 aktivieren

        ;; LED blinken lassen
loop    bsf     GPIO,GP2    ; LED einschalten
        movlw   D'250'      ; 0.25 Sekunden warten
        call    delay
        bcf     GPIO,GP2    ; LED wieder aus
        movlw   D'250'      ; 0.25 Sekunden warten
        call    delay
        goto    loop        ; Endlosschleife
```

```
        ;; Hilfsroutine delay
delay   movwf MEMDLY      ; Den Parameter in W in MEMDLY kopieren
dly1    movlw D'249'      ; Innere Schleife, w von 249 auf 0 runterzählen
dly2    addlw D'-1'
        bnz   dly2
        decf  MEMDLY,f    ; Äußere Schleife, MEMDLY auf 0 runterzählen
        bnz   dly1
        return

        end
```

Der Code beginnt damit, die Peripherie zu initialisieren. Alle verfügbaren Pins werden als digitale Ausgänge konfiguriert und auf 0 geschaltet, dann läßt das Programm die LED etwa in einer Schleife zweimal pro Sekunde aufblinken. Die Routine delay realisiert dabei die nötigen Pausen von 250 ms. Durch den folgenden Aufruf des Assemblers wird aus diesem Quellcode das Speicherabbild im Hex-Format erstellt, in dem neben den Instruktionen auch die Configuration Fuses enthalten sind:

```
gpasm -w1 -a inhx8m blinkled.asm
```

Ein an den PC angeschlossenes Programmiergerät wird benutzt, um das Speicherabbild in den PIC zu brennen. Dazu wird der Mikrocontroller in den Sockel des Gerätes gesteckt. Dieses löscht zunächst den gesamten Speicher des PICs und schreibt Programmcode, Configuration Fuses und gegebenenfalls Daten für den EEPROM-Speicher in die dafür vorgesehenen Bereiche. Der fertig programmierte Mikrocontroller kann in die Schaltung eingesetzt werden. Beim Einschalten der Stromversorgung startet sofort die Software und läßt die Leuchtdiode blinken.

Abbildung 3.8: Programmiergerät für PIC Mikrocontroller

Die Beschaltung des PICs mit Reset-Taster und Quarz kann geändert werden, falls die Aufgabe dies erfordert. Die Konfiguration muß dann entsprechend angepaßt werden.

- Der Quarz mit den beiden Kondensatoren kann durch einen Resonator ersetzt werden. Dabei handelt es sich um einen Baustein, der alle drei Teile in einem Gehäuse enthält. Die Taktfrequenz eines Resonators ist nicht so präzise wie die eines Quarzes, dafür ist er deutlich kleiner.

- Der Pin MCLR kann bei Bedarf als Eingang konfiguriert werden. Damit fällt allerdings auch die Möglichkeit weg, von außen einen Reset des PICs erzwingen zu können.
- Falls noch mehr Pins benötigt werden, kann statt des Quarzes auch der interne 4 MHz-Oszillator des PICs benutzt werden. Die Pins GP4 und GP5 sind dann frei, dafür unterliegt die Taktfrequenz etwas größeren Schwankungen.

Die Schaltung aus dem Beispiel kann einfach mit wenigen Teilen auf einer Steckplatine aufgebaut werden, wie in Abbildung 3.9 zu sehen ist. Als Taktquelle wurde ein 4 MHz-Resonator benutzt, und aus Gründen der Übersichtlichkeit wurde der Reset-Taster weggelassen.

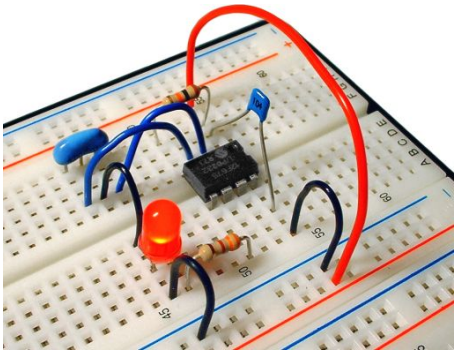


Abbildung 3.9: Beispielschaltung mit dem PIC12F675

Zum Entwickeln und Testen von Schaltungen mit Mikrocontrollern sind fertige Prototyp-Platinen allerdings deutlich besser geeignet. Sie vereinigen alle zum Betrieb des Controllers nötigen Komponenten auf einer Platine und lassen sich so konfigurieren, daß einfache Programme ohne weitere Bauteile lauffähig sind. Bei Bedarf ist auch der Anschluß externer Komponenten möglich. Entsprechende Platinen für die PICs mit 8 und 18 Pins sind in den Anhängen A.4 und A.5 beschrieben.

Der bisher beschriebene Entwicklungsweg hat den Nachteil, daß bei jeder Änderung der Software der PIC aus der Schaltung entnommen werden muß, um neu beschrieben zu werden. Deutlich einfacher ist es, wenn der PIC in der Schaltung selbst programmiert werden kann. Dann sind allerdings zusätzliche Vorkehrungen nötig, damit es keine Wechselwirkungen zwischen den Anschlüssen des Programmiergerätes und dem Rest der Schaltung gibt. Um entsprechenden Modifikationen vornehmen zu können, ist es nötig, den Ablauf des Programmiervorgangs zu kennen.

3.5 Aufbau von Programmiergeräten

Zum Beschreiben eines Mikrocontrollers sind ein Programmiergerät nötig, daß die entsprechenden Signale erzeugen und an den PIC leiten kann, sowie eine Software, die den Vorgang steuert. Der folgende Abschnitt beschreibt die grundlegende Funktionsweise eines solchen Gespanns aus Hardware und Software, wie sie für fast alle PICs gültig ist.

Das Programmiergerät muß mit dem Mikrocontroller über die 5 Leitungen VCC, MCLR, Data, Clock und GND verbunden sein. Eine feste Sequenz bringt den PIC in einen Modus, in dem er Befehle und Daten entgegennimmt, und leitet somit den Programmiervorgang ein.

- Der MCLR-Pin wird auf Masse gezogen. Dadurch geht der PIC in den Reset-Zustand und alle I/O-Pins werden hochohmig, was den Mikrocontroller praktisch vom Rest der Schaltung isoliert.
- Das Programmiergerät versorgt den PIC über die Leitung VCC mit Strom.

- Anschließend wird der MCLR-Pin auf die Spannung VPP gebracht, typischerweise 13,2 V. Der PIC wechselt jetzt in den Programmiermodus und wartet auf weitere Befehle. Bei älteren Modellen dient die Spannung an MCLR direkt zum Schreiben des Flash-Speichers und wird dementsprechend mit 50-100 mA belastet, neuere Modelle erzeugen die Spannung intern aus der normalen Versorgungsspannung und belasten VPP praktisch nicht.
- Das Programmiergerät sendet jetzt Befehle zum Löschen, Schreiben, Lesen oder Verifizieren des Speichers seriell über die Kommunikationsleitungen Data und Clock. Der PIC kann die Datenleitung ebenfalls benutzen, um Antworten zu senden. In beiden Fällen wird der Takt vom Programmiergerät vorgegeben.
- Nach dem Abschluß des Vorgangs wird noch ein Reset ausgelöst. Der Mikrocontroller ist damit beschrieben und bereit, die neue Software auszuführen.

Das Timing und die Details des Kommunikationsprotokolls unterscheiden sich von PIC zu PIC etwas. Die Details sind in entsprechenden Datenblättern des jeweiligen Modells festgehalten (EPROM Memory Programming Specification).

3.5.1 In-Circuit Programmierung

Nahezu alle Mikrocontroller aus der PIC-Familie können auch in der Anwendungsschaltung umprogrammiert werden. Microchip nennt diese Technik In-Circuit Serial Programming, oder kurz ICSP. Durch sie läßt sich beim Entwickeln eine Menge Zeit einsparen, es sind dafür jedoch ein geeignetes Programmiergerät und einige Anpassungen in der Schaltung nötig. Einerseits darf das Programmiergerät nur auf den PIC und nicht auf den Rest der Schaltung einwirken, andererseits darf die Schaltung die zum Programmieren nötigen Operationen nicht stören.

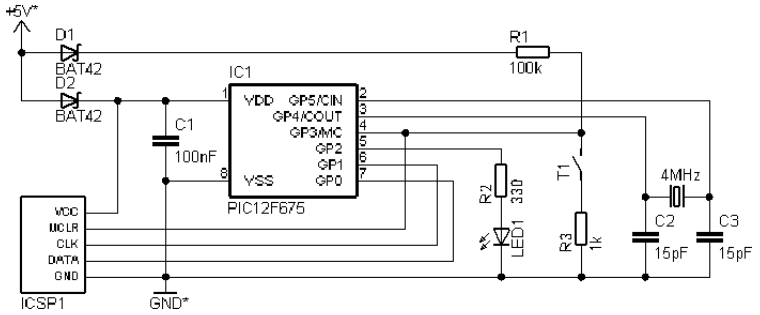


Abbildung 3.10: Einfache Beispielschaltung mit ICSP-Interface

Die Grundschaltung für PICs aus 3.7 kann relativ einfach ICSP-fähig gemacht werden. Dazu wird sie um eine Anschlußbuchse für das Programmiergerät ergänzt und an einigen Stellen leicht verändert, wie in Abbildung 3.10 zu sehen ist.

- Das Programmiergerät muß in der Lage sein, den MCLR-Pin auf Masse zu ziehen. Sein Innenwiderstand bildet dabei mit R_1 einen Spannungsteiler, also muß R_1 ausreichend groß gewählt werden. Seine Größe wurde daher auf 100 kΩ erhöht.
- Der Widerstand R_3 verhindert, daß das Programmiergerät über den Reset-Taster kurzgeschlossen werden kann. Er muß für seine Funktion deutlich kleiner als R_1 gewählt werden.
- Die Diode D_1 schützt die Zielschaltung vor der Programmierspannung. Wenn der MCLR-Pin auf 13,2 V gebracht wird, könnte diese Spannung sonst über R_1 in andere Teile der Schaltung gelangen und sie beschädigen.

- D_2 erfüllt die selbe Aufgabe für die Versorgungsspannung. Das Programmiergerät würde sonst die gesamte Zielschaltung mit Strom versorgen, was seine Treiberkapazität überschreiten kann. Die Spannung am Mikrocontroller verringert sich durch die Diode etwas, so daß für D_2 eventuell eine Schottky-Diode benutzt werden muß.
- Am schwierigsten ist der Anschluß der Leitungen Data und Clock. Wenn die dazugehörigen Pins des PICs in der Schaltung gebraucht werden, kann dies zu Schwierigkeiten führen. Unter Umständen stören die an diesen Pins angeschlossenen Bauteile die serielle Kommunikation während der Programmierung, oder das Programmiergerät löst in den Bauteilen ungewollte Aktionen aus. Es muß sichergestellt sein, daß beides nicht passieren kann. In dieser einfachen Schaltung war das nicht nötig, die Pins werden exklusiv von der ICSP-Schnittstelle benutzt.

Im Allgemeinen müssen die Leitungen Data und Clock sich die Pins mit der Anwendungsschaltung teilen. Eine geeigneter Aufbau ist dann nötig, um die beiden Funktionen zu trennen. Dies kann zum Beispiel durch große Widerstände oder Treiberbausteile erfolgen. Die universellste Methode ist jedoch, eine mechanische Umschaltmöglichkeit vorzusehen, beispielsweise in Form von Jumpern.

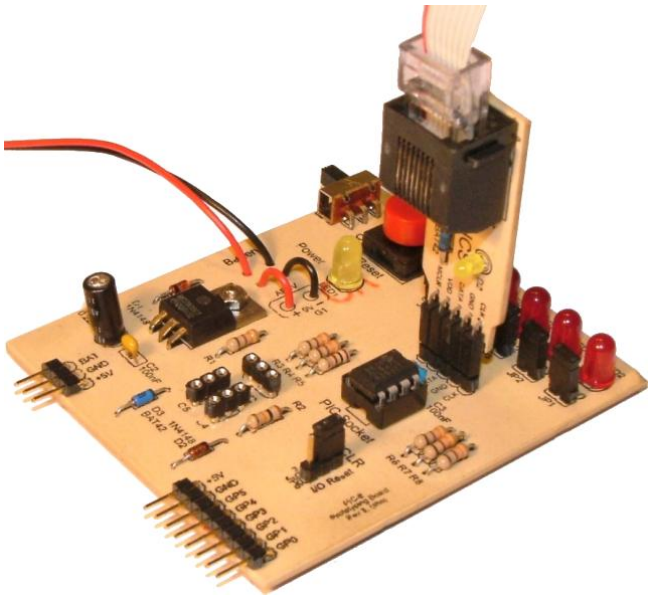


Abbildung 3.11: Schaltung mit ICSP-Schnittstelle zum Programmiergerät

Wenn das Programmiergerät zusätzlich noch in der Lage ist, seine Anschlüsse bei Inaktivität hochohmig zu schalten, ist ein Umprogrammieren des PICs sogar im laufenden Betrieb möglich. Anhang A.6 beschreibt den Aufbau eines geeigneten Programmiergerätes. In Anhang A.7 wird eine dazu passende ICSP-Schnittstelle vorgestellt, die eine mechanische Umschaltmöglichkeit für Clock und Data vorsieht, und nur minimale Veränderungen an der Anwendungsschaltung erfordert.

3.5.2 Low Voltage Programming

Das bisher vorgestellte Schema zum Programmieren eines PICs heißt nach der relativ hohen Spannung, die zum Aktivieren des Programmiermodus gebraucht wird, High Voltage Programming (HVP). Einige der neueren PICs unterstützen zusätzlich noch ein Schema namens Low Voltage Programming (LVP), bei dem nur Spannungen von 5 V benötigt werden.

Dadurch vereinfacht sich der Aufbau der Programmiergeräte deutlich, andererseits wird ein I/O-Pin des Prozessors exklusiv zum Aktivieren des Programmiermodus reserviert. Bei den meisten Applikationen ist aber gerade die Anzahl der I/O-Pins so wichtig, daß diese Einschränkung inakzeptabel ist. Außerdem lassen sich alle PICs auch mit HVP beschreiben, so daß die meisten Programmiergeräte sich darauf beschränken.

Bei allen PICs mit LVP-Unterstützung ist diese werksseitig aktiviert, der LVP-Pin ist als digitaler Eingang konfiguriert und löst aktiviert bei einer steigenden Flanke sofort den Programmiermodus. Es bietet sich daher an, den LVP-Modus vor dem Einbau in eine Schaltung zu deaktivieren. Dazu wird ein Programm in den Controller geladen, bei dem LVP in den Configuration Fuses deaktiviert ist. Ohne diesen Schritt ist es möglich, daß sich der Mikrocontroller nicht per ICSP programmieren läßt.

3.5.3 Auswahl der Programmiersoftware

Zum Betrieb eines Programmiergerätes wird natürlich eine passende Software benötigt. Hersteller wie Mikrochip liefern daher eine für ihre Geräte passende Software mit. Bei den im Internet weit verbreiteten Geräten für den Parallelport hat sich ein Standard herausgebildet, der auf [Tait] zurückgeht. Das verwendete Protokoll ist einheitlich genug, daß ein entsprechendes Programm eine Vielzahl verschiedener Brenner ansteuern kann. Eventuelle Unterschiede lassen sich bei solchen Programmen durch entsprechende Konfigurationseinstellungen kompensieren.

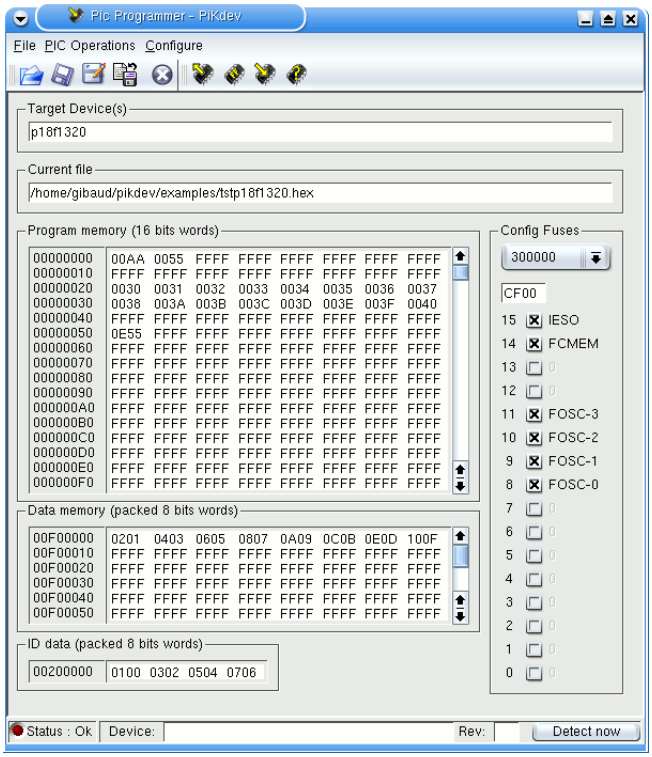


Abbildung 3.12: Programmiermodul aus PiKdev

Zu diesen Programmen gehören zum Beispiel PBrenner von [Bredendiek] für Windows sowie das in PiKdev enthaltene Brennmodul für Linux (siehe [Piclinks]). Ein weiteres Kommandozeilenprogramm für Linux ist im Anhang B.5 beschrieben. Es beschreibt einen PIC schneller als die anderen Programme, bietet mehr Optionen und ist auf gute ICSP-Unterstützung ausgelegt. Dafür unterstützt es nur den Tait-Standard und nur wenige PICs, darunter sind aber die in dieser Arbeit eingesetzten Modelle.

3.6 Serielles Debug-Interface

Zu den wichtigsten Werkzeugen bei der Softwareentwicklung gehört neben Editor und Compiler ein Debugger, mit dem das laufende Programm untersucht und auf Fehler überprüft werden kann. Bei Mikrocontrollern kann ein Simulator diese Aufgabe übernehmen, mit dem die Software schrittweise ausgeführt und untersucht werden kann. Dabei ist es allerdings nötig, auch die gesamte Peripherie des Mikrocontrollers für den Simulator nachzubilden, was einen erheblichen Arbeitsaufwand bedeuten kann. Einige Fehler treten außerdem nur in der Schaltung und unter Echtzeitbedingungen auf.

Sinnvoller ist es also, Befehle zum Ausgeben von Statusmeldungen oder Programmvariablen in die eigentliche Software einzubauen und sie auf der echten Hardware ausführen zu lassen. Dafür wird jedoch ein geeignetes Ausgabegerät benötigt. Diese Aufgabe kann ein PC sehr gut übernehmen. Er kann die Meldungen darstellen, zwischenspeichern und programmgesteuert auswerten, um nach eventuellen Auffälligkeiten zu suchen. Der folgende Abschnitt beschreibt eine Lösung, die nach diesem Prinzip funktioniert und Daten über die serielle Schnittstelle an den PC sendet.

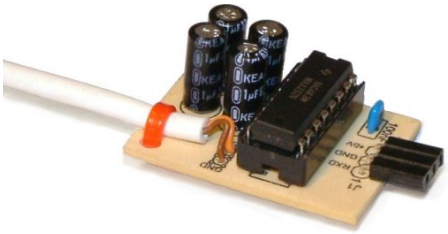


Abbildung 3.13: Serielles Debugging-Interface

Ein Pin des Mikrocontrollers wird als Ausgang konfiguriert und über einen Pegelwandler mit der seriellen Schnittstelle des PCs verbunden. Das Senden von Daten übernimmt ein Softwaremodul, das in das zu untersuchende Programm eingebunden wird. Anschließend können verschiedene Makros benutzt werden, um Informationen vom PIC zum PC zu senden. Der PC decodiert die empfangenen Nachrichten und zeigt sie an. Im Assemblerprogramm können die folgenden Makros aufgerufen werden:

- **usesdbg** bindet alle erforderlichen Routinen in den Code ein
- **sdinit** setzt den gewählten Ausgangspin vor der ersten Übertragung auf den Ruhepegel.
- **sdsendw** sendet den Inhalt des W-Registers an den PC. Ein Flag steuert, wie die Ausgabe erfolgen soll. Zur Wahl stehen die Formate binär, hexadezimal, dezimal mit und ohne Vorzeichen sowie als Zeichen. Es ist außerdem über ein spezielles Flag möglich, mehrere Bytes zu einem Wert zusammenzusetzen, bevor dieser ausgegeben wird.
- **sdsendf** entspricht sdsendw, übermittelt allerdings den Inhalt einer Speicherzelle.
- **sdinfo** sendet eine beliebige statische Textnachricht.
- **sdstop** signalisiert dem Monitorprogramm auf dem PC, daß es sich beenden soll.

Die nötige Hardware in Form des Pegelwandlers ist auf einer Adapterplatine untergebracht, die auf die zu untersuchende Schaltung gesteckt werden kann. Ihr Aufbau ist im Anhang A.8 beschrieben, während Anhang B.6 die Benutzerschnittstelle, das Übertragungsprotokoll und das Monitorprogramm für Linux beschreibt.

Kapitel 4

Ortung mit Ultraschall

Viele Lego-Roboter sind autonome Fahrzeuge und werden komplett vom integrierten RCX gesteuert, ohne Anweisungen von außen zu erhalten. Sie bewegen sich mit Hilfe ihrer Motoren in einer unbekannten Umgebung, um eine Aufgabe zu erfüllen. In der Regel müssen sie zumindest eine grobe Vorstellung haben, wo sie sich befinden und welche Strecken sie seit dem Start zurückgelegt haben. Lego hat jedoch keine Sensoren vorgesehen, mit denen ein Roboter seine Position direkt feststellen kann.

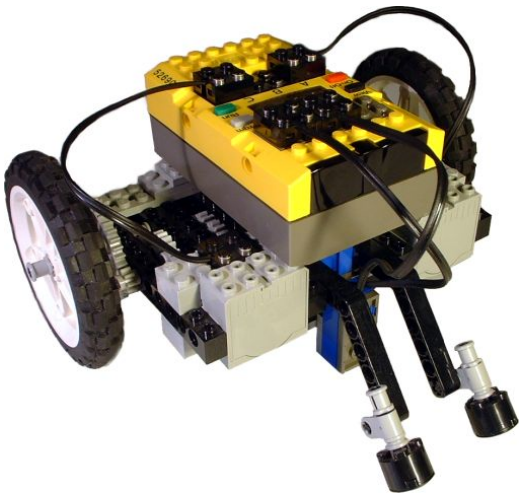


Abbildung 4.1: Autonomer, mobiler Lego-Roboter

Am häufigsten wird daher der Ausweg gewählt, die Position zu extrapolieren. Ausgehend von einem bekannten Startpunkt integriert das Steuerprogramm alle Bewegungen seiner Aktoren auf, um so eine Schätzung der aktuellen Position zu erhalten. Die Berechnungen können im einfachsten Fall auf der reinen Laufzeit der Motoren und einer bekannten Durchschnittsgeschwindigkeit basieren.

Das Verfahren wird zuverlässiger, wenn Sensoren an der Antriebsmechanik angebracht werden, deren Daten mit Hilfe eines physikalischen Modells des Roboters ausgewertet werden (siehe [Borenstein]). Diese Technik heißt Koppelnavigation (engl: dead reckoning). Ihr größter Nachteil ist, daß kleine Fehler nicht korrigiert werden können und im Laufe der Zeit ohne jede Schranke anwachsen.

Wesentlich besser sind Verfahren geeignet, welche die absolute Position zu jeder Zeit bestimmen können, statt sich auf relative Änderungen zu stützen. Dazu müssen im Aktionsradius des Roboters Stationen an bekannten Positionen aufgestellt werden, die mit den Sensoren erfaßt werden können. Der Roboter mißt die Entfernung zu diesen Leuchtfeuern oder die Richtung, in der sie sich befinden. Aus den Meßdaten und den Koordinaten der Leuchtfeuer kann der Roboter seine Position berechnen.

Es gibt zwei grundsätzlich verschiedene Methoden, wie eine Ortsbestimmung auf diese Art funktioniert.

- Bei der Triangulation peilt der Roboter von seinem Standort aus alle Leuchtfeuer an und bestimmt die Winkel zwischen den Peilungen. Über trigonometrische Berechnungen läßt sich aus den Winkelangaben und den Koordinaten der Leuchtfeuer die Position des Roboters eindeutig bestimmen. Für eine Ortsbestimmung in der Ebene sind drei Leuchtfeuer erforderlich.
- Bei der Trilateration bestimmt der Roboter seine Entfernung zu allen Leuchtfeuern. Um die Koordinaten jedes Leuchtfeuers wird ein Kreis gelegt, dessen Radius der Entfernung vom Leuchtfeuer zum Roboter entspricht. Die Kreise schneiden sich in einem Punkt, an dem sich folglich der Roboter befinden muß. Auch hier sind im Allgemeinen drei Leuchtfeuer für eine eindeutig bestimmte Ortung in der Ebene erforderlich.

Die Triangulation wird hauptsächlich in der nautischen Navigation eingesetzt. Ihr Nachteil ist, daß das Anpeilen eines einzelnen Punktes mit genauer Winkelmessung sehr aufwendig ist. Ein Sensor für diese Aufgabe ist schwierig zu konstruieren, und der Roboter darf sich während einer Ortung nicht bewegen. Für eine praktische Realisierung ist also die Trilateration besser geeignet. Einzige technische Voraussetzung ist die Möglichkeit, die Entfernung zwischen zwei Punkten messen zu können.

Der Aktionsradius von Lego-Robotern ist in der Regel durch die Größe eines Zimmers und damit auf wenige Meter beschränkt. Die Entfernungsmessung muß also nur kurze Strecken bestimmen können, die dafür aber möglichst genau. Der in dieser Arbeit gewählte Lösungsansatz ist, ein Ultraschallsignal von jedem der Leuchtfeuer zum Roboter zu senden und die Laufzeit des Schalls zu messen. Aus dieser Zeit und der Schallgeschwindigkeit läßt sich die Entfernung ausrechnen.

4.1 Basistechnik Entfernungsmessung

Als Ultraschall bezeichnet man Schallwellen mit einer Frequenz von mindestens 20 kHz, die vom menschlichen Ohr nicht mehr wahrgenommen werden. Ihre Ausbreitungsgeschwindigkeit ist stark vom Medium abhängig, in Luft liegt sie unter Normalbedingungen (0°C, 1013 hPa) bei c = 331 m/s, was etwa einem Zentimeter in 30 Mikrosekunden entspricht. Die Wellenlänge nimmt mit steigender Frequenz ab, bei 40 kHz liegt sie beispielsweise bei rund 8 Millimetern.

$$\lambda \;=\; \frac{c}{f} \;\leq\; \frac{331\ \frac{m}{s}}{40000\ \text{Hz}} \;=\; 8,275\ \text{mm} \tag{4.1}$$

Eine Reihe von Gründen spricht dafür, Ultraschall zur Messung kurzer Strecken einzusetzen.

- Durch die kurze Wellenlänge breiten sich die Schallwellen geradlinig aus. Außerdem lassen sie sich bei Bedarf gut reflektieren und bündeln.
- Die Schallwellen lassen sich einfach und kostengünstig erzeugen und detektieren.
- Die Ausbreitungsgeschwindigkeit ist ausreichend für eine schnelle Messung und trotzdem niedrig genug, so daß die Zeitmessung im Empfänger nicht mit sehr hoher Auflösung erfolgen muß.

Die Entfernung zwischen zwei Punkten wird bestimmt, indem ein kurzer Ultraschallimpuls (sogenannter Burst) von einem Punkt zum anderen gesendet wird. Aus der Laufzeit t des Signals und dessen Geschwindigkeit c kann die Entfernung d berechnet werden.

$$d \;=\; c \cdot t \tag{4.2}$$

Die Ausbreitungsgeschwindigkeit in Luft hängt in sehr guter Näherung nur von der Temperatur T ab, nicht jedoch vom Luftdruck oder der Frequenz des Schalls. Es gilt die Beziehung (4.3).

$$c(T) \quad = \quad \sqrt{\frac{\kappa \cdot p_0 \cdot T}{\rho_0 \cdot 273,15}} \tag{4.3}$$

Dabei ist T die Temperatur in Kelvin, p_0 der Normaldruck von 101325 Pa, $\kappa = 1{,}402$ der Adiabatenexponent für Luft und $\rho_0 = 1{,}2935$ kg/m^3 die mittlere Dichte der Luft unter Normalbedingungen. Das Diagramm in Abbildung 4.2 gibt die Schallgeschwindigkeiten für typische Umgebungsbedingungen an. Der Kurvenverlauf entspricht auf dem gewählten Intervall näherungsweise einer Geraden.

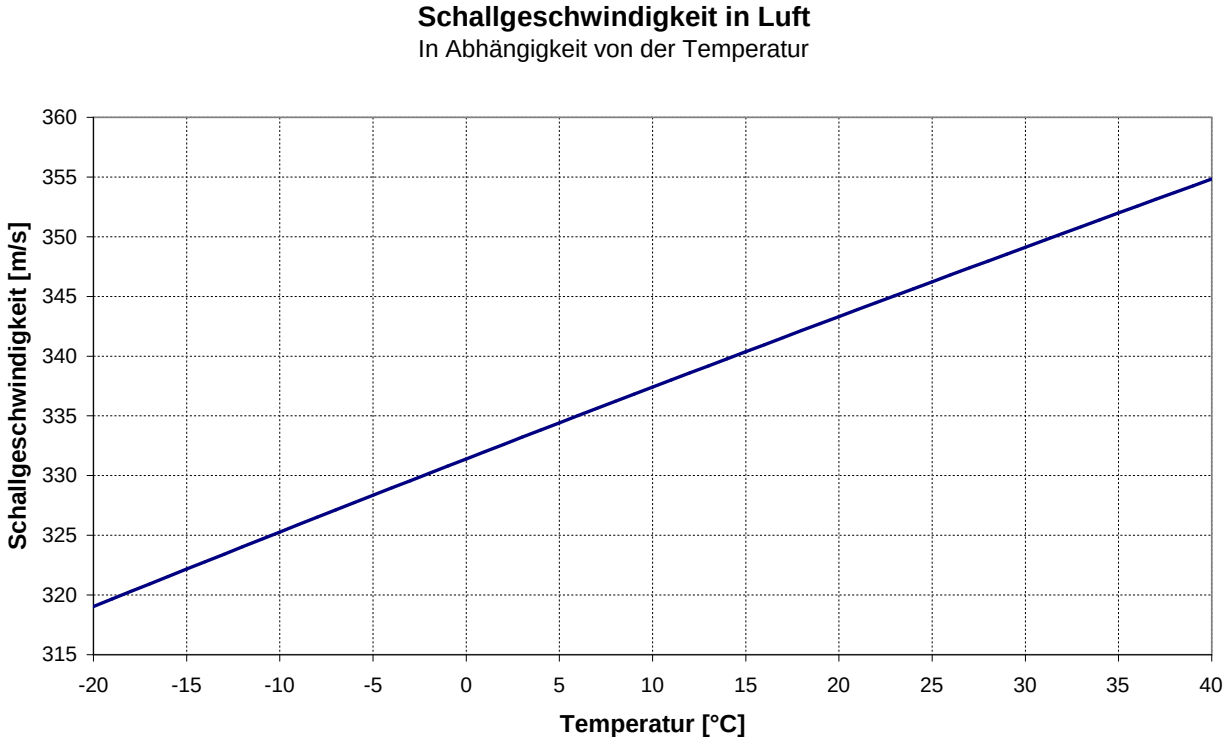


Abbildung 4.2: Isoliniendiagramm der Schallgeschwindigkeiten in Luft

Normalerweise sind Sender und Empfänger physikalisch getrennt und verfügen auch über keine gemeinsame Systemzeit. Wenn der Empfänger einen Burst registriert, weiß er daher nicht, wann dieser genau gesendet wurde. Das ist für eine Entfernungsmessung aber unverzichtbar. Der Empfänger muß den Sendezeitpunkt also auf einem anderen Weg ermitteln.

- Der Sender kann gleichzeitig zwei Signale mit unterschiedlichen Ausbreitungsgeschwindigkeiten aussenden, worauf der Empfänger die Zeitdifferenz Δt zwischen dem Eintreffen beider Signale mißt. Wenn t die (dem Empfänger unbekannte) absolute Laufzeit des schnelleren Signals ist, gilt $d = c_1 \cdot t$. Für das langsamere Signal ist entsprechend $d = c_2 \cdot (t + \Delta t)$. Aus diesen beiden Formeln kann die Entfernung durch Elimination von t berechnet werden.

  $$d \quad = \quad \frac{c_1 \cdot c_2}{c_1 - c_2} \cdot \Delta t \tag{4.4}$$

  Wenn c_1 sehr groß gegenüber c_2 ist, wie es zum Beispiel bei einem Funk- und einem Schallsignal der Fall ist, vereinfacht sich diese Formel zu $d \approx c_2 \cdot \Delta t$.

- Der Sender kann aus mehreren räumlich getrennten Leuchtfeuern bestehen, die nacheinander in einem bekannten Zeitschema senden. Der Empfänger registriert die Zeitdifferenz zwischen dem Eintreffen der verschiedenen Impulse und berechnet daraus die Entfernung. Die Vorgehensweise bei dieser Rechnung wird später in diesem Kapitel an Beispielen vorgestellt.

Beide Verfahren sind praktisch realisierbar und haben jeweils Vorteile und Nachteile. Sie werden im Laufe dieses Kapitels weiter untersucht und verglichen.

4.2 Erzeugen und Detektieren von Ultraschall

Ultraschallsender und Empfänger sind relativ einfach zu bauen. Die dafür nötigen keramischen Wandler gibt es im Elektronikhandel fertig zu kaufen, wie sie in Abbildung 4.3 zu sehen sind. Die wichtigsten technischen Daten eines typischen Wandlerpaares, bestehend aus Sender und Empfänger, sind in Tabelle 4.1 zusammengefaßt (entnommen aus dem Datenblatt von [Marquardt]).

Abbildung 4.3: Kapseln zum Erzeugen und Detektieren von Ultraschall

Die Sendekapsel wird von einer angelegten Wechselspannung U_S der richtigen Frequenz in Schwingung versetzt und strahlt daraufhin Schallwellen durch das Gitter an ihrer Oberseite ab. Trifft die Schallwelle in der Entfernung d auf eine Empfängerkapsel, gerät diese in Resonanz und erzeugt eine sinusförmige Wechselspannung U_E an ihren Anschlüssen.

Abmessung	**Größe**
Durchmesser	16 mm
Höhe	12 mm (ohne Anschlüsse)
Resonanzfrequenz	40 kHz
Maximaler Schalldruck	117 dB in 30 cm Entfernung
Empfindlichkeit	-67 dB
Maximale Treiberspannung	20 V_{RMS}
Öffnungswinkel Schallkeule	50°

Tabelle 4.1: Typische technische Daten eines Ultraschallwandlers

Sowohl beim Senden als auch beim Empfangen gilt, daß die Schallenergie an der Kapsel proportional zum Quadrat der anliegenden Spannung ist. Diese Energie fällt mit dem Quadrat der Entfernung. Wenn die Kapseln also optimal aufeinander ausgerichtet sind und der Schallweg frei von Hindernissen ist, gilt für die Spannungen Gleichung (4.5).

$$U_E^2 \sim \frac{U_S^2}{d^2} \quad \text{bzw.} \quad U_E \sim \frac{U_S}{d} \tag{4.5}$$

Abbildung 4.4 zeigt für eine exemplarische Messung, daß die am Empfänger gemessene Spannung proportional zur Sendespannung ist, wenn der Abstand der beiden konstant bleibt.

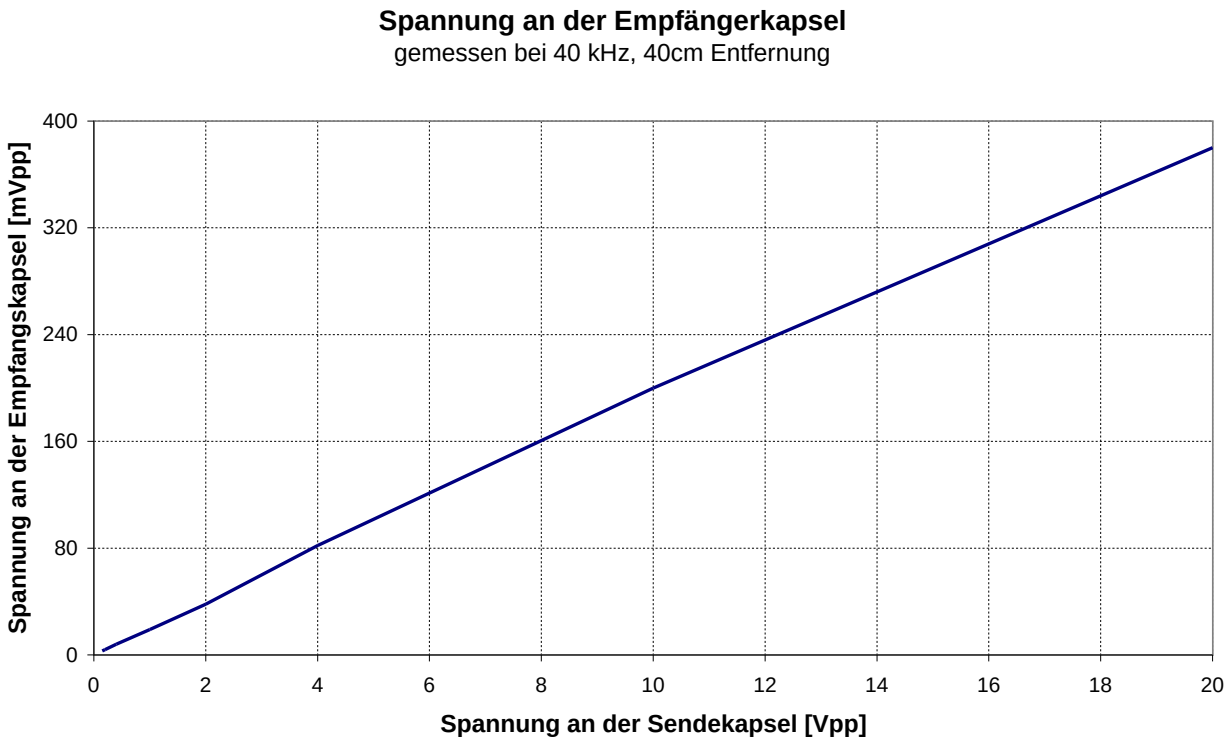


Abbildung 4.4: Spannungen an Sender und Empfänger bei konstantem Abstand

Des weiteren belegt Abbildung 4.5, daß die Spannung am Empfänger mit dem Kehrwert der Entfernung zum Sender fällt. Diese Messung wurde mit konstanter Signalstärke am Sender durchgeführt.

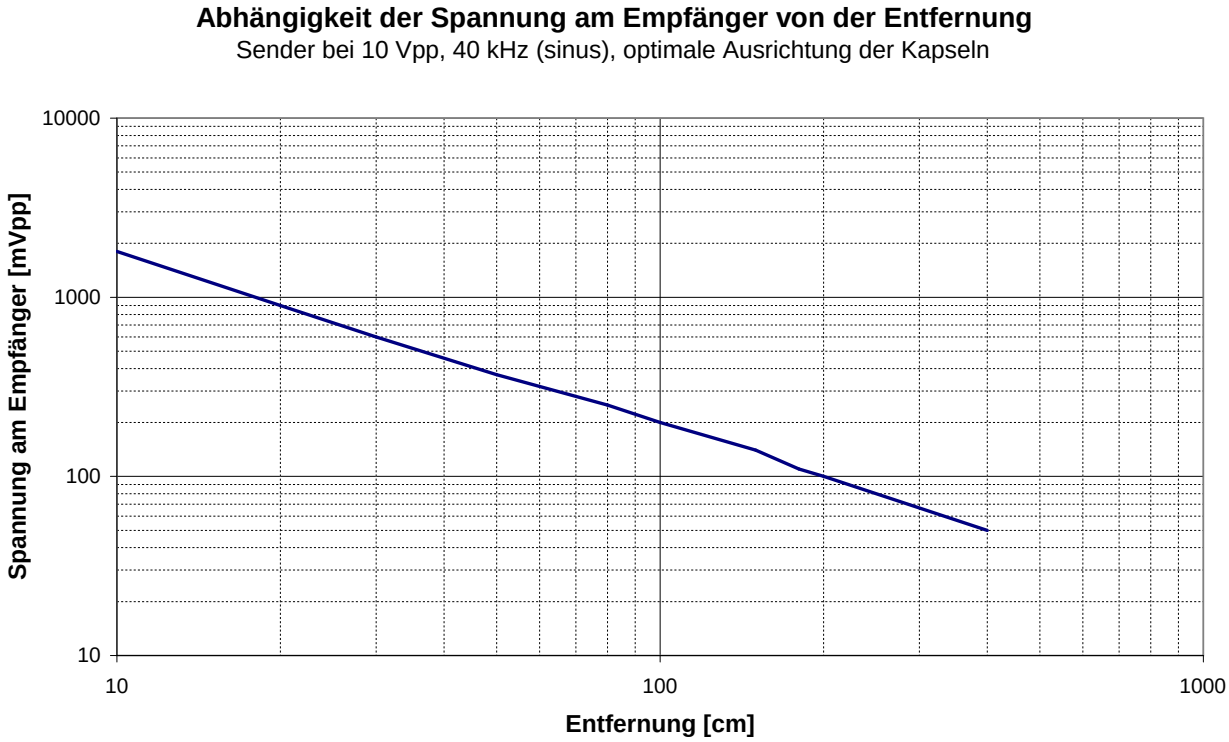


Abbildung 4.5: Veränderung der Signalstärke bei verschiedenen Abständen

Die Konsequenzen für den Aufbau eines Ortungssystems sind einfach. Die Reichweite läßt sich verdoppeln, indem entweder die Sendespannung oder der Verstärkungsfaktor im Empfänger verdoppelt wird, und beides läßt sich verhältnismäßig einfach erreichen.

Der größte Nachteil der Kapseln ist ihre Richtungsabhängigkeit. Der höchste Schalldruck und die größte Empfindlichkeit werden nur in Richtung der Längsachse erreicht. Bereits 25° neben dieser Linie verschlechtern sich die Werte auf die Hälfte (laut Datenblatt von [Marquardt]). Um jeden Punkt des abzudeckenden Gebiets erreichen zu können, müssen die Schallwellen daher durch Reflektoren entsprechend umgelenkt und gestreut werden.

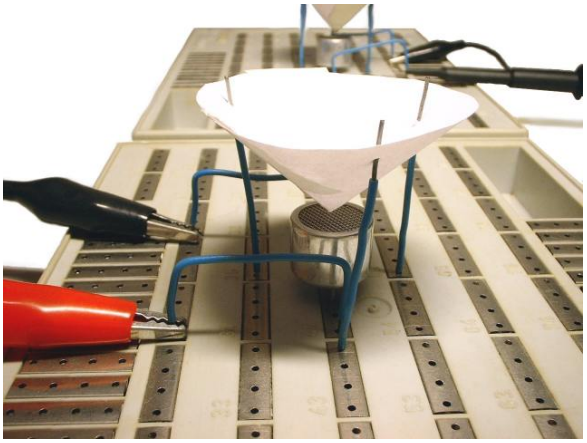


Abbildung 4.6: Reflektoren lenken den Ultraschall um

Bringt man einen kegelförmigen Reflektor über der Sendekapsel an, wie Abbildung 4.6 zeigt, breitet der Schall sich hauptsächlich in einer Ebene parallel zum Boden aus. Ein genauso gebauter Reflektor über der Empfängerkapsel lenkt den Schall aus der Ebene in die Kapsel um. Damit ist das Problem der Richtungsabhängigkeit gelöst, allerdings müssen Sender und Empfänger in einer Ebene liegen, und durch die Streuung sinkt die Signalstärke und damit die maximale Reichweite des Systems.

4.3 Optimale Geometrie eines Reflektors

Die Form und Größe des Reflektors ist also entscheidend, damit Sender und Empfänger in der Ebene richtungsunabhängig arbeiten können. Der folgende Abschnitt erklärt, wie eine optimale Form aussieht und wie sie aus gegebenen Parametern berechnet werden kann.

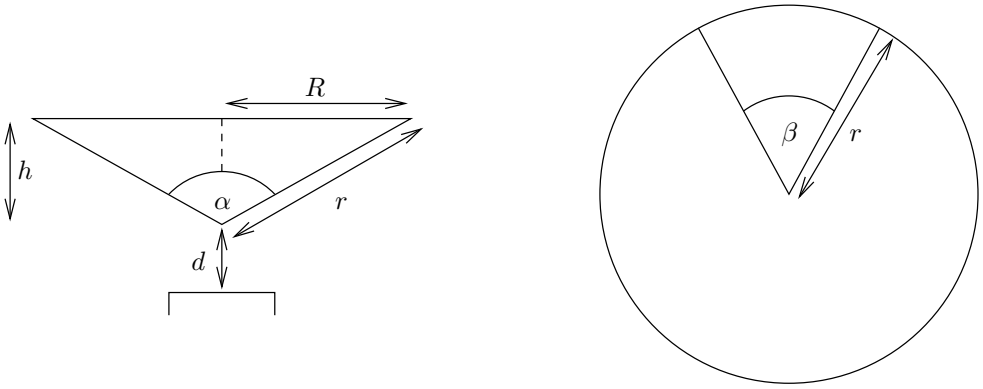


Abbildung 4.7: Parameter für die Berechnung der Reflektorenform

Die Abbildung 4.7 stellt die wichtigsten Parameter der Reflektorgeometrie dar. Dabei handelt es sich in erster Linie um den Öffnungswinkel α und die Kantenlänge r, wichtig für die Berechnung ist noch der geplante Abstand d von der Reflektorspitze zur Ultraschallkapsel. Aus den gegebenen Parametern lassen sich die Höhe h und der Öffnungsradius R des Kegels einfach berechnen.

$$h = r \cdot \cos\left(\frac{\alpha}{2}\right) \tag{4.6}$$

$$R = r \cdot \sin\left(\frac{\alpha}{2}\right) \tag{4.7}$$

Einen Reflektor mit diesen Maßen läßt sich herstellen, indem aus einer Kreisscheibe mit dem Radius r ein Segment der Größe β herausgeschnitten wird. Das verbleibende Stück wird an den Schnittstellen zusammengeklebt und bildet den gewünschten Kegel. Der Winkel β kann über den Umfang U des Kegelbodens berechnet werden.

$$\begin{aligned} U = 2\pi R &= 2\pi r - \beta r \\ \beta &= 2\pi\left(1 - \frac{R}{r}\right) \\ \beta &= 2\pi\left(1 - \sin\left(\frac{\alpha}{2}\right)\right) \end{aligned} \tag{4.8}$$

Ein Reflektor gilt als optimal, wenn ein Strahl, der den Kegel horizontal in der Mitte trifft, genau in die Mitte der Ultraschallkapsel reflektiert wird. Abbildung 4.8 definiert alle dabei auftretenden Winkel.

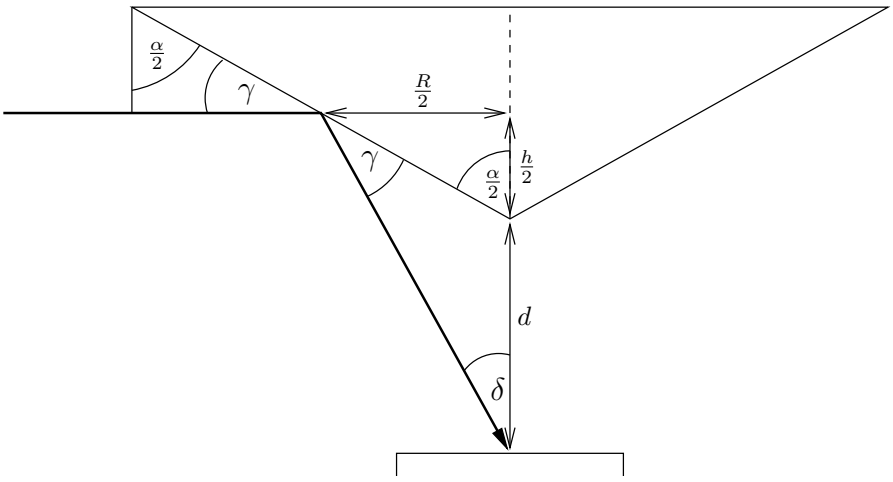


Abbildung 4.8: Berechnung der optimalen Reflektorform

Über die Winkelsumme der verschiedenen Dreiecke lassen sich die Winkel γ und δ berechnen.

$$\gamma = \frac{\pi}{2} - \frac{\alpha}{2} \tag{4.9}$$

$$\delta = \pi - \left(\pi - \frac{\alpha}{2}\right) - \gamma = \alpha - \frac{\pi}{2} \tag{4.10}$$

$$\tan(\delta) = \frac{\frac{R}{2}}{d + \frac{h}{2}} = \frac{R}{2d + h}$$

$$\tan\left(\alpha - \frac{\pi}{2}\right) = \frac{R}{2d + h} \tag{4.11}$$

Gleichung (4.11) setzt alle relevanten Außenmaße des Reflektors zueinander in Beziehung und beschreibt dadurch die Geometrie eines optimalen Reflektors.

Der Reflektor kann maximal eine Wellenfront der Breite h reflektieren, die ihn horizontal trifft. Die Wellenfront ändert dabei ihre Richtung, nicht jedoch ihre Breite, und bewegt sich auf die Empfängerkapsel zu. Es macht keinen Sinn, einen Reflektor zu verwenden, dessen Höhe größer als der Kapseldurchmesser ist. Die reflektierte Wellenfront wäre sonst breiter als die Kapsel und würde nur teilweise aufgefangen. Andererseits muß der Reflektor auch deutlich größer als die Wellenlänge des Schalls sein, damit er funktionieren kann, [Turowski] gibt eine Kantenlänge von mindestens 22 mm an. Hier gilt es also, einen Kompromiß zu finden.

Abmessung		**Größe**
Öffnungswinkel des Kegels	α	120°
Abstand zur Ultraschallkapsel	d	14 mm
Kantenlänge des Kegels	r	28 mm
Durchmesser des Kegelbodens	2R	48,5 mm
Höhe des Reflektorkörpers	h	14 mm
Auszuschneidendes Segment	β	48,2°

Tabelle 4.2: Beispielmaße für einen Reflektor

In einer Versuchsreihe mit sehr vielen verschiedenen Kegelformen wurde experimentell verifiziert, daß es sich bei der beschriebenen Form tatsächlich um einen Reflektor mit möglichst guten Praxiseigenschaften handelt. Tabelle 4.2 gibt die Maße des Reflektors an, der für die Umsetzung des Ortungssystems verwendet wurde.

4.4 Fehlerquellen beim Messen

Nachdem das Ultraschallsignal von der Sendekapsel erzeugt und über den Sendereflektor umgelenkt wurde, legt es die zu messende Strecke zum Empfänger zurück. Dessen Reflektor bringt den Impuls in die Empfangskapsel, die wiederum zu schwingen anfängt. Eine Schaltung aus Verstärker und Detektor erkennt das Signal und steuert damit die Zeitmessung. Auf diesem Weg können einige Fehler auftreten, welche die Messung beeinträchtigen oder sogar unmöglich machen können.

- Die Schallgeschwindigkeit muß möglichst genau bekannt sein. Entweder muß sie regelmäßig manuell eingegeben oder vom System gemessen werden. Wenn die Luft über dem Meßaufbau sich bewegt, addieren sich die Geschwindigkeitsvektoren der Luft und des Schalls und verfälschen den Wert leicht. In geschlossenen Räumen spielt dies jedoch keine Rolle.
- Der gemessene Weg entspricht nicht genau dem Abstand der Kapseln in der Ebene, es kommen noch die Strecken von der Kapsel zum Reflektor hinzu. Diese Strecken müssen bei einer Kalibrierung gemessen und bei der Auswertung abgezogen werden.
- Die Verbindungslinie zwischen Sender und Empfänger muß frei von Hindernissen sein, sonst nimmt der Schall einen Umweg und die gemessenen Strecken erscheinen zu lang.
- Mit wachsendem Abstand wird der Schallimpuls schwächer und ist damit schwieriger zu erkennen. Außerdem braucht die Kapsel länger, bis sie ins Schwingen gerät und die für den Detektor erforderliche Signalstärke liefert. Als Folge können Strecken zusätzlich verlängert erscheinen.
- Die Empfindlichkeit des Empfängers stellt immer einen Kompromiß zwischen möglichst großer Reichweite und hoher Störunempfindlichkeit dar.
- Breitbandige Störimpulse können den Detektor im Empfänger fälschlicherweise auslösen und damit auch gültige Messungen verhindern. Viele Alltagsgeräusche sind dazu in der Lage, dazu gehören beispielsweise schon das Klappern eines Schlüsselbundes.

- Jeder Ultraschallimpuls ruft im Raum Echos hervor, die erst nach einer gewissen Zeit abklingen. In dieser Zeit sind keine weiteren Messungen möglich. Der Sender muß dies berücksichtigen und sich dabei eventuell sogar auf die Gegebenheiten des jeweiligen Raumes einstellen.
- Die Streckenmessung kann natürlich nicht präziser sein als die Messung der Laufzeit. Die Schallwellen legen einen Millimeter in etwa 3 Mikrosekunden zurück, die Zeitmessungen müssen mit entsprechender Auflösung erfolgen.

4.5 Berechnung der Koordinaten

Die Koordinaten des Roboters können aus den gemessenen Laufzeiten oder Laufzeitunterschieden berechnet werden. Die im folgenden gezeigten Verfahren gehen von einem Feld der Größe a × b aus, an dessen Rand bis zu 5 Leuchtfeuer stehen und in dem der Roboter R an den Koordinaten (x,y) steht. Der verwendete Lösungsansatz basiert auf dem in [Mahajan] skizzierten Rechenweg.

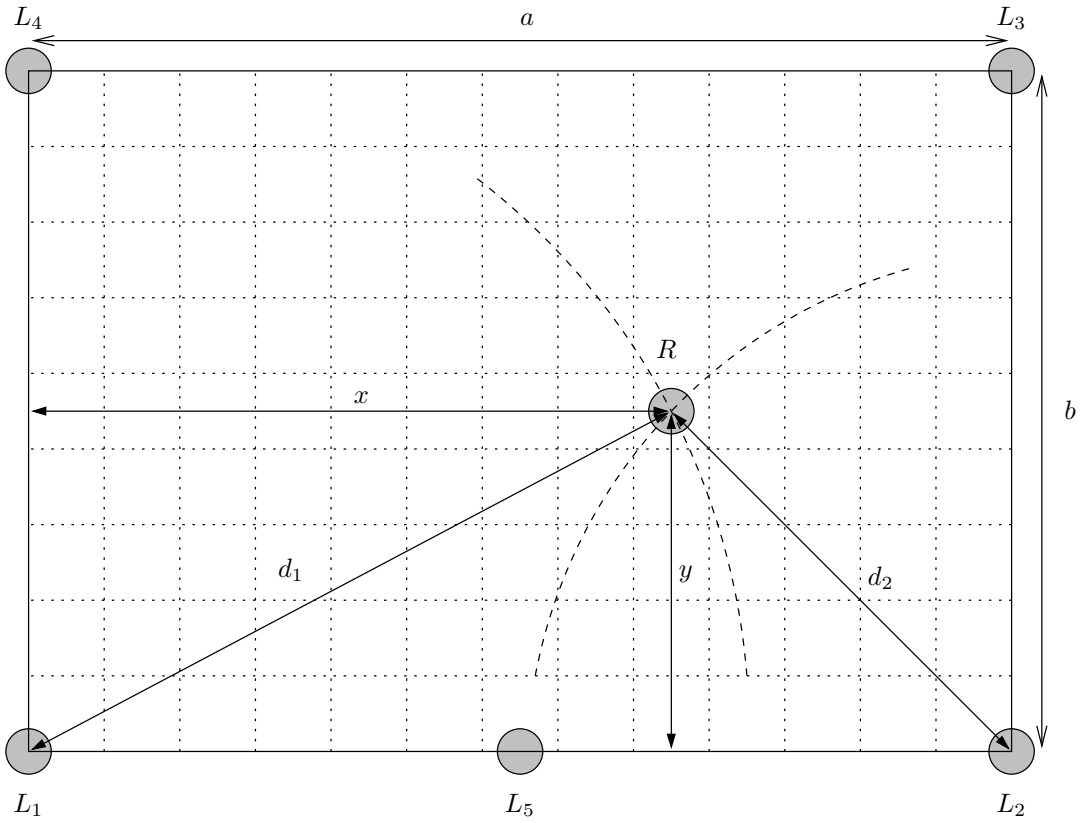


Abbildung 4.9: Schema zur Berechnung der Koordinaten

Die Berechnung der Koordinaten erfolgt zunächst durch Anwendung des Satzes des Pythagoras.

$$d_1^2 = c^2t_1^2 = x^2 + y^2 \tag{4.12}$$
$$d_2^2 = c^2t_2^2 = (a-x)^2 + y^2 \tag{4.13}$$
$$d_3^2 = c^2t_3^2 = (a-x)^2 + (b-y)^2 \tag{4.14}$$
$$d_4^2 = c^2t_4^2 = x^2 + (b-y)^2 \tag{4.15}$$
$$d_5^2 = c^2t_5^2 = \left(\frac{a}{2} - x\right)^2 + y^2 \tag{4.16}$$

Diese Formeln bilden die Grundlage für zwei komplett unterschiedliche Rechenwege.

4.5.1 Berechnung mit Hilfe absoluter Laufzeiten

Die Koordinaten lassen sich aus den Gleichungen (4.13) bis (4.15) direkt ermitteln, wenn die absoluten Laufzeiten t_i und damit die Entfernungen d_i bekannt sind. Dazu werden die genannten Formeln ausmultipliziert und durch Subtraktion von (4.12) alle Vorkommen von x^2 und y^2 eliminiert. In Gleichung (4.14) sind sowohl x als auch y noch enthalten. Durch geeignete Subtraktion der übrigen Formeln entstehen daraus zwei Gleichungen mit jeweils nur noch einer Koordinate.

$$a^2 - 2ax = c^2(t_2^2 - t_1^2) \tag{4.17}$$
$$a^2 - 2ax = c^2(t_3^2 - t_4^2) \tag{4.18}$$
$$b^2 - 2by = c^2(t_3^2 - t_2^2) \tag{4.19}$$
$$b^2 - 2by = c^2(t_4^2 - t_1^2) \tag{4.20}$$

Das Umstellen der Ergebnisse nach x und y liefert die gewünschten Koordinaten, wobei jeweils zwei alternative Formeln zur Berechnung von x und y entstehen.

$$x = \frac{a^2 + c^2(t_1^2 - t_2^2)}{2a} = \frac{a^2 + c^2(t_4^2 - t_3^2)}{2a} \tag{4.21}$$
$$y = \frac{b^2 + c^2(t_1^2 - t_4^2)}{2b} = \frac{b^2 + c^2(t_2^2 - t_3^2)}{2b} \tag{4.22}$$

Mit in den Ecken plazierten Leuchtfeuern ist es nicht möglich, die Schallgeschwindigkeit und die Position des Roboters gleichzeitig zu berechnen. Die Elimination der Koordinaten führt immer dazu, daß sich auch die Schallgeschwindigkeit weghebt. In solchen Fällen gewinnt das fünfte Leuchtfeuer und mit ihm Gleichung (4.16) an Bedeutung. Sie wird ausmultipliziert und (4.12) von ihr subtrahiert.

$$\frac{a^2}{2} - 2ax = 2c^2(t_5^2 - t_1^2)$$

Aus dieser Gleichung ergeben sich zwei Formeln zur Berechnung der Schallgeschwindigkeit, indem sie von (4.17) beziehungsweise (4.18) abgezogen wird.

$$c^2 = \frac{a^2}{2(t_1^2 + t_2^2 - 2t_5^2)} = \frac{a^2}{2(t_4^2 + t_3^2 - 2t_5^2)} \tag{4.23}$$

4.5.2 Berechnung mit Hilfe von Laufzeitunterschieden

Wenn das Ortungssystem sich auf den Einsatz von Ultraschall beschränken soll, können keine absoluten Laufzeiten mehr gemessen werden. Der Empfänger kann nur die Zeitdifferenz zwischen dem Eintreffen zweier Impulse feststellen und muß mit diesen Informationen auskommen. Dies ist aber durchaus möglich, wie die folgende Rechnung zeigt.

- Der Sender besteht aus zwei Leuchtfeuern, die im Abstand Δt je einen kurzen Impuls senden.
- Wenn der Sender zum Zeitpunkt t_0 mit einem Durchlauf beginnt, startet der erste Impuls zum Zeitpunkt t_0 und erreicht den Empfänger bei $t_0 + t_1$.
- Entsprechend erreicht der zweite Impuls den Empfänger bei $t_0 + \Delta t + t_2$.
- Der Empfänger registriert die Zeit, die zwischen dem Eintreffen der beiden Impulse vergeht, also $(t_0 + \Delta t + t_2) - (t_0 + t_1) = \Delta t + t_2 - t_1$.
- Wenn der Empfänger die Konstante Δt kennt, kann er diese subtrahieren und erhält so den Laufzeitunterschied $t_{21} = t_2 - t_1$.

Dieses Verfahren kann problemlos auf alle vorhandenen Leuchtfeuer erweitert werden.

$$t_{21} = t_2 - t_1 \tag{4.24}$$
$$t_{31} = t_3 - t_1 \tag{4.25}$$
$$t_{41} = t_4 - t_1 \tag{4.26}$$
$$t_{51} = t_5 - t_1 \tag{4.27}$$

Damit stellen sich die Formeln zur Koordinatenberechnung von L_1, L_2 und L_5 wie folgt dar.

$$x^2 + y^2 = c^2 t_1^2 \tag{4.28}$$
$$(a-x)^2 + y^2 = c^2 (t_{21} + t_1)^2 \tag{4.29}$$
$$(a-x)^2 + (b-y)^2 = c^2 (t_{31} + t_1)^2 \tag{4.30}$$
$$x^2 + (b-y)^2 = c^2 (t_{41} + t_1)^2 \tag{4.31}$$
$$\left(\frac{a}{2} - x\right)^2 + y^2 = c^2 (t_{51} + t_1)^2 \tag{4.32}$$

Die Gleichungen(4.29) bis (4.32) werden ausmultipliziert und durch Subtraktion von (4.28) alle Vorkommen von x^2 und y^2 eliminiert. (4.30) enthält beide Koordinaten und wird durch Subtraktion der übrigen Gleichungen in die beiden Teile (4.35) und (4.36) mit jeweils nur einer Koordinate aufgeteilt.

$$a^2 - 2ax = c^2 (t_{21}^2 + 2 t_{21} t_1) \tag{4.33}$$
$$b^2 - 2by = c^2 (t_{41}^2 + 2 t_{41} t_1) \tag{4.34}$$
$$a^2 - 2ax = c^2 (t_{31}^2 + 2 t_{31} t_1 - t_{41}^2 - 2 t_{41} t_1) \tag{4.35}$$
$$b^2 - 2by = c^2 (t_{31}^2 + 2 t_{31} t_1 - t_{21}^2 - 2 t_{21} t_1) \tag{4.36}$$
$$\frac{a^2}{4} - ax = c^2 (t_{51}^2 + 2 t_{51} t_1) \tag{4.37}$$

Aus diesen Gleichungen können verschiedene Formeln für die absolute Laufzeit t_1 hergeleitet werden, indem jeweils zwei Gleichungen so kombiniert werden, daß die Koordinaten wegfallen.

$$t_1 = \frac{t_{21}^2 - t_{31}^2 + t_{41}^2}{2(t_{31} - t_{41} - t_{21})} \quad \text{aus (4.33)/(4.35) bzw. (4.34)/(4.36)} \tag{4.38}$$
$$t_1 = \frac{a^2 - 2c^2(t_{21}^2 - 2t_{51}^2)}{4c^2(t_{21} - 2t_{51})} \quad \text{aus (4.33)/(4.37)} \tag{4.39}$$
$$t_1 = \frac{a^2 - 2c^2(t_{31}^2 - t_{41}^2 - 2t_{51}^2)}{4c^2(t_{31} - t_{41} - 2t_{51})} \quad \text{aus (4.35)/(4.37)} \tag{4.40}$$

Mit Hilfe der Gleichungen (4.24) bis (4.27) können dann aus t_1 die sämtlichen absoluten Laufzeiten berechnet und somit die Formeln für die Koordinaten aus dem vorigen Abschnitt benutzt werden.

4.6 Beschreibung verschiedener Szenarien

Aus den vorgestellten Bausteinen kann eine Reihe von Szenarien zusammengestellt werden. Einige davon werden in den folgenden Abschnitten vorgestellt und verglichen. Sie basieren alle auf der in Abbildung 4.9 gezeigten Geometrie und unterscheiden sich in der Anzahl der Leuchtfeuer, ihren Möglichkeiten und der erreichbaren Genauigkeit. Für eine Fehlerabschätzung wurde das Programm `coordsim` benutzt, dessen Bedienung in Anhang B.7 erläutert wird.

4.6.1 Szenario 1: Minimale Lösung mit zwei Leuchtfeuern

Der einfachste denkbare Aufbau besteht nur aus zwei Leuchtfeuern, zum Beispiel L_1 und L_2. Die Schallgeschwindigkeit muß bekannt sein oder vor der eigentlichen Ortung in einer Kalibriermessung bestimmt werden. Die Koordinaten werden mit Hilfe der Formeln (4.21) und (4.12) berechnet.

$$\begin{aligned} x &= \frac{a^2 + c^2(t_1^2 - t_2^2)}{2a} \\ y &= \pm\sqrt{c^2 t_1^2 - x^2} \end{aligned}$$

Der negative Wert für y kann vernachlässigt werden, wenn der Roboter sich nicht unterhalb der X-Achse bewegen kann, wie es das Szenario vorsieht. Falls die Wurzel in Folge von Meßfehlern nicht reell ist, wird stattdessen für y der Wert 0 angenommen.

Die Fehlersimulation in Abbildung 4.10 stellt die zu erwartende Genauigkeit unter Berücksichtigung von Meßfehlern dar. Der Roboter wird dabei auf einige regelmäßig angeordnete Gitterpunkte gesetzt und in jedem seine Position berechnet. Durch Fehler in der Entfernungsmessung ist dies aber nicht exakt möglich, deswegen gibt es um jedem Gitterpunkt einen Unsicherheitsbereich, in dem die berechneten Koordinaten liegen können. Dieser ist als Fläche um den Gitterpunkt dargestellt. Je größer die Fläche ist, desto höher ist auch der Fehler der Ortung an dem betreffenden Punkt. Die Simulation geht von einem Meßfehler von bis zu 3,3 mm plus 0,25 % der gemessenen Strecke aus. Diese Parameter werden auch für alle folgenden Szenarien verwendet, damit die Bilder vergleichbar sind.

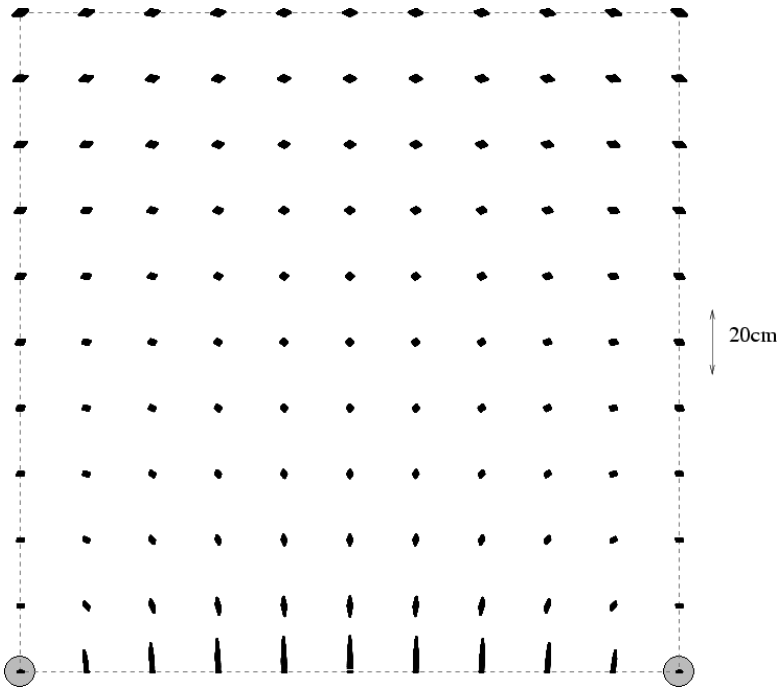


Abbildung 4.10: Fehlersimulation für Szenario 1

Das Ergebnis ist insgesamt zufriedenstellend, nur zwischen den Leuchtfeuern ist der Fehler relativ groß. Dieses Problem läßt sich umgehen, indem die Leuchtfeuer in einer ausreichenden Entfernung vom Feld aufgestellt werden, so daß der Roboter nicht in den problematischen Bereich gelangen kann.

Mit einer Kalibriermessung kann die Schallgeschwindigkeit bestimmt werden. Der Roboter wird dazu genau zwischen die Leuchtfeuer gestellt, so daß alle drei auf einer Linie liegen. Der Empfänger mißt

dann die Laufzeit des Schalls zu beiden Leuchtfeuern und berechnet daraus die Schallgeschwindigkeit.

$$c = \frac{a}{t_1 + t_2} \tag{4.41}$$

In der Praxis muß bei allen Berechnungen noch berücksichtigt werden, daß der Ultraschall durch die Reflektoren einen Weg zurücklegt, der länger als die Verbindungslinie von Sender und Empfänger ist. In Gleichung (4.41) muß daher $a + 2z$ statt a eingesetzt werden, wobei z der zusätzliche Weg bei einer Streckenmessung ist. Entsprechend muß bei allen Zeitmessungen z/c subtrahiert werden. Dies gilt auch für alle anderen Szenarien, die nicht mit Laufzeitunterschieden arbeiten.

4.6.2 Szenario 2: Normale Ortung mit drei Leuchtfeuern

Die problematische Berechnung der Y-Koordinate in Szenario 1 kann entschärft werden, wenn ein zusätzliches Leuchtfeuer hinzugenommen wird. In Frage kommen hierfür L_3 und L_4. Die Berechnung der Koordinaten ist dann wesentlich stabiler möglich, außerdem entfällt die Einschränkung, daß der Roboter sich nicht unterhalb der X-Achse befinden darf. Die Formeln bei Verwendung von L_1, L_2 und L_3 wurden bereits als Gleichungen mit den Nummern (4.21) und (4.22) hergeleitet.

$$x = \frac{a^2 + c^2(t_1^2 - t_2^2)}{2a}$$
$$y = \frac{b^2 + c^2(t_2^2 - t_3^2)}{2b}$$

Der Fehler ist erwartungsgemäß in den meisten Punkten des Feldes relativ klein und ändert sich wenig. Mit wachsendem Abstand von allen drei Leuchtfeuern nimmt er jedoch leicht zu und erreicht in der oberen linken Ecke des Feldes sein Maximum.

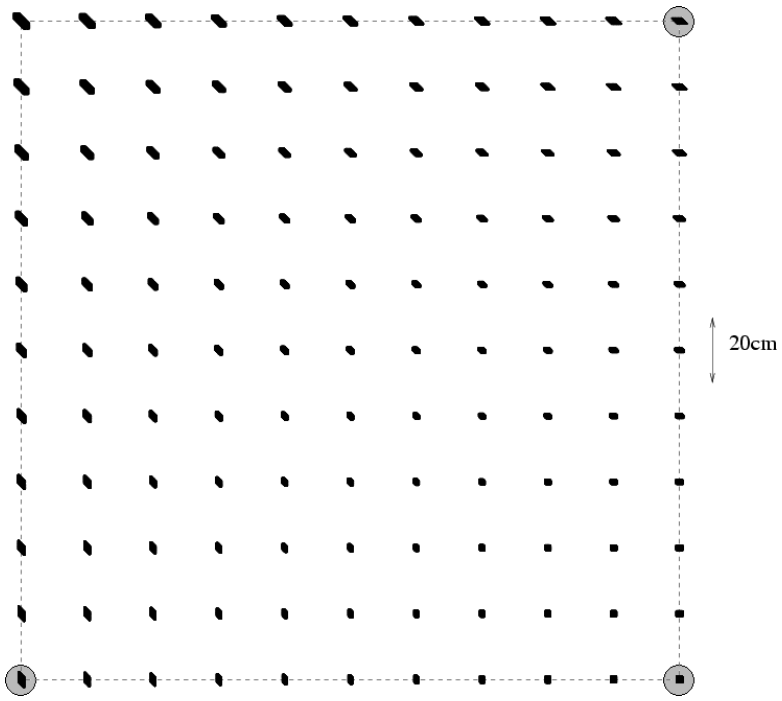


Abbildung 4.11: Fehlersimulation für Szenario 2

Die Genauigkeit läßt sich noch weiter steigern, indem das Leuchtfeuer L_4 hinzugenommen wird. Wenn der Roboter den Bereich mit geringem Fehler verläßt, können die Daten von drei anderen Leuchtfeuern für die Berechnung zugrunde gelegt werden, so daß ein geringerer Fehler zu erwarten ist. Dafür

eignen sich beispielsweise L_3, L_4 und L_1. Die Schallgeschwindigkeit kann mit dem selben Trick wie in Szenario 1 bei einer Kalibriermessung bestimmt werden. Die Verlängerung aller gemessenen Strecken durch die Reflektoren muß auch hier berücksichtigt werden.

4.6.3 Szenario 3: Integrierte Bestimmung der Schallgeschwindigkeit

Ein Nachteil der beiden ersten Szenarien ist, daß die Schallgeschwindigkeit als Konstante in die Berechnung der Koordinaten eingeht. Sie muß einmal fest eingestellt werden und paßt sich danach bei Veränderungen der Umgebung nicht an. Außerdem bedeutet die Kalibriermessung einen zusätzlichen Aufwand. Diese Nachteile lassen sich umgehen, indem ein weiteres Leuchtfeuer in das Szenario integriert wird. Die zusätzliche Koordinatengleichung erlaubt es, eine weitere Unbekannte aus den Meßwerten auszurechnen und damit Schallgeschwindigkeit und Position gleichzeitig zu bestimmen. Für diese Aufgabe ist L_5 als einziges der fünf Leuchtfeuer geeignet.

Die Schallgeschwindigkeit kann über die Formel (4.23) berechnet werden. Mit dieser Grundlage können über die Gleichungen (4.21) und (4.22) die Koordinaten berechnet werden.

$$\begin{aligned} c^2 &= \frac{a^2}{2(t_1^2 + t_2^2 - 2t_5^2)} \\ x &= \frac{a^2 + c^2(t_1^2 - t_2^2)}{2a} \\ y &= \sqrt{c^2 t_1^2 - x^2} \end{aligned}$$

Die Berechnung der Schallgeschwindigkeit funktioniert stabil, solange der Roboter sich nicht sehr weit vom Feld entfernt. Anderenfalls konvergiert der Nenner von (4.23) gegen Null.

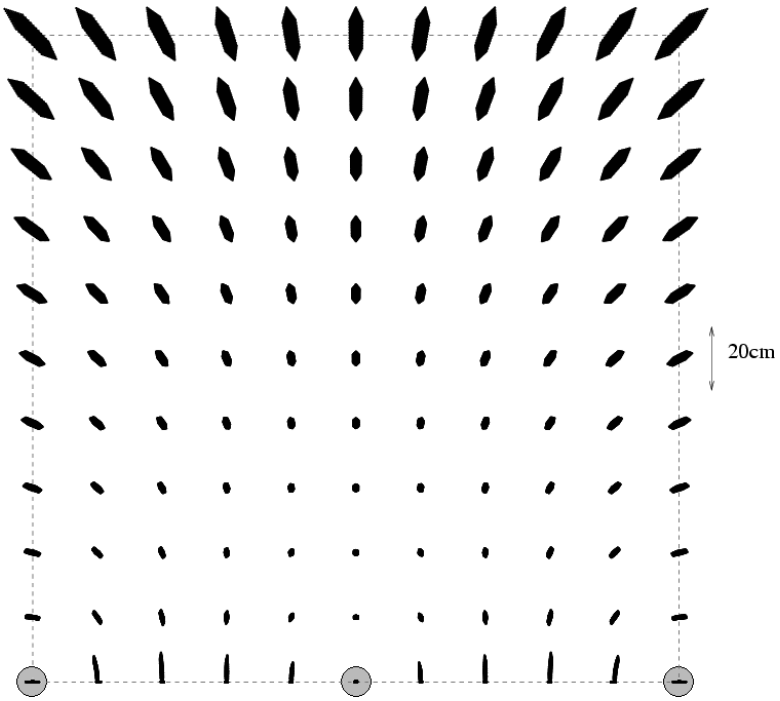


Abbildung 4.12: Fehlersimulation für Szenario 3

Das Ergebnis der Simulation in Abbildung 4.12 zeigt, daß der Fehler in der unteren Hälfte des Feldes akzeptabel klein ist. Betritt der Roboter die obere Hälfte, wächst der Fehler deutlich an. Das

Steuerprogramm kann im oberen Bereich also auf eine bereits vorher ermittelte Schallgeschwindigkeit zurückgreifen und diese nur in der unteren Hälfte aktualisieren.

4.6.4 Szenario 4: Ortung nur mit Ultraschall

Die ersten drei Szenarien sind davon ausgegangen, daß das System absolute Laufzeiten und damit auch die Entfernungen direkt messen kann. Steht das dafür nötige Funksignal nicht zur Verfügung, bleibt nur die Messung der Laufzeitunterschiede. Dies hat aufwendigere Berechnungen zur Folge und erfordert ein zusätzliches Leuchtfeuer. Das einfachste denkbare Szenario benutzt also die drei Leuchtfeuer L_1, L_2 und L_5. Die für die Berechnung nötigen Formeln sind (4.39), (4.24), (4.21) und (4.12).

$$\begin{aligned} t_1 &= \frac{a^2 - 2c^2(t_{21}^2 - 2t_{51}^2)}{4c^2(t_{21} - 2t_{51})} \\ t_2 &= t_{21} + t_1 \\ x &= \frac{a^2 + c^2(t_1^2 - t_2^2)}{2a} \\ y &= \sqrt{c^2 t_1^2 - x^2} \end{aligned}$$

Die Koordinatenberechnung funktioniert zwar, ist aber mit erheblichen Fehlern verbunden, wie Abbildung 4.13 belegt. Dabei ist der für die Simulation angenommene Fehler (3,3 mm absolut und 0.25 % relativ) niedrig angesetzt in der Praxis nur schwer einzuhalten.

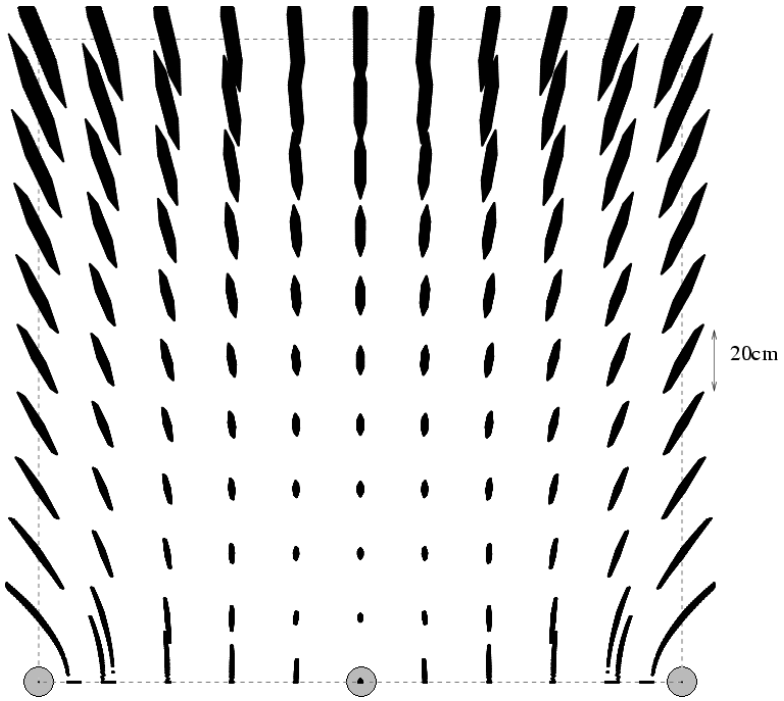


Abbildung 4.13: Fehlersimulation für Szenario 4

Bei der Berechnung von t_1 konvergiert der Nenner des Bruches gegen Null, wenn der Roboter sich L_1 oder L_2 stark nähert. Als Folge werden die Formeln instabil und dürfen bei der Koordinatenberechnung nicht benutzt werden. Der Algorithmus kann stattdessen die Koordinaten des nächstgelegenen Leuchtfeuers liefern.

Szenarien, die mit Laufzeitunterschieden arbeiten, haben einen gemeinsamen großen Vorteil. Alle gemessenen Strecken werden durch die Reflektoren um den selben Betrag verlängert, dieser hebt sich durch die Differenzbildung weg und braucht in den Rechnungen nicht berücksichtigt zu werden.

4.6.5 Szenario 5: Erster Verbesserungsversuch

Bei der Untersuchung von Szenario 4 fällt auf, daß der Fehler in den beiden oberen Ecken am größten ist. Dies legt die Vermutung nahe, daß der Fehler deutlich geringer sein sollte, wenn Leuchtfeuer in allen Ecken plaziert werden. Ihr Abstand ist dann größer und die Abdeckung ist gleichmäßiger. Die nötigen Formeln für diesen Aufbau sind (4.38), (4.24) bis (4.25), (4.21) und (4.22).

$$\begin{aligned} t_1 &= \frac{t_{21}^2 - t_{31}^2 + t_{41}^2}{2(t_{31} - t_{41} - t_{21})} \\ t_2 &= t_{21} + t_1 \\ t_3 &= t_{31} + t_1 \\ x &= \frac{a^2 + c^2(t_1^2 - t_2^2)}{2a} \\ y &= \frac{b^2 + c^2(t_2^2 - t_3^2)}{2b} \end{aligned}$$

Das Problem dieses Ansatzes ist aber, daß der Nenner bei der Berechnung von t_1 auf den Mittelachsen des Feldes ($x = a/2$, $y = b/2$) Null ist. Dadurch kommt es in der Nähe dieser Achsen zu einer Fehlerexplosion, wie es in Abbildung 4.14 dargestellt ist.

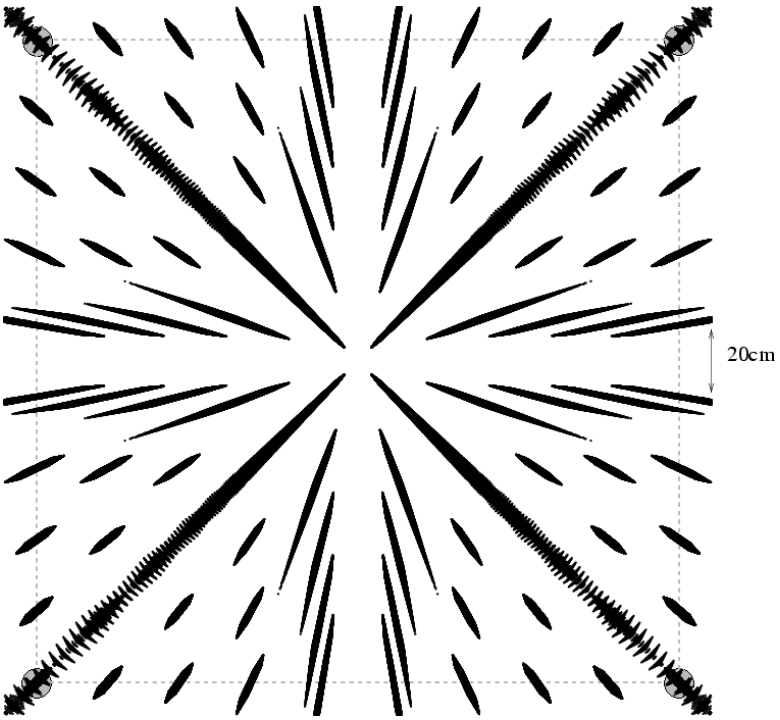


Abbildung 4.14: Fehlersimulation für Szenario 5

Dieser Ansatz ist für die Praxis unbrauchbar, zeigt aber einen wichtigen Aspekt bei der Entwicklung von Formeln für die Koordinatenberechnung. Es ist oft schwierig zu sehen, ob ein Ansatz die gewünschte Genauigkeit erreicht und wo seine Schwächen liegen. Meistens ist eine Fehlersimulation

oder eine mathematische Untersuchung der Formeln nötig, um den Algorithmus beurteilen zu können. Die Anordnung der Leuchtfeuer sagt alleine wenig über den zu erwartenden Fehler aus.

4.6.6 Szenario 6: Zweiter Verbesserungsversuch

Bei einer genaueren Untersuchung von Szenario 4 stellt sich heraus, daß der Fehler in Y-Richtung deutlich größer als in X-Richtung ist. Eine Verbesserung könnte also darin bestehen, eine andere Formel für die Y-Koordinate zu benutzen. Dazu wird Szenario 4 um das Leuchtfeuer L_3 erweitert und Formel (4.22) zur Berechnung der Y-Koordinate benutzt.

$$\begin{aligned} t_1 &= \frac{a^2 - 2c^2(t_{21}^2 - 2t_{51}^2)}{4c^2(t_{21} - 2t_{51})} \\ t_2 &= t_{21} + t_1 \\ t_3 &= t_{31} + t_1 \\ x &= \frac{a^2 + c^2(t_1^2 - t_2^2)}{2a} \\ y &= \frac{b^2 + c^2(t_2^2 - t_3^2)}{2b} \end{aligned}$$

Die Lösung erweist sich als deutliche Verbesserung und erreicht vor allem in der Feldmitte relativ geringe Fehler. Die Fehlerabschätzung zeigt aber auch, daß die Schallgeschwindigkeit bei den Berechnungen bekannt sein muß. Wäre sie eine Variable wie in Szenario 3, wäre der resultierende Fehler so groß, daß die Messungen in der Praxis unbrauchbar sind.

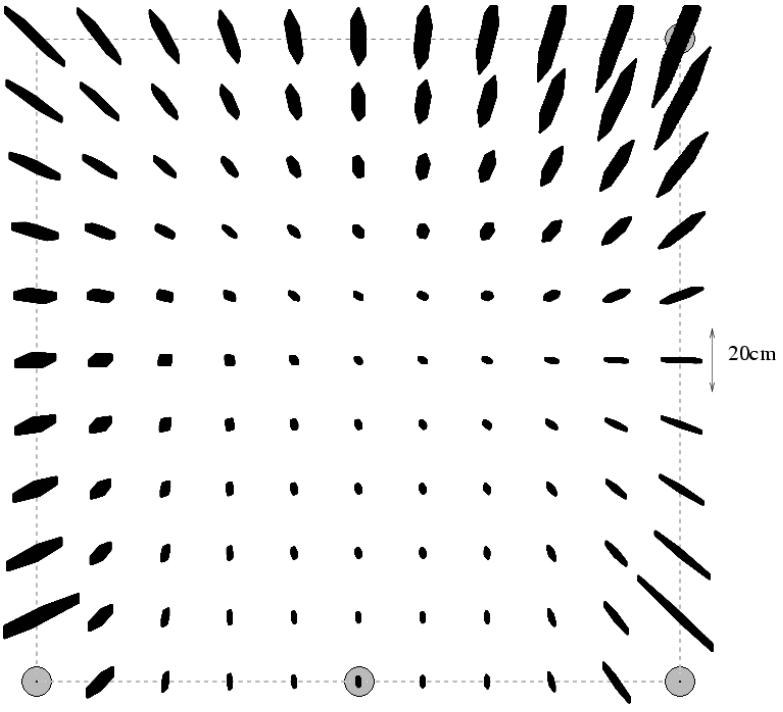


Abbildung 4.15: Fehlersimulation für Szenario 6

4.6.7 Szenario 7: Dritter Verbesserungsversuch

Ein potentieller Nachteil des Szenarios ist noch, daß die an der Berechnung von t_1 beteiligten Leuchtfeuer relativ dicht zusammenstehen. Es könnte erfolgversprechend sein, L_2 durch L_4 zu ersetzen. Dadurch

vergrößert sich die Entfernung der entscheidenden Leuchtfeuer, außerdem kommt bei der Berechnung die bisher unbenutzte Formel (4.40) zum Einsatz.

$$\begin{aligned} t_1 &= \frac{a^2 - 2c^2 \cdot (t_{31}^2 - t_{41}^2 - 2t_{51}^2)}{4c^2(t_{31} - t_{41} - 2t_{51})} \\ t_3 &= t_{31} + t_1 \\ t_4 &= t_{41} + t_1 \\ x &= \frac{a^2 + c^2(t_4^2 - t_3^2)}{2a} \\ y &= \frac{b^2 + c^2(t_1^2 - t_4^2)}{2b} \end{aligned}$$

Die Hoffnungen bestätigen sich leider nicht, auch dieser Rechenweg reagiert sehr instabil auf Meßfehler und liefert in zwei Quadranten keine brauchbaren Ergebnisse. Damit steht fest, daß von den drei möglichen Formeln zur Berechnung von t_1 nur (4.39) sinnvoll eingesetzt werden kann.

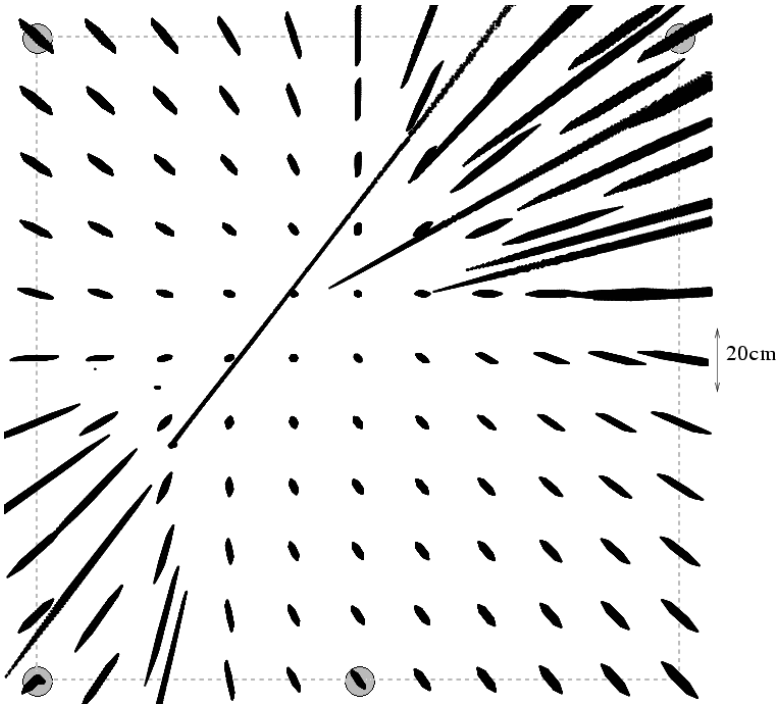


Abbildung 4.16: Fehlersimulation für Szenario 7

4.6.8 Gemischte Szenarien

Die verschiedenen Ansätze stellen natürlich nur Beispiele dar. Es sind auch vollkommen andere Anordnungen der Leuchtfeuer möglich, genauso können in einem Programm mehrere Rechenwege kombiniert werden. Die Software kann anhand der grob geschätzten Position denjenigen Algorithmus auswählen, der die geringsten Fehler verspricht. Dem Benutzer stehen alle Möglichkeiten offen.

4.6.9 Rollen von Sender und Empfänger

Alle bisher beschriebenen Szenarien gehen davon aus, daß die Leuchtfeuer Impulse senden, die der Roboter dann passiv empfängt. Dieses Vorgehen ist zwar naheliegend, daneben ist es aber auch möglich,

die Rollen zu vertauschen und vom Roboter zu den Leuchtfeuern zu senden. Beide Ansätze haben ihre Vorteile und Nachteile. Wenn die Leuchtfeuer das Senden übernehmen, muß dies von einer zentralen Steuerung koordiniert werden. Sie kontrolliert mit hoher Genauigkeit und nach einem festen Zeitschema, wann welcher Impuls gesendet werden soll. Für den Empfänger auf dem Roboter muß nachvollziehbar sein, von welchem Leuchtfeuer ein empfangener Impuls kommt. Diese Information kann zusammen mit dem Funksignal übertragen werden, alternativ kann auch eine besonders lange Sendepause den Start eines Sendezyklus signalisieren. Dieses Verfahren erlaubt es, mehrere Roboter in dem selben Areal einzusetzen und ihre Positionen gleichzeitig zu bestimmen. Dafür ist das System nicht besonders schnell, es sind zur Bestimmung einer Position genausoviele Impulse nötig, wie Leuchtfeuer vorhanden sind.

Alternativ kann auch ein Sender auf den Roboter gesetzt werden. Dieser muß dann nur einen Impuls senden, der von den Leuchtfeuern registriert wird. Die Steuerung des Leuchtfeuersystems sammelt alle Meßdaten, berechnet die Koordinaten und sendet diese zum Beispiel per Infrarot an den Roboter. Eine Messung ist also relativ schnell, dafür müssen die Daten aufwendig zum Roboter übertragen werden und es darf nur ein Roboter zur Zeit das System benutzen. Die Entscheidung, wem die Rollen von Sender und Empfänger zugetragen werden, hat aber keinen Einfluß auf die Formeln zur Koordinatenberechnung. Sie gelten unabhängig davon, solange die Zeiten wie gefordert gemessen oder umgerechnet werden können.

Teil II

Realisierung

Kapitel 5

Ein Ortungssystem für Mindstorms

In den vorigen Kapiteln die wichtigen Grundlagen behandelt, die bei der Konstruktion eines Ortungssystems für Lego Mindstorms eine Rolle spielen. Auf dieser Basis ist ein Ortungssystem entstanden, das in diesem und den folgenden Kapiteln beschrieben werden soll. Vor den technischen Details der Hardware und Software sollen aber zunächst die Anforderungen skizziert werden, die an ein solches System gestellt werden müssen.

5.1 Anforderungen an das Ortungssystem

Es gibt eine Reihe von Kriterien, an denen ein Ortungssystem sich in der Praxis messen muß. Eine wichtige Rolle spielen dabei die Begriffe Reichweite und Genauigkeit, daneben sind noch ganz andere technische Eigenschaften und die Handhabung entscheidend, damit das System mit Erfolg eingesetzt werden kann. Diese Kriterien sollen im Folgenden vorgestellt werden. Sie waren die Grundlage für den Entwicklungsprozeß und haben alle Designentscheidungen maßgeblich beeinflußt. An ihnen wird auch das fertige Produkt gemessen.

Meßverfahren und Variationsmöglichkeiten

- Die Messung von Entfernungen erfolgt mit Ultraschallimpulsen, dazu sollte ein weiteres Synchronisationssignal benutzt werden, wie es in Kapitel 4 vorgeschlagen wurde. Nur mit dieser Kombination sind Ortsbestimmungen mit hoher Genauigkeit möglich.
- Die Reichtweite und Abdeckung des Synchronisationssignals darf nicht schlechter als beim Ultraschall sein, damit die Gesamtleitung des Systems nicht beeinträchtigt wird.
- Der Empfänger sollte sowohl die absoluten Laufzeiten als auch die Laufzeitunterschiede zwischen zwei aufeinanderfolgenden Impulsen messen. Will ein Benutzer die Ortung nur auf Basis der Laufzeitunterschiede durchführen, kann er die übrigen Meßdaten ignorieren. Das System arbeitet dann so, als würde es ausschließlich auf Ultraschallimpulsen basieren.
- Es sind zwei verschiedene Varianten denkbar, der Roboter kann entweder mit einem Sender oder einem Empfänger ausgestattet werden, während die Leuchtfeuer die jeweils andere Aufgabe übernehmen. Beide Ansätze unterscheiden sich in ihren Stärken und Schwächen, das System sollte also beides ermöglichen.

Aufbau und Steuerung der Leuchtfeuer

- Der Benutzer muß die Möglichkeit haben, viele verschiedene Szenarien aufzubauen, die sich in der Anzahl und der Plazierung der Leuchtfeuer unterscheiden.

- Eine zentrale Steuerung wird für die Koordination der Leuchtfeuer gebraucht. Sie sorgt entweder für die Einhaltung eines globalen Sendezeitplanes oder sammelt bei empfangenden Leuchtfeuern die gemessenen Daten ein und leitet sie an den Roboter weiter.
- Steuerung und Leuchtfeuer können über Kabel verbunden werden, was die Kommunikation einfach macht und das Nutzen einer gemeinsamen Stromversorgung erlaubt. Die Verkabelung sollte möglichst einfach sein, eine Bustopologie ist vollkommen ausreichend.

Identifizierung der Leuchtfeuer

- Jedem Leuchtfeuer wird eine eindeutige Nummer zugewiesen, die es identifizierbar macht. Die Nummer ist für die Koordinatenberechnung nötig, um jeder gemessenen Entfernung ein Leuchtfeuer und damit eine Position zuordnen zu können.
- Die Numerierung sollte dynamisch erfolgen, zum Beispiel basierend auf der Verkabelung und der Anzahl der angeschlossenen Leuchtfeuer. Dadurch sind die einzelnen Leuchtfeuer beliebig austauschbar und müssen nicht individuell konfiguriert werden. Die Numerierung muß deterministisch und für den Benutzer nachvollziehbar sein, damit dieser weiß, welches Leuchtfeuer wo aufgestellt wird.
- Jeder Empfänger muß herausfinden können, welches Leuchtfeuer an einer Messung beteiligt war. Das Synchronisationssignal kann benutzt werden, um die Nummer des Leuchtfeuers zu übertragen. Eine Alternative ist, die Unterscheidung auf Basis der Zeit durchzuführen. Dazu läßt das System alle Leuchtfeuer nacheinander feuern und legt am Ende der Runde eine besonders lange Pause ein, bevor es wieder von vorne anfängt. Der Empfänger erkennt anhand der Pause den Anfang einer Runde und zählt danach die Impulse mit. Das System sollte beide Verfahren unterstützen.
- Die Nummern können auch für die Adressierung in der Kommunikation zwischen Steuerung und Leuchtfeuern verwendet werden.

Zeitschema beim Senden

- Beim Einsatz sendender Leuchtfeuer ist die Steuerung für die Einhaltung eines vordefinierten Zeitschemas zuständig, das festlegt, wann welches Leuchtfeuer sendet. Das Zeitschema ist durch drei Parameter charakterisiert, die Anzahl der Leuchtfeuer, die Zeit zwischen zwei regulären Impulsen und die Länge der zusätzlichen Pause nach einer Runde.
- Der Benutzer muß die Zeiten einfach ändern können, damit sie an die jeweiligen Gegebenheiten angepaßt werden können. So kann beispielsweise das Echo in einem Raum so lange nachklingen, daß Impulse nur mit relativ großem Abstand aufeinanderfolgen dürfen.

Anschluß an den RCX

- Das System muß über einen Sender und einen Empfänger verfügen, die an den RCX angeschlossen werden können. Der Empfänger leitet seine Meßdaten über das Sensorinterface an den RCX, der Sender wird über einen Motorausgang gesteuert und kann auf Kommando Impulse senden. Weder der Sensoranschluß noch der Motorausgang sind für die Übermittlung komplexer Informationen konzipiert, hier fehlt also eine geeignete Lösung.
- Beide Peripheriebausteine müssen voll kompatibel zu den Interfaces des RCX sein und sich außerdem wie normale Legosteine an einen Roboter anbauen lassen. Sie sollten möglichst klein und leicht sein und mit einem Minimum an Strom auskommen, damit sie aus dem RCX gespeist werden können und nicht auf eine eigene Energieversorgung angewiesen sind.
- Das gesamte Ortungssystem darf die normalen Funktionen eines Mindstorms-Roboters möglichst nicht einschränken. Das betrifft unter anderem die Infrarot-Kommunikation, die verfügbare Rechenleistung und den Speicherplatz für Programmcode. Außerdem muß eine Ortung auch während der Fahrt möglich sein.

Robustheit, Reichweite und Genauigkeit

- Das System darf durch keine Art von Fehlbedienung beschädigt werden können. Diese Robustheit ist insbesondere für den Einsatz im Übungsbetrieb wichtig.
- Die Komponenten sind als eingebettete Systeme ausgelegt und müssen daher fehlertolerant sein. Parameter wie die Spannungspegel des RCX können sich laufend ändern, außerdem unterliegen sämtliche Messungen und Datenübertragungen vielen Störfaktoren.
- Die Ultraschallsender brauchen eine möglichst hohe Signalstärke, genauso müssen die Detektoren in den Empfängern empfindlich für das Signal und möglich unempfindlich für Störungen sein. Diese Faktoren sind Voraussetzung für eine große Reichweite.
- Die einzelnen Messungen müssen mit einer hohen Genauigkeit erfolgen. Entfernungen lassen sich nur auf einen Millimeter genau bestimmen, wenn die Auflösung der Laufzeitmessung im Mikrosekundenbereich liegt.
- Eine Positionsbestimmung sollte möglichst schnell erfolgen, für ein genaues Arbeiten sind mehrere Positionen pro Sekunde nötig.

Einfache Benutzung

- Das gesamte Ortungssystem muß möglichst einfach sein. Dies betrifft zum einen den internen Aufbau, die Schaltungen sollten einfach zu verstehen, zu bauen und bei Bedarf zu warten sein. Die Bedienung sollte von einem Benutzer so schnell durchschaut werden können, so daß er mit einem Minimum an Arbeit das Ortungssystem installieren und zu brauchbaren Ergebnissen kommen kann.
- Einstellmöglichkeiten sind nur an sehr wenigen Stellen nötig. Die meisten Komponenten sind direkt, also ohne Kalibrierung oder das Einstellen irgendwelcher Parameter benutzbar.
- Einfache Anzeigen wie Leuchtdioden können signalisieren, ob das Ortungssystem funktioniert, und gegebenenfalls einen Hinweis auf Fehlerursachen liefern.
- Sender und Empfänger müssen richtungsunabhängig arbeiten können, die Ausrichtung darf keine Rolle spielen. Dies erfordert den Einsatz kegelförmiger Schallreflektoren.
- Die Programmierung auf dem RCX soll über ein möglichst einfaches Interface erfolgen. Unerfahrene Programmierer können im Idealfall mit wenigen Programmzeilen und wenig Hintergrundwissen über das System schnell brauchbare Ergebnisse zu erhalten.

Die genannten Anforderungen geben bereits viele Designentscheidungen vor. Beispielsweise muß das zusätzlich zum Ultraschall eingesetzte Synchronisationssignal sich einfach erzeugen und detektieren lassen, eine hohe Ausbreitunggeschwindigkeit haben und einfache Informationen übertragen können. Unter diesen Bedingungen kommen zunächst Funk- und Infrarotsignale in Betracht. Beide lassen sich mit handelsüblichen Bauteilen oder Modulen erzeugen und detektieren, durch Modulation können Daten übertragen werden. Wesentliche Unterschiede gibt es bei den Ausbreitungseigenschaften. Es ist sehr schwierig, einen größeren Bereich mit synchron geschalteten IR-Sendedioden auszuleuchten. Dafür sind viele Dioden nötig, was aufwendige Installationen und einen hohen Stromverbrauch bedeutet. Außerdem benutzen die gängigen Module Trägerfrequenzen, die dicht genug an der Trägerfrequenz der RCX-Kommunikation liegen, so daß es zu Störungen kommt. Funksignale sind in der Beziehung unkompliziert, ein größerer Bereich kann mit einem Sender lückenlos abgedeckt werden, ohne viel Strom zu verbrauchen. Die Entscheidung ist daher auf fertige Funkmodule mit digitalen Ein- und Ausgängen für das 433 MHz-Band gefallen.

Viele der Komponenten sind eingebettete und verteilte Echtzeitsysteme, sie müssen mit einer sehr hohen zeitlichen Präzision arbeiten und dabei komplizierte Aufgaben wie das Übermitteln codierter Informationen ausführen. Dennoch sollen die Schaltungen einfach bleiben und wenig Strom verbrauchen, hier führt also kein Weg am Einsatz von Mikrocontrollern vorbei.

5.2 Beschreibung des realisierten Systems

Das fertige Ortungssystem besteht aus einer Anzahl von Modulen, mit denen verschiedene Szenarien aufgebaut werden können. Das Baukastenprinzip gibt eine gewisse Variabilität, wie sie im vorigen Abschnitt gefordert wurde. Es sind die folgenden Module vorhanden:

- Zwei Empfänger zum Anschluß an den RCX
- Ein Sender für den RCX
- Vier Leuchtfeuer
- Eine Steuereinheit für die Leuchtfeuer
- Verkabelung, Netzteil, Vibrationsdämpfer und weiteres Zubehör

Es gibt grundsätzlich zwei verschiedene Arten, wie aus diesen Bauteilen ein Ortungssystem aufgebaut werden kann. Sie unterscheiden sich darin, ob auf dem mobilen Roboter ein Sender oder ein Empfänger installiert wird.

5.2.1 Aufbau mit mobilem Empfänger

Es bringt die meisten Vorteile, wenn ein Empfänger auf dem Roboter befestigt wird. Mehrere fest installierte Leuchtfeuer senden regelmäßig Impulse aus, die der Roboter empfängt. Aus den Meßdaten, die der Sensor direkt an den RCX überträgt, kann die Steuersoftware die Koordinaten berechnen. Der Roboter ist nur passiver Beobachter, daher können beliebig viele gleichzeitig eingesetzt werden. Ein typischer Aufbau nach diesem Prinzip ist in Abbildung 5.1 dargestellt.

Abbildung 5.1: Minimalaufbau des Ortungssystems

Die Steuereinheit links im Bild übernimmt die Stromversorgung des fest installierten Teils und sorgt dafür, daß die Leuchtfeuer abwechselnd und in regelmäßigen Abständen Impulse aussenden. Sie ist über Kabel mit den Leuchtfeuern verbunden, die Kette wird hinter dem letzten Leuchtfeuer über ein spezielles Kabelstück (den Terminator) beendet. Die Reihenfolge der Verkabelung gibt gleichzeitig die Numerierung der Stationen vor. Das letzte Leuchtfeuer vor dem Terminator bekommt die 0 zugeordnet, zur Steuereinheit hin werden die Nummern in Einerschritten hochgezählt.

Das System sendet in programmierbaren Abständen Funk- und Ultraschallsignale in Richtung des Roboters. Die Funksignale dienen dabei nicht nur zur Synchronisation, sie übermitteln auch die Nummer des gerade aktiven Leuchtfeuers. Das Zeitschema kann an der Steuereinheit ausgewählt werden, wobei mit zwei Knöpfen zwischen einer Reihe vordefinierter Muster umgeschaltet werden kann. Ein LC-Display zeigt das Muster und weitere Statusinformationen über das System an.

Der Empfänger ist auf dem frei beweglichen Roboter montiert. Er registriert die Funk- und Ultraschallsignale und übermittelt seine Meßdaten an den RCX. Ein Datensatz umfaßt mehrere Parameter.

- Absolute Laufzeit des Ultraschalls vom Leuchtfeuer zum Empfänger
- Zeitdifferenz zwischen dem Eintreffen des aktuellen und des vorigen Impulses
- Identifikation des sendenden Leuchtfeuers
- Zeit von dem Eintreffen des Impulses bis zum Abklingen des Echos
- Flags, die angeben, ob die oben genannten Werte gültig sind

Wenn der RCX den Datensatz komplett empfangen hat, ruft brickOS eine Callback-Funktion des Anwendungsprogramms auf. Diese übernimmt dann die Auswertung der Daten und die Berechnung der Koordinaten. Der Aufbau kann über die Anzahl und die Anordnung der Leuchtfeuer variiert werden, bis zu 4 Stück können von der Steuereinheit verwaltet werden. Mit den beiden vorhandenen Empfängern können zwei Roboter gleichzeitig eingesetzt werden, mit zusätzlicher Hardware wären noch deutlich mehr möglich.

5.2.2 Aufbau mit mobilem Sender

Das im vorigen Abschnitt beschriebene Verfahren hat einen wesentlichen Nachteil. Es sind immer mehrere Impulse nötig, um eine Position zu bestimmen, eine Ortung dauert also relativ lange. Außerdem darf der Roboter sich währenddessen nur möglichst wenig bewegen, damit alle zusammengehörenden Impulse von der selben Position aus gemessen werden. Eine naheliegende Alternative ist, den Sender auf den Roboter zu verlagern und die Leuchtfeuer zu Empfängern zu machen.

Abbildung 5.2: Aufbau des Ortungssystems mit mobilem Sender

Dann reicht ein einziger Impuls für eine Ortsbestimmung aus. Dafür müssen die Leuchtfeuer ihre Meßdaten an den Roboter senden, was die Komplexität des Systems deutlich erhöht und dafür sorgt, daß nur ein Roboter zur Zeit geortet werden kann. Durch die zu erwartenden Schwierigkeiten und die beschränkten Einsatzmöglichkeiten ist diese Lösung nicht der Schwerpunkt der Arbeit, sie kann aber

mit den vorhandenen Teilen aufgebaut werden, wie Abbildung 5.2 zeigt. Der Roboter löst bei Bedarf über einen Motorausgang einen Impuls aus. Der Sender strahlt dann ein Ultraschall- und ein Funksignal aus, die von beiden Empfängern registriert werden. Die Meßdaten der beiden Leuchtfeuer werden an den feststehenden RCX übermittelt, der als Steuereinheit fungiert. Sobald er die Daten der beiden Leuchtfeuer bekommen hat, überträgt der RCX die Rohdaten oder einen aufbereiteten Datensatz über seinen Transceiver an den Roboter. Damit ist der Zyklus beendet und kann von neuem starten. Bis zu drei Leuchtfeuer können dafür an einen RCX angeschlossen werden, wenn genügend viele Empfänger vorhanden sind.

Kapitel 6

Verwendete Baugruppen

Dieses und das folgende Kapitel beschreiben die Funktionsweise des Ortungssystems im Detail. Im Aufbau gibt es eine Reihe von Baugruppen, die mehrfach benutzt werden. Ein Beispiel dafür ist der Ultraschallsender, der sowohl in den Leuchtfeuern als auch im mobilen Sender enthalten ist. Um Redundanzen beim Erklären des Systems zu vermeiden, werden in diesem Kapitel zunächst alle Baugruppen vorgestellt. Wie diese in den verschiedenen Modulen zusammenarbeiten, folgt dann in Kapitel 7.

6.1 Einsatz von Mikrocontrollern

In allen Schaltungen übernimmt ein Mikrocontroller aus der PIC-Serie von Microchip einen Großteil der Steueraufgaben. Die kleineren Module stützen sich auf den 12F675 mit 8 Pins, die Steuereinheit braucht für die Anbindung des LC-Displays mehr Pins und verwendet daher den größeren 16F628. Der grundsätzliche Aufbau ist immer ähnlich und entspricht dem in Abbildung 6.1 gezeigten Schema.

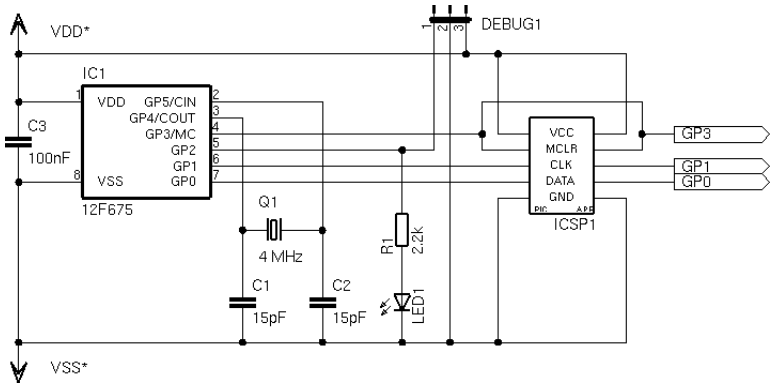


Abbildung 6.1: Beschaltung eines PICs in den meisten Modulen

Der Mikrocontroller wird mit einer stabilisierten Spannung von 5 V versorgt. Als Taktquelle kommt abhängig von der nötigen Präzision entweder ein externer Quarz von 4 MHz oder der interne Oszillator zum Einsatz. Der mobile Sender und die Leuchtfeuer erlauben Schwankungen der Taktfrequenz von einigen Prozent, der Empfänger und die Steuereinheit hingegen sind auf ein genaues Timing angewiesen und brauchen daher einen Quarz. Zwischen einem und drei Pins des Mikrocontrollers werden für Statusausgaben genutzt. Sie sind mit Leuchtdioden und der Debug-Schnittstelle verbunden, bei einigen Schaltungen übernimmt ein Pin zwei dieser Funktionen.

Alle für das Programmieren benötigten Leitungen werden über eine ICSP-Schnittstelle geführt, wie sie in Anhang A.7 beschrieben ist. Auf diese Weise kann der PIC jederzeit umprogrammiert werden, und die entsprechenden Pins bleiben trotzdem nutzbar. MCLR wird in keiner Schaltung zum Auslösen eines Resets gebraucht und ist daher immer als Eingang konfiguriert. In den meisten Schaltungen steht damit ein Großteil der Pins für die Anbindung der Peripherie zur Verfügung.

6.2 Ultraschallsender

Die beiden wichtigsten Baugruppen im Ortungssystem sind naturgemäß Sender und Empfänger für die Ultraschallimpulse. Sie machen eine Abstandsmessung erst möglich und sind damit entscheidend für die gesamte Funktionalität. Beide basieren auf keramischen Schallwandlern mit einer Resonanzfrequenz von 40 kHz, dazu kommt die Elektronik für die Ansteuerung. Die Aufgabe des Sender ist dabei, möglichst starke Impulse zu erzeugen, die auch ein weit entfernter Empfänger noch registrieren kann.

Die Schallerzeugung beginnt in einem Mikrocontroller. Dieser kann die nötigen Impulse generieren, indem er jeweils für eine kurzen Moment ein symmetrisches Rechtecksignal von 40 kHz an einem seiner Pins ausgibt. Verbindet man diesen Ausgang mit dem Schallwandler, hat man bereits einen fertigen Ultraschallsender, dessen Reichweite allerdings sehr gering ist. Die Messungen in Kapitel 4 belegen, daß die Signalstärke und damit die Reichweite proportional zur Amplitude der erzeugenden Spannung ist. Die Sendekapsel kann mit Spannungen bis 20 V_{RMS} betrieben werden, was einem Rechtecksignal mit einer Amplitude von 28,3 V entspricht. Die 5 V vom Ausgang des Mikrocontrollers nutzen dieses Potential nur zu einem Bruchteil aus. Es ist daher nötig, die Spannung in einer nachgeschalteten Treiberstufe deutlich zu erhöhen.

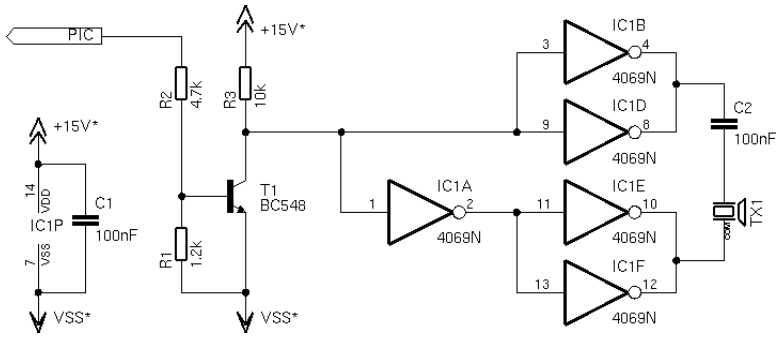


Abbildung 6.2: Treiberstufe des Ultraschallsenders

Abbildung 6.2 zeigt eine solche Treiberstufe, wie sie in vielen Ultraschallsendern enthalten ist. Sie besteht im Wesentlichen aus einem CMOS-IC vom Typ 4069, der mit einer Spannung von bis zu 15 V versorgt wird. Die vom Mikrocontroller erzeugten Impulse werden durch den aus T_1 bestehenden Inverter auf die nötigen Pegel für die Ansteuerung des ICs angehoben. Das Signal teilt sich dann auf zwei Pfade auf, es wird einmal direkt und einmal invertiert auf je eine Ausgangsstufe aus zwei Invertern gebracht. Beide Ausgangsstufen steuern die Sendekapsel an und arbeiten dabei im Gegentakt, auf diese Weise wird die Spannung am Ausgang nahezu verdoppelt und erreicht rund 28 V. Der Kondensator C_2 verhindert, daß im Ruhezustand ständig Strom durch die Kapsel fließt. Sein Wechselstromwiderstand ist hingegen niedrig genug, daß er den Sendevorgang nicht behindert.

Bei der Dimensionierung müssen nur die Größen der Widerstände in der Umgebung von T_1 bestimmt werden. Sie stellen einen Kompromiß zwischen schnellen Schaltzeiten und niedrigem Stromverbrauch dar. Der von R_1 und R_2 gebildete Spannungsteiler muß die Basis des Transistors auf maximal 1 V bringen können, um T_1 zu öffnen. Höhere Spannungen sind nicht wünschenswert, damit T_1 nicht zu

sehr in die Sättigung kommt. Der Gesamtwiderstand des Teilers darf 10 kΩ nicht übersteigen, damit die Schaltgeschwindigkeit nicht beeinträchtigt wird, die gleiche Grenze gilt auch für R_3.

Zehn Perioden des Eingangssignals reichen aus, um einen kräftigen Schallimpuls zu erzeugen. Die Gegentaktsteuerung erhöht die Signalstärke und damit die Reichweite effektiv um 75 Prozent, dadurch erreicht die Treiberstufe eine sehr hohe Ausbeute bei niedrigem Energiebedarf. Sie arbeitet mit einer Versorgungsspannung ab 3 V, ihre maximale Leistung erreicht sie jedoch erst bei 15 V. Der Stromverbrauch während des Sendens liegt dann bei 16 mA, im Ruhezustand, wenn der Eingang auf Masse gehalten wird, fließt praktisch kein Strom. Bei 10 Impulsen pro Sekunde mit einer Dauer von jeweils 10 Perioden resultiert daraus ein mittlerer Stromverbrauch von nur 40 µA. Für den praktischen Einsatz muß noch die Frage geklärt werden, wie die für den Betrieb nötige Spannung von 15 V erzeugt werden kann. Bei den Leuchtfeuern kann einfach ein passendes Netzteil verwendet werden, der an den RCX angeschlossene Sender muß auf einige Tricks zur Erhöhung der vom RCX kommenden Spannung zurückgreifen. Die Details beschreibt Abschnitt 6.6.

6.3 Ultraschalldetektor

Der Ultraschallempfänger hat die Aufgabe, die vom Sender ausgestrahlten Signale zu erkennen. Ein keramischer Schallwandler gerät beim Eintreffen eines Impulses in Schwingung und erzeugt eine Spannung, die durch einen Verstärker geleitet und schließlich einem Detektor zugeführt wird. Der Verstärker ist relativ schwierig zu bauen, einerseits muß er eine sehr große Verstärkung erreichen, andererseits darf er kaum eigenes Rauschen produzieren und muß eingefangene Störungen möglichst gut unterdrücken. Viele kommerzielle Lösungen setzen auf Spezial-ICs, die in der Praxis aber kaum zu beschaffen sind und meistens auch einiges an Strom verbrauchen. Für diese Arbeit wurde daher ein 3-stufiger Transistorverstärker konstruiert, der ausschließlich aus einfachen Standardbauteilen besteht und mit wenig Strom auskommt. Sein Schaltplan ist in Abbildung 6.3 dargestellt.

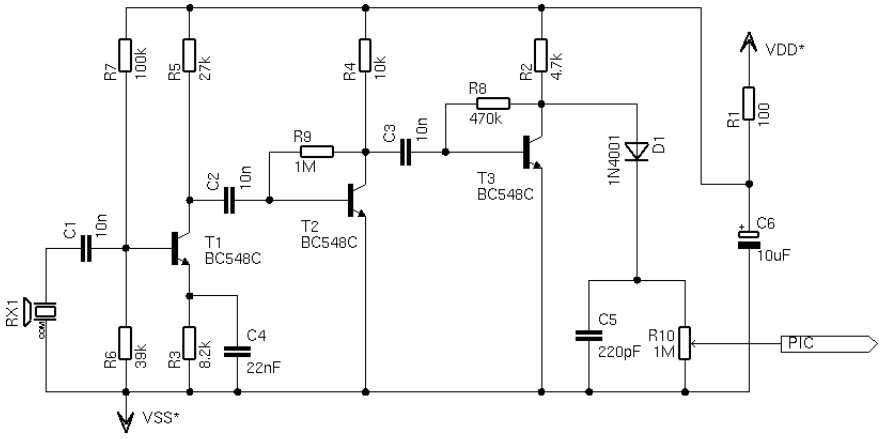


Abbildung 6.3: Empfänger, Verstärker und Detektor für Ultraschall

Die erste Stufe um T_1 ist als Emitterschaltung aufgebaut und bindet die Kapsel an, sie verstärkt das Signal etwa um den Faktor 50. Die zweite und dritte Stufe bestehen aus den Transistoren T_2 und T_3 in Emitterschaltung mit Spannungsgegenkoppelung. Ihre Eigenschaften sind nicht so gut wie die der ersten Stufe, dafür kommen sie mit der Hälfte an Bauteilen aus und erreichen jeweils eine Verstärkung um den Faktor 70. Die Gesamtverstärkung liegt damit bei ungefähr 250.000.

Das Signal wird dann an den aus D_1, C_5 und R_{10} bestehenden Detektor weitergeleitet. Die Diode läßt die positiven Halbwellen durch und lädt damit den Kondensator auf. Dieser entleert sich gleichzeitig

durch R_{10}, so daß sich ein Gleichgewicht einstellt, das von der empfangenen Signalstärke abhängt. Über den Mittelabgriff des Potis wird ein fester Teil der Detektorspannung an einen Eingang des Mikrocontrollers weitergeleitet, der als hochohmiger Komparator konfiguriert sein muß. Übersteigt die Spannung einen fest programmierten Schwellwert, geht der PIC von einem empfangenen Signal aus und reagiert entsprechend. Die tatsächliche Schwelle hängt sowohl von dem Mikrocontroller als auch von der Einstellung des Potis ab und läßt sich damit von innen und außen beeinflussen.

Die Dimensionierung des Empfängers zielt darauf ab, bei niedrigem Stromverbrauch eine möglichst hohe Verstärkung zu erzielen. Dabei wird auch eine drastische Übersteuerung in Kauf genommen, um möglichst steile Flanken am Detektor zu erreichen. Die erste Stufe um T_1 ist auf eine Spannung von 1,4 V an der Basis, einen Kollektorstrom von 0,1 mA und eine Ruhespannung am Kollektor von 2,5 V ausgelegt. Die zweite und dritte Stufe arbeiten mit Spannungsgegenkoppelung ohne Emitterwiderstand, um Bauteile zu sparen. R_2 und C_5 sind so gewählt, daß die Spannung über dem Kondensator beim Eintreffen eines Impulses möglichst schnell ansteigt, wobei das Poti R_{10} auch für die Abklingzeitentscheidend ist. Die übrigen Kondensatoren in der Schaltung sind so ausgelegt, daß niederfrequente Störungen wie das Netzbrummen möglichst stark unterdrückt werden. Aufgrund der hohen Empfindlichkeit muß der Verstärker vor Schwankungen der Versorgungsspannung geschützt werden, derartige Störungen filtert der aus R_1 und C_6 bestehende Tiefpaß heraus. Beim Aufbau der Schaltung auf einer Platine sollte beachtet werden, daß der Verstärker möglichst kompakt, mit kurzen Leiterbahnen und isoliert von seiner Umgebung plaziert wird. Externe Störsender können so kaum in den Aufbau einstrahlen.

Der Stromverbrauch des Verstärkers beträgt 1 mA und seine Verstärkung ist ausreichend, um relativ große Distanzen überbrücken zu können. Beim Einsatz von Reflektoren in Sender und Empfänger, durch die ein erheblicher Teil der Schallenergie verloren geht, läßt sich eine Strecke von 10 Metern noch zuverlässig überbrücken. Aber auch in Szenarien, in denen nur kürzere Distanzen auftreten können, hat die große Verstärkung Vorteile. Sie sorgt dafür, daß der Detektor sehr schnell anspricht, und erlaubt damit ein Erkennen des Impulses ohne nennenswerte Verzögerungen.

Das Ortungssystem von [Restena] verfolgt einen ähnlichen Ansatz wie diese Arbeit, theoretisch hätte daher auch die Möglichkeit bestanden, deren Verstärker zu übernehmen. Er basiert auf drei hintereinandergeschalteten OPs, enthält aber einige grundlegende Fehler. Das Signal wird dadurch maximal um den Faktor 15.000 verstärkt, außerdem ist das Signal-Rausch-Verhältnis sehr schlecht. Ein Nachbau auf einer Steckplatine hatte im Test so schlechte Eigenschaften, daß im Rauschen gar kein Signal mehr festzustellen war. Die Schaltung war damit also nicht geeignet.

6.4 Synchronisation per Funk

Neben dem Ultraschall wird für die Abstandsmessung noch ein zusätzliches Signal zur Synchronisation von Sender und Empfänger benötigt. Es hat gleichzeitig die Aufgabe, die Identifikation des Senders zu übertragen. Aus Gründen der Effizienz und der einfachen Handhabung ist die Entscheidung auf eine Funkübertragung gefallen, und zwar im 433 MHz-Band.

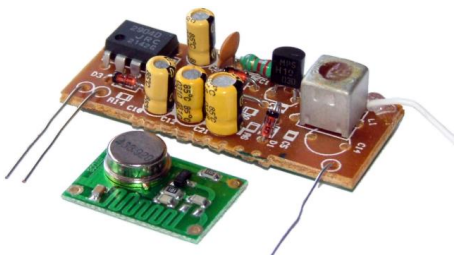


Abbildung 6.4: 433 MHz-Funkmodule

Dieses ISM-Frequenzband wird für viele Aufgaben wie Fernsteuerungen und Meßwertübertragungen benutzt, typische Anwendungen sind funkferngesteuerte Steckdosen, Wetterstationen mit externen Meßfühlern und Funkkopfhörer. Daneben ist das Band für Anwendungen mit niedriger Sendestärke bis 10 mW freigegeben, solange die Übertragungen nicht ständig erfolgen, damit ein unregulierter Betrieb mehrerer Geräte gleichzeitig möglich ist. Die in Abbildung 6.4 gezeigten Module aus dem Sortiment von Conrad Elektronik arbeiten mit einer Spannung von 5 V und verfügen über Ein- beziehungsweise Ausgänge für TTL-Signale. Die Modulation erfolgt intern, zum Senden einer logischen 1 strahlt der Sender (die kleine grüne Platine) ein Trägersignal aus, bei einer 0 wird kein Träger emittiert. Der Stromverbrauch des Senders liegt bei 10 mA während der Ausstrahlung des Trägers, sonst fließt praktisch kein Strom. Der Empfänger hingegen braucht ständig etwa 1 mA.

Im Idealfall ist die Funkübertragung für einen Benutzer transparent, in der Realität werden die Signale allerdings etwas verzerrt. Der Empfänger spricht auf ein schwaches Trägersignal verspätet an. Das führt dazu, daß die steigenden Flanken des Signals später registriert werden, während die fallenden Flanken mit deutlich geringerem Zeitversatz übertragen werden. Die nutzbare Bandbreite liegt bei 2 kHz, schnellere und sehr viel langsamere Signale können nicht übertragen werden. Daher können nur Manchester-codierte serielle Datenströme übertragen werden, was die nutzbare Bandbreite auf 1 kBit/s reduziert. Zum Überbrücken größerer Distanzen sind Antennen auf beiden Seiten nötig, dabei muß es sich um Lambda-Viertel-Antennen handeln, also Wurf- oder Stabantennen mit einer Länge von 17 cm. Andere Bauformen haben im Test deutlich schlechter funktioniert.

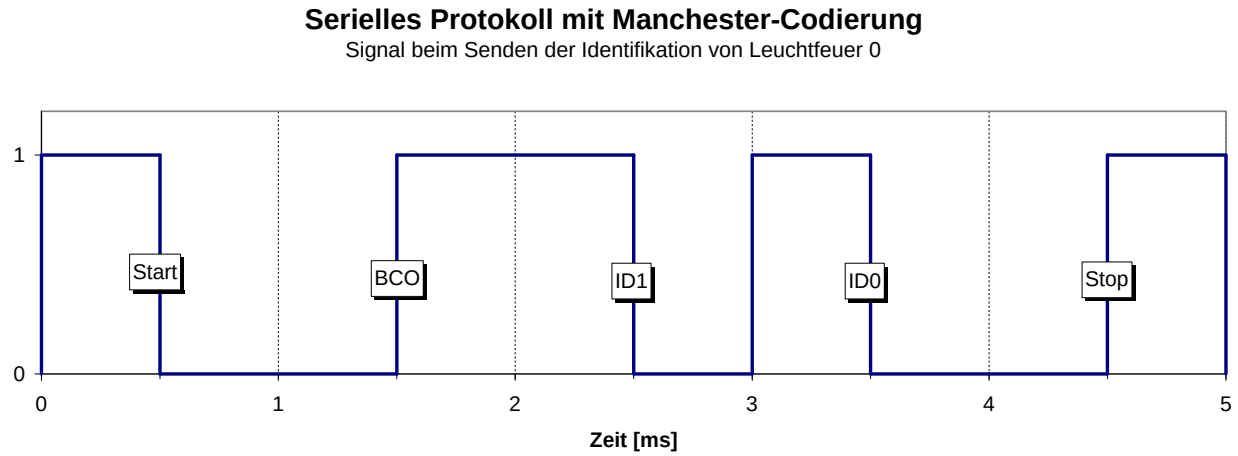


Abbildung 6.5: Manchester-Codierung beim Senden der Leuchtfeuer-Identifikation 0

Der Sender überträgt vor einem Ultraschallimpuls seine Identifikation an den Empfänger. Diese besteht aus einem Flag, das angibt, ob es sich um ein Leuchtfeuer handelt (BCO) und einer zwei Bit breiten Identifikationsnummer (ID1:ID0), die einen Wert zwischen 0 und 3 enthält. Aus diesen Werten generiert der Mikrocontroller einen codierten seriellen Datenstrom, den er direkt auf den Dateneingang des angeschlossenen Sendemoduls weitergibt.

- Aus den vorgegebenen Datenbits wird ein Wort mit einer Länge von 5 Bit erzeugt, bestehend aus einem Startbit (0), BCO, ID1, ID0 und dem Stopbit (1).
- Das Datenwort wird Manchester-codiert, eine binäre 0 wird als Bitsequenz 10, eine 1 entsprechend als 01 dargestellt. Auf diese Weite kann kein Pegel länger als zwei Takte gehalten werden, was den Anforderungen der Funkmodule entgegenkommt.
- Der codierte Datenstrom wird an das Sendemodul ausgegeben. Bei voller Ausnutzung der Bandbreite von 2 kHz dauert ein Bit 0,5 ms, das gesamte Datenwort braucht 5 ms. Ein Beispiel für die Datenübertragung ist in Abbildung 6.5 dargestellt.

Das ausgestrahlte Signal wird von dem Empfangsmodul wieder aufgefangen und kann theoretisch direkt an den einen Eingangspin des Mikrocontrollers angeschlossen werden. In der Praxis ist die Ausgangsstufe des Empfängers dafür aber nicht geeignet. Zum einen sind die Signalflanken bei weitem nicht steil genug, zum anderen liegt die Ausgangsspannung des Moduls bei einer logischen 1 deutlich unter der Versorgungsspannung. Wenn die Batterien des RCX fast leer sind, kommt die Spannung sogar in den für Logikpegel verbotenen Bereich und kann so den Mikrocontroller beschädigen. Die Signale des Funkmoduls müssen daher durch einen Schmitt-Trigger aufbereitet werden.

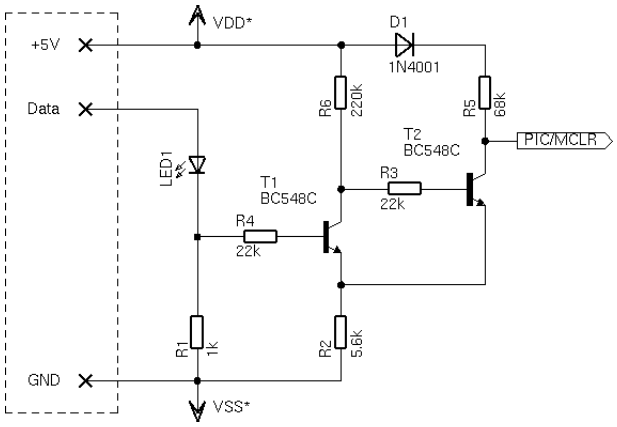


Abbildung 6.6: Schmitt-Trigger zur Anbindung des Funkempfängers

Die Schaltung in Abbildung 6.6 ist aus diskreten Komponenten aufgebaut. Ihre Dimensionierung ist darauf ausgelegt, bei 5 V einen Verbrauch von weniger als 0,1 mA zu erreichen. Die Schaltschwellen liegen bei 0,7 und 0,9 V, was eine Hysterese von 0,2 V ergibt. Der Schmitt-Trigger garantiert gültige Logikpegel bei Versorgungsspannungen bis herunter auf 2 V. Die Leuchtdiode LED_1 hat zwei Funktionen, zum einen leuchtet sie bei empfangenen Daten auf, zum anderen sperrt sie das Funkmodul, wenn die Spannung zu weit absinkt. Das Modul erzeugt dann ohne äußeren Einfluß zufällige Signale, was jede Messung unmöglich machen würde. Die Leuchtdiode reduziert die Spannung des Ausgangssignals so weit, daß der Schmitt-Trigger dann nicht mehr anspricht.

Das Programm im Mikrocontroller wertet die empfangenen Daten aus und versucht, die Identifikation des Senders zu decodieren. Dabei müssen Verzerrungen des Signals durch die Funkübertragungen und unterschiedlich schnell laufende Uhren auf beiden Seiten kompensiert werden.

Vorbemerkungen zum Algorithmus

- Das Programm erkennt Bits anhand der Zeit, die zwischen zwei Pegelwechseln vergeht. Bei einem einzelnen Bit entspricht diese einem Takt, bei zwei aufeinanderfolgenden gleichen Bits sind es zwei Takte. Andere Werte können bei der Manchester-Codierung nicht auftreten.
- Es wird immer am Anfang eines Taktes der neue Wert als Bit in einem Puffer gespeichert. Dann wartet das Programm auf das Ende des Bits.

Einlesen des Datenwortes

- Der steigende Pegel am Anfang des Startbits aktiviert die Empfangsroutine.
- Der aktuelle Wert am Ausgang des Funkmoduls wird im Bitpuffer gespeichert.
- Anschließend wartet das Programm maximal 150% der Taktdauer auf einen Pegelwechsel. Wenn dieser rechtzeitig kommt, wird die Warteschleife abgebrochen, in jedem Fall springt das Programm anschließend zum vorigen Punkt zurück. Entscheidend an der Wartezeit ist nur, daß das Programm sich auf jede erkannte Flanke synchronisiert.

- Dieser Mechanismus wiederholt sich, bis alle 10 Bits im Puffer sind. Der Empfänger hat damit das komplette Datenwort zusammen, während noch die zweite Hälfte des Stopbits übertragen wird.

Decodieren, Fehlerprüfungen und Synchronisation

- Der Mikrocontroller hat ausreichend Zeit, das Datenwort zu decodieren. Dabei wird die Identifikation des Leuchtfeuers berechnet und gespeichert.
- Für die Fehlerprüfung gibt es zwei Ansatzpunkte. Erstens sind in der Codierung nur die Bitsequenzen 01 und 10 gültig, wird 00 oder 11 empfangen, muß die gesamte Übertragung fehlerhaft sein. Zweitens müssen Start- und Stopbit die vorgegebenen Werte haben. Beide Prüfungen zusammen ergeben einen relativ sicheren Mechanismus zum Erkennen von Übertragungsfehlern und Störsendern.
- Nach den Berechnungen und Prüfungen wartet der Mikrocontroller auf die fallende Flanke am Ende des Stopbits. Sie ist für ihn das Signal, daß jetzt auch der Ultraschall ausgesendet wird. Er kann seine Uhr in diesem Moment also mit dem Sender abgleichen.

Da die Datenübertragung asymmetrisch ist (die steigende Flanke kommt immer etwas zu spät, die fallende nie), werden in der Implementierung statt 150% zwei unterschiedliche Wartezeiten benutzt. Wenn das Signal gerade 1 ist, reichen 130%, bei einer 0 sind es hingegen 170%. Das Verfahren toleriert große Abweichungen von dem vorgeschriebenen Timing. Die Uhren von Sender und Empfänger müssen auch nur ungefähr gleichschnell laufen, da bei jeder Flanke wieder synchronisiert wird.

Die Handhabung der Module hat sich als nicht ganz unproblematisch herausgestellt, kleine Variationen im Aufbau können eine große Wirkung auf die Reichweite haben. Beispielsweise hängt viel von der Form und der Ausrichtung der Antenne ab, und es darf sich kein Metall in der Nähe der Senderplatine befinden. Entgegen der Spezifikation funktioniert der Sender auch nicht mit 3 bis 12, sondern nur mit 5 V. Die Funkübertragung funktioniert zwar stabil mit einer Reichweite von 10 m, mittlerweile gibt es im Sortiment des Elektronikhandels aber bessere Module. Sie verursachen weniger Probleme und sind auch in Bezug auf Abmessungen und Stromverbrauch mit den hier verwendeten Modulen vergleichbar.

6.5 Steuerung der Leuchtfeuer

Wenn mehrere Sender koordiniert werden sollen, muß dafür ein eigenes Kommunikationssystem aufgebaut werden. Dies ist bei den Leuchtfeuern der Fall, die Steuereinheit gibt zentral die genauen Sendezeitpunkte vor und sendet entsprechende Aktivierungsbefehle an einzelne Leuchtfeuer. Da die Anzahl variabel ist, müssen für die Kommunikation einige Voraussetzungen erfüllt sein.

- Die Steuereinheit muß die Anzahl der angeschlossenen Leuchtfeuer ermitteln können, damit sie ihr Zeitschema darauf einstellen kann und nur vorhandene Stationen adressiert.
- Jedem Leuchtfeuer muß eine Nummer zugeordnet werden, die es eindeutig identifiziert. Diese Nummer sind sowohl der Steuereinheit als auch dem Benutzer des Systems bekannt und müssen immer nach dem selben Schema vergeben werden.
- Vom der Steuereinheit werden Befehle zu den Leuchtfeuern gesendet.
- Fehler in der Verkabelung sollten erkannt und vom System angezeigt werden.

Am schwierigsten ist die Numerierung der Leuchtfeuer zu realisieren. Ein erster Ansatz wäre, die Nummer über Jumper oder als Konstante in der Firmware des Mikrocontrollers vorzugeben. Sie wird dann zusätzlich auf das Gehäuse gedruckt, damit der Benutzer weiß, wo er das Leuchtfeuer aufstellen

muß. Die Steuereinheit adressiert alle möglichen Leuchtfeuer und erkennt anhand der Antworten, welche vorhanden sind. Der Ansatz hat allerdings erhebliche Nachteile, die Individualisierung der Leuchtfeuer ist sehr aufwendig, es wird ein Rückkanal zur Steuereinheit benötigt und der Ausfall eines einzigen Leuchtfeuers führt zu einem Ausfall des gesamten Systems.

Wesentlich besser ist es, wenn alle Leuchtfeuer identisch aufgebaut sind und die Numerierung durch die Verkabelung vorgegeben wird. Dann ist sie für den Benutzer nachvollziehbar und die einzelnen Stationen sind beliebig austauschbar. Durch einen trickreichen Aufbau können alle Probleme gelöst werden, ohne daß irgendeine programmgesteuerte Kommunikation nötig ist. Der Schaltplan in Abbildung 6.7 zeigt schematisch, wie die einzelnen Stationen miteinander verbunden sind.

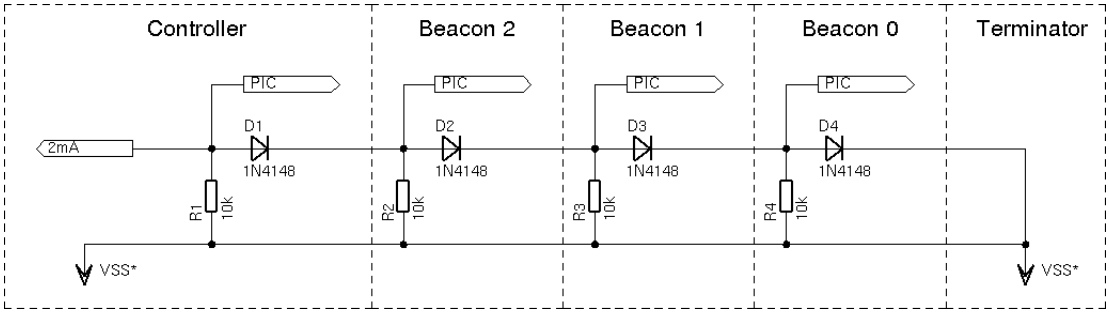


Abbildung 6.7: Schematische Darstellung der Kommunikation mit den Leuchtfeuern

Die Steuereinheit erzeugt mit einer Konstantstromquelle einen Strom von 2 mA, der in jeder Station durch eine Diode vom Typ 1N4148 geleitet wird, bevor er im Terminator auf Masse geführt wird. Bei diesem Strom fällt über jeder Diode eine Spannung von 0,6 V ab. Die Mikrocontroller in den einzelnen Stationen können die Spannung messen und damit feststellen, wieviele Dioden bzw. Stationen sich rechts von ihnen befinden. Auf diese Weise können die Leuchtfeuer durch eine einfache Spannungsmessung ihre Nummer feststellen, und die Steuereinheit ermittelt auf dem selben Weg die Anzahl der angeschlossenen Leuchtfeuer. Wenn die Verkabelung nicht in Ordnung ist, fließt der Strom über die Widerstände statt durch den Terminator ab und erzeugt dabei einen wesentlich größeren Spannungsabfall in allen Stationen vor der Fehlerstelle. Daran kann die Steuereinheit den Fehler erkennen. Das Senden von Befehlen an die Leuchtfeuer ist auch sehr einfach zu realisieren, der Mikrocontroller in der Steuereinheit kann einen Großteil des Stroms aus der Konstantstromquelle abziehen und damit die Spannung in allen Leuchtfeuern absenken. Auf diese Weise kann ein serieller Datenstrom übertragen werden. Das gleiche Kabel läßt sich auch für die Stromversorgung der Leuchtfeuer benutzen.

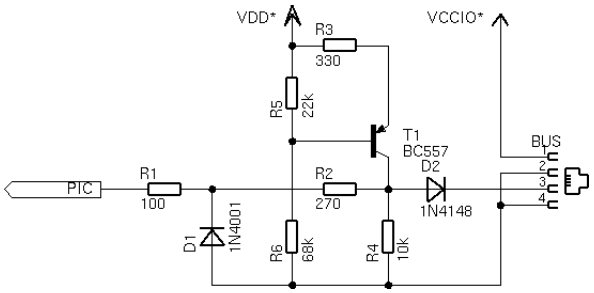


Abbildung 6.8: Businterface in der Steuereinheit

In Abbildung 6.8 ist die komplette Busanbindung der Steuereinheit zu sehen. Leitung 1 des Kabels transportiert die Versorgungsspannung des Netzteils mit bis zu 20 V, die Leitungen 2 und 4 bilden die

Masse, während Leitung 3 für die Kommunikation reserviert ist. Die Anordnung der Leitungen und die Schutzschaltungen an den Eingängen im gesamten System sind so ausgelegt, daß keine Komponente durch eine falsche Verkabelung beschädigt werden kann. Die Bauteile T_1, R_3, R_5 und R_6 bilden die Konstantstromquelle von 2 mA. D_2 und R_4 sind Bestandteil der oben erwähnten Diodenkette, R_1, R_2 und D_1 schützen den Eingang des PICs vor externen Störimpulsen. Der Mikrocontroller kann den gesamten Bus mit einem Pin steuern. Als Eingang dient er dazu, die Anzahl der Leuchtfeuer festzustellen, als Ausgang ist er für das Senden von Daten zuständig.

Da der verwendete 16F628 über keinen Analog-Digital-Wandler verfügt, muß die Spannung mit dem Komparator und der programmierbaren Referenzspannungsquelle in mehreren Versuchen eingekreist werden. Aus den Ergebnissen kann die Anzahl der Leuchtfeuer ausgerechnet oder eine falsche Verkabelung erkannt werden. Der eingeschränkte Wertebereich der Referenzspannungsquelle begrenzt die Anzahl der möglichen Leuchtfeuer auf 4, mehr sind aber auch nicht sinnvoll, weil eine Ortung dann sehr lange dauert und fehleranfällig wird. Beim Senden der Aktivierungsbefehle für einzelne Leuchtfeuer kommt exakt das selbe Protokoll wie bei der Funkübertragung zum Einsatz, allerdings mit invertierten Pegeln. Es ist in Abschnitt 6.4 beschrieben. Der Steuerpin wird zum Senden einer 1 als hochohmiger Eingang geschaltet, für eine 0 wird er als Ausgang konfiguriert und auf 0 gesetzt.

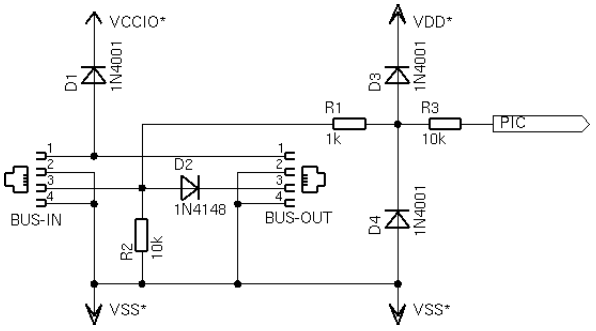


Abbildung 6.9: Businterface eines Leuchtfeuers

Das Businterface der Leuchtfeuer zeigt Abbildung 6.9. Es führt alle Leitungen von der Eingangs- zur Ausgangsbuchse und schaltet nur die Bauteile D_2 und R_2 dazwischen. Von dem Interface wird auch die Versorgungsspannung abgeleitet. Die Schutzschaltung aus R_1, R_3, D_3 und D_4 sichert den Pin des Mikrocontrollers vor Störimpulsen der permanent als Eingang konfiguriert ist.

Das Steuerprogramm des Leuchtfeuers bestimmt nach seinem Start als erstes die Position in der Kette. Dazu wird einige Zeit lang die Spannung auf dem Bus mit dem Analog-Digital-Wandler gemessen und das Maximum bestimmt. Die Messung muß lang genug dauern, daß sie auch während einer laufenden Übertragung mindestens einen hohen Pegel messen kann. Nach einer Plausibilitätskontrolle steht die Nummer fest und das Leuchtfeuer kann den normalen Betrieb aufnehmen. Um schneller auf Signale auf dem Bus reagieren zu können, wird der Komparator so programmiert, daß sein Schwellwert zwischen dem zu erwartenden Minimum und Maximum der Spannungen liegt. Danach wartet das Leuchtfeuer auf einen Aktivierungsbefehl mit seiner Nummer.

6.6 Stromversorgung

In den Schaltungen des Ortungssystems kommen zwei verschiedene Stromversorgungen zum Einsatz. Die meisten Baugruppen wie Mikrocontroller, Funkmodule, Ultraschalldetektor und die verschiedenen Kommunikationseinrichtungen brauchen eine stabilisierte Spannungsquelle von 5 V. Auf der anderen Seite benötigt der Ultraschallsender eine relativ hohe Spannung von bis zu 15 V.

Bei den Leuchtfeuern und der Steuereinheit ist die Stromversorgung unproblematisch. Hier kann ein Steckernetzteil mit 8 bis 20 V und wenigstens 150 mA benutzt werden, unterhalb von 16 V sinkt die Reichweite des Ultraschalls allerdings langsam ab. Der vom Netzteil kommende Strom wird über die Verkabelung auf alle Stationen verteilt. Dort stellen Spannungsregulatoren vom Typ 78L05 die stabilisierten 5 V für den Logikteil her. Diese Bausteine sind einfach zu beschaffen, dafür sind sie nicht sehr effizient, was bei der Stromversorgung durch ein Netzteil aber kein Problem darstellt.

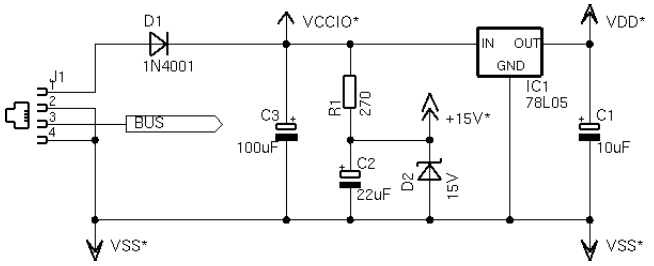


Abbildung 6.10: Stromversorgung eines Leuchtfeuers

Die Versorgung der Ultraschallsenders wird direkt aus der Netzteilspannung bestritten, wie es in Abbildung 6.10 dargestellt ist. Die Spannung wird durch die Zenerdiode D_2 auf 15 V begrenzt, wobei R_1 dafür sorgt, daß der Strom durch die Diode nicht zu groß wird. C_2 speichert Energie für das Senden und lädt sich nach dem Impuls innerhalb weniger Millisekunden wieder vollständig über R_1 auf.

Mit diesen einfachen Mitteln werden alle nötigen Spannungen für die Steuereinheit und die Leuchtfeuer erzeugt. Beim RCX sind die Voraussetzungen schwieriger, hier stehen im Batteriebetrieb typischerweise nur zwischen 6 und 8,5 V zur Verfügung. Die Basis für eine darauf aufbauende Stromversorgung bildet die in Kapitel 2 vorgestellte Grundschaltung für aktive Sensoren. Die von ihr gelieferte Spannung wird durch einen LP2950 auf 5 V stabilisiert. Dieser Regulator arbeitet bereits stabil, wenn seine Eingangsspannung nur knapp über 5 V liegt, außerdem ist sein Stromverbrauch sehr gering. Damit sind seine Leistungsdaten wesentlich besser als die eines 78L05, dafür ist er auch teurer und schwieriger zu beschaffen. Er ist aber unverzichtbar für die Anbindung empfindlicher Peripherie an den RCX. Der Sensoranschluß in Abbildung 6.11 wird mit dem Rest der Schaltung und insbesondere dem Mikrocontroller verbunden. Über diese Schnittstelle liefert der Sensor seine Meßwerte an den RCX.

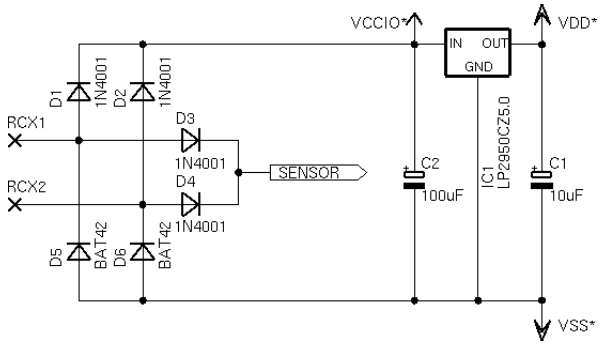


Abbildung 6.11: Stromversorgung aus dem RCX

Das selbe Interface ist prinzipiell auch für den mobilen Sender geeignet, die Spannung reicht allerdings nicht aus, um den Ultraschallsender voll auszunutzen. Als Folge wäre die Reichweite deutlich eingeschränkt. Mit einer zusätzlich eingebauten Ladungspumpe kann die Spannung für diese Stufe nahezu verdoppelt werden, dafür werden lediglich ein paar Bauteile und ein Rechtecksignal benötigt.

Abbildung 6.12 zeigt die erweiterte Stromversorgung. Die Bauteile D_8, D_9, C_1 und C_2 bilden eine Ladungspumpe, die von einem Rechtecksignal am Minuspol von C_2 angetrieben wird. Die Pumpe funktioniert so, daß Ladungen aus der Versorgungsspannung über D_8 geholt und dann über D_9 in den Kondensator C_1 gedrückt werden. Dadurch erhöht sich die Spannung über C_1, bis sie fast doppelt so hoch wie die Eingangsspannung ist.

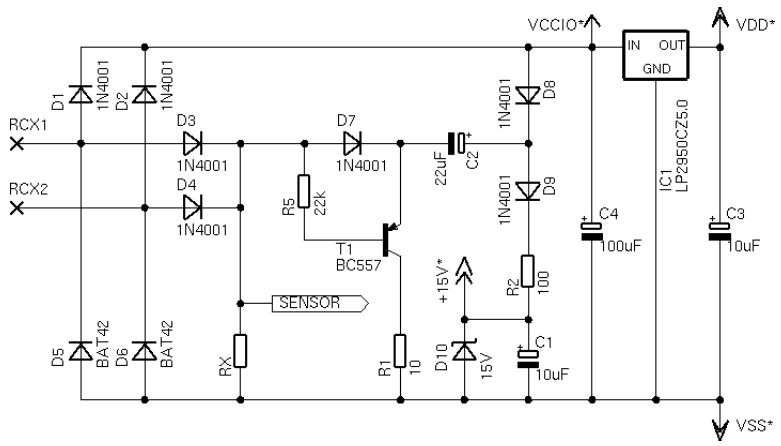


Abbildung 6.12: Stromversorgung des mobilen Senders

Es liegt nahe, das Signal von der Pulsweitenmodulation für den Antrieb der Pumpe zu benutzen. Dies hat den Vorteil, daß die Schaltung keinen eigenen Signalgenerator benötigt, andererseits ist das Rechtecksignal der PWM nicht direkt geeignet. Wenn der Aktor vom RCX nicht mit Strom versorgt werden soll, schaltet der RCX den Motorausgang hochohmig, statt ihn auf Masse zu legen (siehe Kapitel 2). Diese Funktion müssen T_1, R_1, R_5 und D_7 zusammen mit R_X übernehmen. Solange der RCX den Aktor mit Strom versorgt, sperrt T_1 und C_2 drückt Ladungen in die Pumpe hinein. Schaltet der RCX seinen Ausgang hochohmig, wird die Basis von T_1 durch die Widerstände R_5 und R_X auf Masse gezogen, wodurch der Transistor leitend wird. Er entleert dann C_2 über R_1, wie es sonst der niedrige Pegel des Rechtecksignals tun würde. D_7 verhindert dabei, daß Ladungen von der rechten Seite in die Basis gelangen können. Die erzeugte Spannung wird über eine Zenerdiode auf 15 V beschränkt, R_2 schützt die Diode vor zu hohen Strömen und C_1 dient als Energiepuffer für den Sender. Die Ladungspumpe verbraucht nur nennenswert Strom, während sie tatsächlich die Spannung über C_1 erhöht. Danach ist ihre Stromaufnahme nahezu Null.

Diese Anordnung erreicht gut brauchbare Ergebnisse. Wenn das Batteriepaket im RCX eine Spannung von 8,4 V hat, werden 8 V auf dem Aktorausgang gegeben. Hinter den Dioden des Interfaces sind davon noch 7 V übrig. Die Ladungspumpe erzeugt daraus 12,4 V für den Ultraschalltreiber. Die Spannung bricht nach dem Senden eines Impulses um 180 mV ein und erholt sich dann innerhalb von 80 ms wieder vollständig. Dafür muß brickOS allerdings ein Rechtecksignal mit einer möglichst hohen Frequenz erzeugen. Dies passiert bei einer Motorgeschwindigkeit von 50%, das resultierende Signal ist symmetrisch und hat eine Frequenz von 500 Hz.

6.7 Kommunikation mit Aktoren

Der mobile Sender ist ein Modul des Ortungssystems, das an den RCX angeschlossen werden kann. Er bezieht seine Stromversorgung aus dieser Verbindung, und über sie wird auch das Senden eines Impulses ausgelöst. Die einfachste Option zur Steuerung wäre, den Sender so zu bauen, daß er nach dem Einschalten einen einzelnen Impuls sendet. Der RCX muß dann den Motorausgang zum Senden kurz einschalten und ihn die übrige Zeit stromlos lassen. Dies ist aber nicht praktikabel, da die

Ladungspumpe im Sender eine relativ lange Vorlaufzeit zum Aufbauen der Sendespannung benötigt. Außerdem dauert es nach dem Abschalten der Stromversorgung eine Weile, bis der Sender wirklich stromlos ist. Mit diesem Ansatz kann maximal ein Impuls pro Sekunde gesendet werden, was deutlich zu langsam ist. Eine deutlich bessere Alternative ist, den Sender permanent in Bereitschaft zu halten und ihn durch eine Veränderung der Pulsweitenmodulation bei Bedarf zu aktivieren. Dabei sind drei verschiedene Einstellungen der Motorgeschwindigkeit sinnvoll.

- **Geschwindigkeit 0/8tel (Motor ausgeschaltet)**
  Mit dieser Einstellung ist der Aktor stromlos. Wenn er vorher aktiv war, verbraucht er seine in Kondensatoren gespeicherte Energie bis er sich abschaltet, auf jeden Fall sendet er nicht.
- **Geschwindigkeit 4/8tel (1 ms ein, 1 ms aus)**
  Diese Einstellung ist optimal für die Ladungspumpe, um die Sendespannung aufzubauen. Der Sender befindet sich in Bereitschaft und wartet auf den Befehl zum Senden.
- **Geschwindigkeit 7/8tel (7 ms ein, 1 ms aus)**
  Wenn der Sender diesen Zustand erkennt, sendet er einen einzelnen Impuls aus und wartet dann darauf, daß er wieder in den Bereitschaftszustand geschaltet wird. Es ist ein nützlicher Nebeneffekt, daß die Stromversorgung bei dieser Geschwindigkeit die meiste Zeit eingeschaltet ist. Dadurch steht beim Senden ausreichend Strom für alle Komponenten zur Verfügung, es werden keine großen Kondensatoren im Sender benötigt.

Der Mikrocontroller in der Schaltung muß in der Lage sein, das Signal der Pulsweitenmodulation abzutasten, was mit der Schaltung in Abbildung 6.13 relativ einfach ist. In der Mitte des Spannungsteilers aus R_1 und R_2 die aktuelle Spannung vom RCX gemessen werden. Das Signal ist hier nicht durch einen Kondensator geglättet, wie es hinter D_1 und D_2 der Fall wäre. Der Teiler ist nötig, damit keine Spannungen über 5 V an den Mikrocontroller gelangen können.

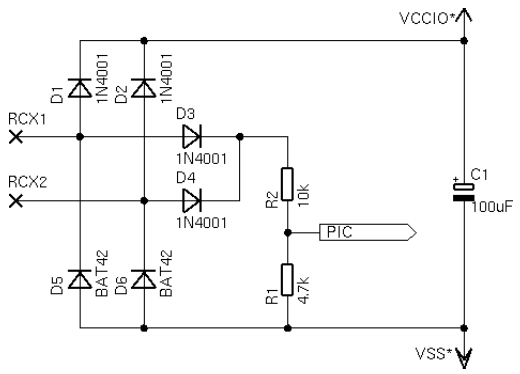


Abbildung 6.13: Interface zur Kommunikation mit dem RCX

Der Algorithmus zum Abtasten des Signals muß sich darauf einstellen, daß die Spannung vom RCX schwanken kann, je nachdem wie die Batterien gerade belastet werden. Er beginnt daher mit einer Kalibrierung und korrigiert den dabei ermittelten Wert laufend, um sich anzupassen.

Aufschalten auf das Signal

- Nach dem Einschalten kalibriert sich der Mikrocontroller auf die Batteriespannung des RCX. Dazu mißt er 10 ms lang mit den ADC das Minimum und Maximum der Pulsweitenmodulation. Liegen die Spannungen weit genug auseinander, war die Kalibrierung erfolgreich. Jetzt übernimmt ein Komparator das Abtasten, als Schwelle wird eine Spannung etwas unterhalb des vorher gemessenen Maximums programmiert. Ab diesem Moment kann der aktuelle Pegel sehr schnell über den Komparator festgestellt werden.

Verfolgen des Signals

- Das Hauptprogramm verfolgt laufend die Veränderungen der Pulsweitenmodulation.
- Wechselt das Signal auf einen hohen Pegel, wird die erreichte Spannung mit dem ADC gemessen und die Referenzspannung des Komparators entsprechend korrigiert. Dadurch werden langsame Spannungsschwankungen ausgeglichen.
- Es muß mindestens alle 10 ms ein Pegelwechsel erfolgen. Anderenfalls stimmt die Kalibrierung nicht mehr und die Messung muß wiederholt werden.

Übertragen von Kommandos

- Es wird laufend die Zeit gemessen, die das Signal auf dem hohen Pegel bleibt. Sie ist ein Maß für die aktuell eingestellte Pulsweitenmodulation.
- Wenn die Zeit erstmals 6 ms überschreitet, wurde die Geschwindigkeit gerade auf 7/8tel erhöht. Der Sender bereitet sich dann auf das Senden vor, wartet aber noch, bis die Stromversorgung wieder eingeschaltet wird. Dann wird ein einzelner Impuls gesendet.
- Unterschreitet die Zeit 5 ms, wechselt der Sender in den Bereitschaftsmodus. Er ist damit auch wieder bereit, einen weiteren Impuls zu senden.

Das auf dem RCX laufende Programm kann also durch entsprechendes Verändern der Motorgeschwindigkeit den Sender steuern. Wie eine typische Befehlssequenz aussieht, zeigt das folgende Beispiel.

- Als erstes wird der Motorausgangs mit 4/8tel Geschwindigkeit aktiviert. Nach etwa 250 ms ist der Sender voll einsatzbereit.
- Zum Senden wird die Geschwindigkeit auf 7/8tel erhöht und mindestens 20 ms auf diesem Niveau gehalten. Der Impuls wird etwa in der Mitte dieser Zeit ausgestrahlt. Anschließend wird die Geschwindigkeit wieder auf 4/8tel reduziert. Der Sender ist frühestens nach 20 ms wieder bereit, voll erholt hat er sich nach 80-100 ms.
- Der vorige Schritt wird nach Bedarf wiederholt.
- Nach dem Abschalten der Stromversorgung dauert es mindestens 250 ms, bis der Sender sich komplett abgeschaltet hat.

Wesentlich andere Geschwindigkeiten als die genannten sollten nicht benutzt werden, und die beiden Pausen von 20 ms sind nötig, damit der Sender die Einstellung sicher erkennen kann. Die Schaltung funktioniert auch, wenn sie nicht von brickOS sondern von dem Lego-Interpreter gesteuert wird. Damit können auch Sprachen wie NQC zur Programmierung benutzt werden, allerdings arbeitet die Ladungspumpe dann durch den niedrigeren Takt deutlich weniger effizient.

6.8 Datenübertragung vom Sensor

Der Empfänger des Ortungssystems zeichnet eine ganze Reihe von Meßdaten auf, dazu gehören beispielsweise die Laufzeit des Schalls und die Identifikation des Senders. Diese Daten müssen nach jedem Impuls an den RCX übermittelt werden. Die Sensorschnittstelle kann allerdings nur analoge Spannungswerte auswerten, so daß eine direkte Übertragung nicht möglich ist. In [Restena] wird eine Lösung für dieses Problem angegeben. Ein an den Mikrocontroller angeschlossener Digital-Analog-Wandler erzeugt eine in 32 Stufen verstellbare Spannung und leitet sie in das Sensorinterface. Das Programm im RCX rekonstruiert aus dem Meßwert die Spannungsstufe, so daß effektiv ein Startbit und 4 Datenbits auf einmal übermittelt werden können. Der Preis ist allerdings der große Aufwand an Hardware, außerdem ist es nicht trivial, alle 32 Stufen sicher unterscheiden zu können.

Wesentlich besser ist es, die Daten als seriellen Bitstrom zu übertragen. Dafür ist auf der Sensorseite nur sehr wenig Hardware nötig, zwei Widerstände und ein Pin des Mikrocontroller reichen bereits aus. Die Firmware im Sensor übernimmt das Senden der Daten, im RCX nimmt ein entsprechend erweitertes brickOS die Bits entgegen und setzt sie zu Bytes oder ganzen Datenpaketen zusammen. Dabei muß das Timing auf beiden Seiten exakt aufeinander abgestimmt sein. Da der Sensor in unregelmäßigen Abständen ausgelesen wird und es kein zusätzliches Synchronisationssignal gibt, muß bei jedem Auslesevorgang genau ein Bit übertragen werden.

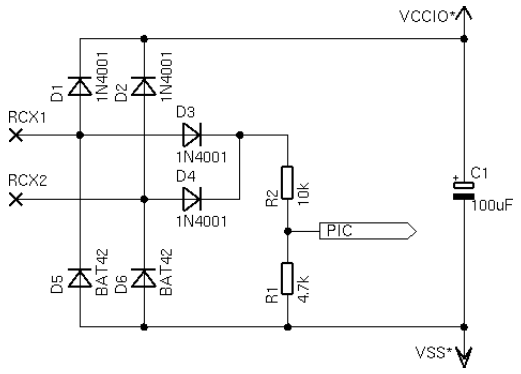


Abbildung 6.14: Interface zur Kommunikation mit dem RCX

Die Schaltung des Sensorinterfaces entspricht dem Aktorinterface und funktioniert auch genauso. Solange der RCX den Sensor mit Strom versorgt, liegt an ihm eine relativ hohe Spannung an, die der Mikrocontroller messen kann. Wenn der Auslesevorgang beginnt, bricht die Spannung deutlich ein. Diese Änderung stellt der Mikrocontroller fest und kann seinen Pin dann als Ausgang konfigurieren, um den Sensorwert zu beeinflussen. Insgesamt sind also drei Zustände möglich.

- Wenn der Anschluß des Mikrocontrollers als hochohmiger Eingang konfiguriert ist, mißt der RCX den Gesamtwiderstand des Spannungsteilers und kommt auf eine Wert von etwa 670.
- Als Ausgang mit hohem Pegel erzeugt der PIC im Inneren des Sensors eine Spannung von 5 V. Dadurch sperren die Dioden des Sensorinterfaces, es fließt kein Strom und der RCX mißt den Maximalwert 1023.
- Ein auf Masse geschalteter Ausgang schließt den Widerstand R_1 kurz. Der Sensorstrom erhöht sich, als Folge mißt der RCX mit etwa 580 den kleinsten Wert.

Wenn der Mikrocontroller einen Bitstrom senden soll, läuft dies nach dem folgenden Schema ab.

- Die Spannungen am Eingang des Mikrocontrollers müssen sehr schnell gemessen werden können. Dafür kommt nur der Komparator in Frage, der ADC ist viel zu langsam. Als erstes wird also eine Kalibriermessung durchgeführt, um die Referenzspannung für den Komparator zu ermitteln. Dazu mißt der Sensor mit dem ADC eine Zeit lang die Spannungen und bestimmt das Maximum. Die Zeit wird lang genug gewählt, daß mindestens eine Messung in die Versorgungsphase fallen muß. Die Referenzspannung wird auf einen Wert knapp unter dem Maximum eingestellt, dann übernimmt der Komparator die Messungen am Eingang. Seine Einstellung bleibt fest und kann im Laufe des Datenpaketes nicht mehr korrigiert werden.
- Der Mikrocontroller wartet dann auf eine fallende Flanke des Eingangssignals und damit den Beginn eines Auslesevorgangs. Sobald sie registriert wird, konfiguriert der Mikrocontroller den Pin als Ausgang und gibt das nächste Datenbit aus. Nach einer Wartezeit, die dem RCX genug Zeit zum Auslesen geben soll, wird der Pin wieder als Eingang konfiguriert. Dieser Schritt wiederholt sich, bis alle Bits übertragen wurden.

- Kommt es beim Warten zu einem Timeout, war die Kalibrierung wohl fehlerhaft. In so einem Fall kann die laufende Übertragung nicht mehr gerettet werden und wird stattdessen abgebrochen.

Das Timing von brickOS und dem Mikrocontroller muß genau ineinandergreifen, damit die Datenübertragung funktioniert. Der Sensorhandler von brickOS muß nach dem Abschalten der Versorgungsspannung lange genug warten, daß der Sensor reagieren und ein Datenbit auf die Leitung legen kann. Umgekehrt muß der PIC den Ausgang dann lange genug halten, bis brickOS den Sensorwert gelesen hat. Beide Zeiten müssen ausreichend lang sein, zu lang ist aber auch schädlich. Der Sensorhandler von brickOS wird so häufig aufgerufen, daß jeder zusätzliche Taktzyklus der Anwendung meßbar Rechenleistung entzieht. Im Sensor erhöht sich der Stromverbrauch unnötig, wenn der Mikrocontroller den Ausgang zu lange geschaltet läßt.

Auf der beschriebenen physikalischen Schnittstelle setzt ein Protokoll auf, mit dem sich ganze Datenbytes übertragen lassen. Dazu wird ein Startbit mit dem Wert 1 übertragen, gefolgt von den Datenbits D_7 bis D_0 des Bytes. Daran kann sich unmittelbar ein weiteres Byte mit seinem Startbit anschließen, bis ein Paket aus mehreren Bytes zusammengesetzt ist. In brickOS wurde der Sensorhandler erweitert, so daß er die übertragenen Daten zusammensetzen und an ein Anwendungsprogramm weitergeben kann. Jeder Sensor kann als serielle Datenquelle konfiguriert werden. Dann benutzt der Sensorhandler das oben beschriebene Timing und wartet laufend auf ein Startbit. Entdeckt er eines, sammelt er ab diesem Moment bei jedem Auslesevorgang ein Bit in einem Schiebepuffer ein, bis ein Byte komplett ist. Dieses wird dann an eine Callback-Funktion der Anwendung übergeben. Es liegt dann bei ihr, aus mehreren Bytes ein Paket zusammenzusetzen und es auszuwerten. In Anhang B.8 ist die Schnittstelle zu den genannten brickOS-Funktionen dokumentiert. Dabei handelt es sich um die unterste Protokollschicht, das Ortungssystem setzt noch ein benutzerfreundliches Interface darauf, das in B.9 beschrieben ist.

Das Interface zum Sensor arbeitet sehr zuverlässig und erreicht dabei Geschwindigkeiten von etwa 6 kBit/s. Die effektive Datenrate hängt aber von der brickOS-Konfiguration ab. Das Datenpaket mit den Sensordaten umfaßt 10 Bytes und ist typischerweise in 15 ms übertragen. Der Ausleserythmus der Sensoren wird zwar etwas langsamer, sobald ein serieller Sensor konfiguriert ist, außerdem gehen einige Prozent Rechenzeit verloren. Der Effekt ist aber ähnlich groß wie beim Rotationssensor und beeinträchtigt die Anwendungen in der Regel nicht.

6.9 Ansteuerung des LC-Displays

Die Steuereinheit benutzt ein alphanumerisches LCD, um Statusinformationen anzeigen zu können. Dabei handelt es sich um ein Standardmodell auf Basis des Display-Controllers HD44780, wie es in Abbildung 6.15 zu sehen ist.

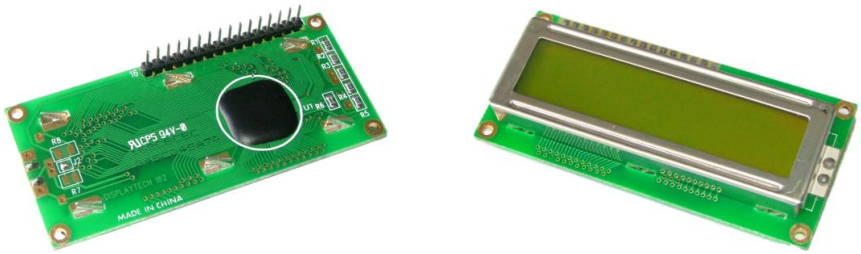


Abbildung 6.15: LC-Dotmatrixdisplay mit 16x2 Zeichen

Das Modul wird über ein 14-poliges Interface gesteuert. Zwei Leitungen sind für die Stromversorgung mit 5 V reserviert, dazu kommt eine Leitung zum Einstellen des Kontrastes. Drei weitere Pins sind

für die Steuerung zuständig und die übrigen 8 Leitungen bilden den bidirektionalen Datenbus. Der Controller unterstützt auch einen Betriebsmodus, in dem nur die oberen 4 Bits des Datenbusses benötigt werden, so daß ein Display mit 7 Pins eines Mikrocontrollers gesteuert werden kann.

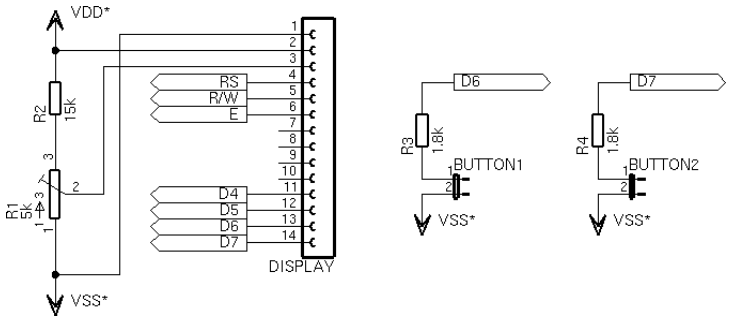


Abbildung 6.16: Ansteuerung des LC-Displays und zweier Taster

Dazu muß das Display wie im linken Teil von Abbildung 6.16 beschaltet werden. Die notwendigen Details zur Programmierung verrät das Datenblatt des HD44780, [Bredendiek] enthält einige praktische Hinweise und Beispielprogramme. Die vier Leitungen für den Datenbus können gleichzeitig noch für den Anschluß von Tastern benutzt werden. Diese werden an die Datenleitung angeschlossen und können sie über einen Pulldown-Widerstand von 1,8 kΩ auf Masse ziehen. Solange die Ausgangstreiber des Mikrocontrollers oder des Displays aktiv sind, hat der Taster keine Wirkung. Der Widerstand ist zu groß, um die Pegel nennenswert zu verändern.

Wenn das Programm den Zustand der Taster auslesen will, schaltet es das Display inaktiv und konfiguriert den entsprechenden Pin als Eingang mit Weak Pullup. Wenn der Taster nicht gedrückt ist, hält die Pullup-Funktion die Leitung auf einem hohen Pegel. Mit geschlossenem Kontakt reicht der schwache Strom nicht mehr aus und der Widerstand senkt den Pegel bis fast auf 0 V ab. Der Mikrocontroller kann so den Zustand des Tasters auslesen und anschließend den normalen Betrieb wieder aufnehmen.

Kapitel 7

Zusammensetzung der Module

Die Baugruppen aus dem vorigen Kapitel können zu den verschiedenen Modulen des Ortungssystems zusammengesetzt werden. Auf diese Weise entstehen Steuereinheit, Leuchtfeuer, mobiler Sender und Empfänger. Dem Steuerprogramm jedes Moduls fällt dabei die Aufgabe zu, die verschiedenen Funktionen zu steuern und zu koordinieren. Das folgende Kapitel beschreibt das Zusammenspiel der einzelnen Baugruppen, die vollständigen Schaltpläne und Hinweise zum Aufbau befinden sich im Anhang A.

7.1 Steuereinheit

Die Steuereinheit hat zwei Aufgaben. Sie erlaubt es dem Benutzer, über ein Interface den Status des Leuchtfeuernetzwerks einzusehen und das Sendetiming zu ändern. Daneben überwacht sie die Einhaltung des Timings, sendet Funksignale und die Aktivierungsbefehle an die einzelnen Leuchtfeuer.

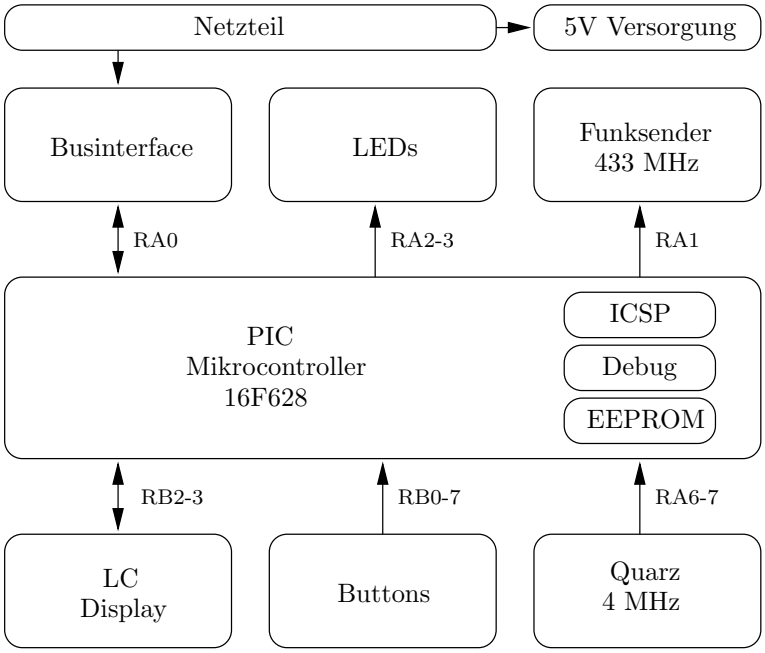


Abbildung 7.1: Schematischer Aufbau der Steuereinheit

Dementsprechend teilt sich auch der Kontrollfluß in zwei Teile auf. Die Interaktion mit dem Benutzer über das LC-Display und zwei Taster läuft im Vordergrund, zeitkritische Aktionen wie das Senden und

die Steuerung des Kommunikationsbusses werden in regelmäßigen Abständen vom Timer-Interrupt angestoßen und unterbrechen so das Hauptprogramm. Beide Stränge sind stark getrennt, sie greifen auf unterschiedliche Ressourcen des Mikrocontrollers zu und kommunizieren nur über einen kleinen Satz globaler Variablen. Die wichtigsten gemeinsamen Daten beschreiben das Sendeschema, es wird durch die drei Parameter n, T_A und T_B charakterisiert. Dabei ist n die Anzahl der Leuchtfeuer, T_A die Pause zwischen dem Senden zweier aufeinanderfolgender Leuchtfeuer und T_B die zusätzliche Pause am Ende einer Runde, wenn mehr als ein Leuchtfeuer vorhanden ist.

Nummer	**T_A**	**T_B**	**Ortung nur mit Ultraschall**
0	80 ms	0 ms	Nicht möglich
1	90 ms	0 ms	Nicht möglich
2	100 ms	0 ms	Nicht möglich
3	120 ms	0 ms	Nicht möglich
4	110 ms	20 ms	Bis Feldgröße 2x2 m
5	130 ms	30 ms	Bis Feldgröße 3x3 m
6	150 ms	50 ms	Bis Feldgröße 5x5 m
7	180 ms	60 ms	Bis Feldgröße 6x6 m
8	200 ms	70 ms	Bis zur maximalen Feldgröße
9	250 ms	80 ms	Bis zur maximalen Feldgröße
10	1000 ms	100 ms	Bis zur maximalen Feldgröße

Tabelle 7.1: Vorprogrammierte Sendezeitschemen

Die Anzahl der Leuchtfeuer wird dynamisch ermittelt, die beiden Zeiten können vom Benutzer aus einer Tabelle im EEPROM des Mikrocontrollers ausgewählt werden. Dort wird auch der gerade aktive Eintrag gespeichert, so daß die Steuereinheit nach dem Einschalten mit dem zuletzt aktiven Sendeschema weiterarbeiten kann. In Tabelle 7.1 sind die vorprogrammierten Muster aufgelistet. Das Hauptprogramm arbeitet nach einem einfachen Schema, zunächst initialisiert es das System und reagiert dann nur noch auf die verschiedenen Ereignisse.

Initialisierung

- Zunächst wird der gesamte Mikrocontroller inklusive Peripheriebausteinen initialisiert.
- Das LC-Display wird initialisiert und ein Startbildschirm ausgegeben.
- Der Mikrocontroller liest das zuletzt aktive Sendemuster aus der Tabelle im EEPROM und speichert es in globalen Variablen.
- Der Timer-Mechanismus wird aktiviert.

Reaktion auf externe Ereignisse

- Ist der linke Taster gedrückt, wird das nächstschnellere Timing aus dem EEPROM geladen und aktiviert. Entsprechend wird beim rechten Taster das Timing verlangsamt. Das neue Sendeschema wird auf den LCD angezeigt.
- Wenn der Timer-Handler eine Zustandsänderung auf dem Bus meldet (Anzahl der Leuchtfeuer geändert, Verkabelungsfehler aufgetreten oder beseitigt), wird der neue Zustand auf dem LCD angezeigt.

Der Timer-Handler wird über den Interrupt-Mechanismus exakt alle 10 ms aufgerufen. Er steuert das Businterface, führt das vom Hauptprogramm vorgegebene Sendetiming aus und sendet dazu Funksignale und Aktivierungsbefehle an die Leuchtfeuer.

Bei jedem Aufruf des Handlers

- Wenn der Sendemechanismus nicht gesperrt ist, wird die Zeit bis zum nächsten Impuls heruntergezählt.
- Erreicht der Zähler die Null, wird ein Impuls gesendet. Dazu überträgt die Steuereinheit die Identifikation des zu aktivierenden Leuchtfeuers gleichzeitig per Funk und auf dem Datenbus. Das Leuchtfeuer ist dann für das Senden des Ultraschalls zum passenden Zeitpunkt zuständig. Anschließend werden die Wartezeit bis zum nächsten Leuchtfeuer und dessen Nummer programmiert. Während des Vorgangs blitzen beide LEDs kurz auf.
- Wenn die Feuersequenz sehr langsam ist, wird ein Heartbeat-Signal mit dem Wert 0 über den Datenbus gesendet, so daß zwischen zwei Übertragungen nie mehr als 100ms vergehen. Dadurch haben die Leuchtfeuer die Möglichkeit, einen Ausfall der Kommunikation zu erkennen. Heartbeats werden nur als Lückenfüller eingesetzt, sie haben niemals Vorrang gegenüber echten Aktivierungsbefehlen.
- Anschließend wird der Zustand des Businterfaces überprüft. Es wird festgestellt, ob die Verkabelung in Ordnung ist und wie viele Leuchtfeuer angeschlossen sind. Das Hauptprogramm kann auch auf das Ergebnis der Messung zugreifen und so das LCD aktualisieren.
- Tritt bei der Messung ein Fehler auf, wird der Sendemechanismus gesperrt, bis der Fehler beseitigt ist. Während dieser Zeit leuchtet die rote LED im Dauerbetrieb. Wenn der Sender wieder reaktiviert wird, initialisiert der Handler auch alle Variablen. Dies passiert insbesondere nach dem Programmstart beim ersten Aufruf des Handlers.

Der Interrupt-Handler stellt sicher, daß die Steuereinheit das geforderte Timing möglichst präzise einhält. Durch die Teilung der Aufgaben zwischen Hauptprogramm und Handler können beide sehr einfach aufgebaut sein und müssen nur wenig Rücksicht aufeinander nehmen. Es muß lediglich sichergestellt sein, daß der knapp bemessene Stack niemals überlaufen kann.

7.2 Leuchtfeuer

Im Vergleich zur Steuereinheit sind die Leuchtfeuer relativ einfach gestrickt. Sie sind am Datenbus angeschlossen und überwachen die gesamte Kommunikation, bis sie ein Sendekommando entdecken, das an ihre Adresse gerichtet ist. Wenn ein Leuchtfeuer so einen Befehl entdeckt, muß es zu einem genau definierten Zeitpunkt nach dem Ende der Übertragung einen kurzen Ultraschallimpuls aussenden.

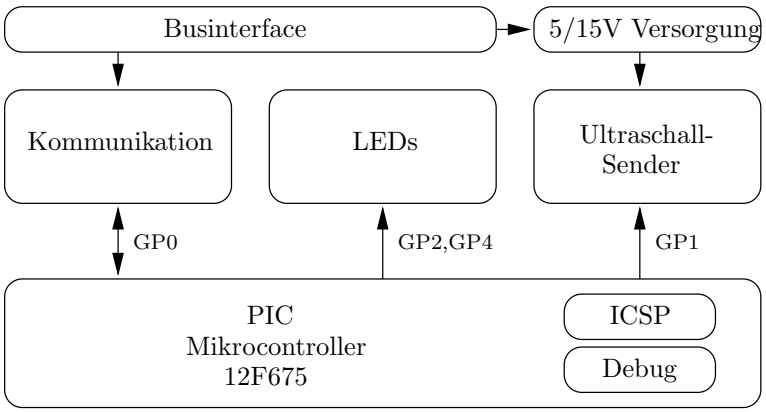


Abbildung 7.2: Schematischer Aufbau eines Leuchtfeuers

Bei jedem Aufruf des Handlers

- Bei Programmstart kalibriert sich das Leuchtfeuer auf die Spannung am Businterface. Es überprüft die Integrität des Verbindung und findet dabei auch die eigene Stationsnummer heraus, wie es in Abschnitt 6.5 beschrieben ist. Die rote LED leuchtet solange, bis die Kalibrierung erfolgreich beendet werden kann.
- Danach wird auf Datenübertragungen gewartet, es müssen Sendebefehle oder Heartbeats im Abstand von maximal 100 ms eintreffen. Wenn es zu Fehlern in der Übertragung kommt oder 128 ms lang keine Aktivität zu verzeichnen ist, kalibriert sich das Programm neu.
- Bei der Auswertung der empfangenen Daten werden alle Heartbeats ignoriert. Korrekte Sendebefehle, die aber an andere Leuchtfeuer gerichtet sind, führen zu einem kurzen Aufblitzen der grünen LED.
- Stellt das Leuchtfeuer einen an sich adressierten Sendebefehl fest, strahlt es nach dem Ende der Übertragung einen kurzen Ultraschallimpuls aus. Währenddessen leuchten sowohl die rote als auch die grüne LED für einen kurzen Moment auf.

Der einzig kritische Punkt in der Programmierung betrifft die Frage, wie das Leuchtfeuer auf das Ende der Befehlsübertragung wartet. Die Wartezeit beim Abfragen des Busses ist zwangsläufig mit einem Jitter verbunden. Dieser wirkt sich auch darauf aus, wieviel Zeit zwischen dem Funk- und dem Ultraschallsignal vergeht. Je größer der Jitter ist, desto weniger genau kann die Entfernungsmessung und damit die Ortsbestimmung erfolgen. Polling scheidet von vornherein aus, da der Zustand der Leitung innerhalb von 6 µs maximal einmal abgefragt werden kann. Ein Interrupthandler kann erst 14 µs nach dem auslösenden Ereignis den ersten Benutzerbefehl ausführen, dafür beträgt sein Jitter maximal eine Mikrosekunde. Wenn dieser Mechanismus das Ende der Funkübertragung erkennt und den Ultraschall aussendet, kann der Impuls zu einem sehr genau definierten Zeitpunkt gesendet werden.

Abbildung 7.3: Leuchtfeuer von innen

Die Taktquelle der Leuchtfeuer muß nicht sonderlich präzise sein, Abweichungen von einigen Prozent sind ohne weiteres tolerabel und stören die Genauigkeit der Abstandsmessung nicht. Die Leuchtfeuer arbeiten daher ausschließlich mit dem internen Taktgenerator.

7.3 Mobiler Sender

Der an den RCX anschließbare Sender weist einige Ähnlichkeiten zum Leuchtfeuer auf. Er muß ebenfalls auf einen Befehl von außen reagieren und daraufhin einen Impuls aussenden. Anders als beim Leuchtfeuer muß der mobile Sender den Funkimpuls selbst erzeugen, da hier keine Steuereinheit diese Aufgabe übernehmen kann. Die Schaltung erzeugt ohne Eingriff der Software auf dem PIC die nötigen Spannungen zum Senden, der Mikrocontroller muß nur die Pulsweitenmodulation vom RCX auf seinen Aktivierungsbefehl hin untersuchen.

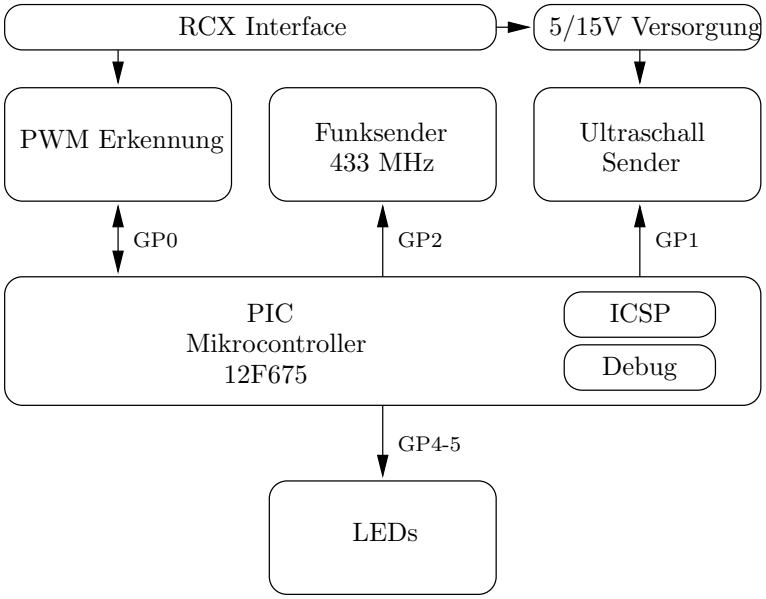


Abbildung 7.4: Schematischer Aufbau des mobilen Senders

Die Hauptschleife des Programms beschäftigt sich daher primär mit der Analyse des PWM-Signals.

Initialisierung

- Nach dem Einschalten seiner Stromversorgung beginnt der Sender mit einer Kalibrierung, bei der er sich auf die Spannungspegel vom RCX einstellt. Während dieser Phase leuchtet die rote LED permanent, bis die Kalibrierung erfolgreich beendet ist.
- Ab dann verfolgt das Programm das Signal und paßt die Kalibrierung laufend an, wie es in Abschnitt 6.7 beschrieben ist.

Reaktion auf Kommandos vom RCX

- Sobald der Mikrocontroller einen Aktivierungsbefehl entdeckt, erzeugt er zunächst ein Funksignal mit der Identifikation des mobilen Senders, darauf folgt der Ultraschallimpuls.
- Zum Anzeigen des Sendevorganges blitzen beide LEDs kurz auf.
- Sobald das PWM-Signal wieder auf den Ruhezustand wechselt, ist der Sender wieder bereit, einen weiteren Impuls zu senden.

Ein hochpräzises Timing ist auch bei dieser Schaltung nicht nötig, der RCX kann den Sendezeitpunkt sowieso nur auf einige Millisekunden genau festlegen. Daher wird der Interrupt-Mechanismus nicht benutzt und als Taktquelle reicht der interne Oszillator aus.

7.4 Empfänger

Die Funk- und Ultraschallimpulse von den verschiedenen Sendern werden von einem entsprechenden Empfänger aufgefangen und ausgewertet. Dabei wird eine Reihe von Werten gemessen oder berechnet.

- Die absolute Laufzeit des Schalls vom Sender zum Empfänger (Differenz zwischen dem Ende der Funkübertragung und dem Registrieren des Ultraschalls)
- Der relative Laufzeitunterschied (die Zeit zwischen dem Eintreffen des aktuellen und des vorigen Ultraschallimpulses)
- Die Zeit, in der das Echo des Ultraschallimpulses mit dem Detektor nachweisbar ist
- Die Identifikation des Senders, wie sie per Funk übermittelt wurde
- Verschiedene Flags, welche die Gültigkeit der einzelnen Meßwerte anzeigen

Eine ganze Reihe von Baugruppen muß zusammenarbeiten, um die gewünschte Funktionalität zu erreichen, und die Schaltung nutzt alle Ressourcen des Mikrocontrollers voll aus. Die grüne LED muß sogar direkt vom Funkmodul gesteuert werden, da der Mikrocontroller keinen Pin mehr frei hat.

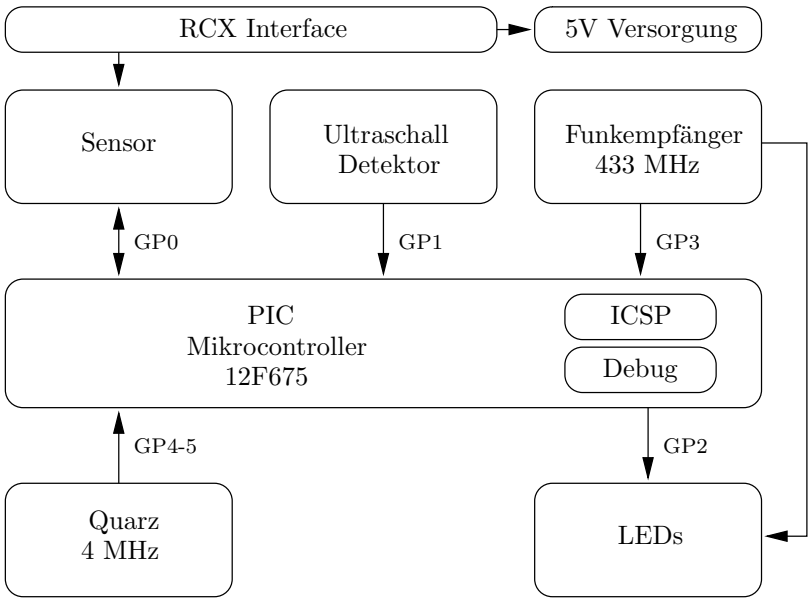


Abbildung 7.5: Schematischer Aufbau des Empfängers

Die Software im Mikrocontroller wartet auf Signale von den Detektoren und reagiert entsprechend.

Initialisierung

- Nach dem Einschalten der Stromversorgung konfiguriert das Programm alle Komponenten und wartet dann, bis beide Detektoren stabil sind und keine Störsignale mehr liefern.

Hauptschleife

- Timer 1 zählt laufend die Mikrosekunden hoch, das Steuerprogramm erweitert ihn um ein manuell aktualisiertes Byte auf eine Breite von 24 Bit.

- Bei jedem Eintreffen eines Ultraschallimpulses wird der Timer ausgelesen und wieder auf 0 gesetzt. Der ausgelesene Wert entspricht genau dem Laufzeitunterschied.
- Wenn der Empfänger ein Funksignal registriert, decodiert er es und speichert die Identifikation des Senders. Am Ende der Übertragung wird zusätzlich eine Zwischenzeit von Timer 1 genommen. Auf diese Weise kann später die absolute Laufzeit durch Subtraktion der beiden Zeiten bestimmt werden.
- Wenn der Ultraschallimpuls ankommt, wird noch die Zeit abgewartet, bis das Echo abgeklungen ist und der Detektor 10 ms lang nicht mehr anspricht. Die Zeit (abzüglich der 10 ms) wird als Echozeit gespeichert.
- Mit dieser Information ist das Paket mit den Meßdaten komplett und kann an den RCX übermittelt werden. Wie dies im einzelnen geschieht, beschreibt Abschnitt 6.8.
- Nach der Übertragung beginnt der Zyklus mit laufendem Timer 1 wieder von vorne.

Damit die Messungen mit möglichst wenig Jitter erfolgen, benutzt das Programm den Interrupt-Mechanismus. Sobald einer der beiden Detektoren seinen Zustand ändert, wird ein Handler aufgerufen, der die untersten 8 Bit von Timer 1 speichert und den Interrupt-Mechanismus wieder deaktiviert, bevor er sich beendet. Das Hauptprogramm kann so mit einer relativ einfachen Kontrollstruktur arbeiten und die Detektoren per Polling abfragen. Die dabei festgestellten Zeiten sind zwar mit einem Jitter von einigen Mikrosekunden behaftet, durch Korrektur mit dem vom Interrupt gespeicherten Wert kann der Jitter aber auf eine Mikrosekunde reduziert werden. Das Programm muß den Interrupt nur bei Bedarf für die richtige Quelle aktivieren.

Abbildung 7.6: Platine eines Empfängers

Die grüne LED wird direkt von der Elektronik des Funkempfängers angesteuert und blitzt beim Empfang eines Signals auf. Der Mikrocontroller läßt die rote LED beim Empfang von Ultraschall aufleuchten.

Kapitel 8

Einsatz aus Anwendersicht

Das Ortungssystem hat die Aufgabe, die Koordinaten eines Mindstorms-Roboters zu ermitteln, der sich innerhalb eines Feldes der Größe a $\times$ b befindet. Zur Orientierung dienen ihm dabei sogenannte Leuchtfeuer, dabei handelt es sich um Basisstationen an festen Positionen am Feldrand. Die Bestimmung der Koordinaten kann prinzipiell auf drei verschiedene Arten erfolgen.

Mobiler Empfänger

- Die Leuchtfeuer können Impulse aussenden und werden dabei von der Steuereinheit koordiniert, so daß sie reihum und ist festen Abständen senden. Ein auf dem Roboter montierter Empfänger registriert diese Impulse. Im einzelnen sind dies ein Funksignal, das zur Synchronisation dient und außerdem die Identifikation des Senders überträgt, sowie einen darauf folgenden Ultraschallimpuls. Der Empfänger mißt die Zeitdifferenz, mit der die beiden Signale eintreffen. Damit kennt er die Laufzeit des Schalls und damit auch die Entfernung des Roboters zum Sender. Die Daten werden an den RCX übertragen und dort ausgewertet. Aus allen Entfernungen und der Geometrie des Aufbaus kann der Roboter seine Position berechnen.
- Das Funksignal ist eine große Erleichterung bei der Ortung, es geht aber auch ohne. Der Empfänger kann die Zeitdifferenzen zwischen dem Eintreffen aufeinanderfolgender Ultraschallimpulse messen und daraus seine Position bestimmen. Dieses Verfahren ist aufwendiger und etwas anfälliger für Fehler, es funktioniert aber ebenfalls. Der Aufbau ist der selbe wie im vorigen Punkt, der RCX wertet jetzt allerdings andere Meßdaten des Empfängers aus und ignoriert alles, was mit Hilfe des Funksignals ermittelt wurde.

Mobiler Sender

- Es ist möglich, die Rollen von Sender und Empfänger zu vertauschen. In diesem Fall sendet der Roboter die Impulse aus und die Leuchtfeuer registrieren sie. Das Verfahren ist schneller, da für eine Ortung nur ein einziger Impuls benötigt wird. Dafür müssen die Leuchtfeuer unter Umständen alle Meßdaten zum Roboter zurücksenden. Dieser Aufbau ist mit den vorhandenen Modulen möglich, indem die beiden Empfänger stationär aufgestellt und mit einem RCX verbunden werden. Für den Roboter gibt es einen eigenen Sender, der an einem Aktorausgang betrieben werden kann.
- Auch in diesem Fall wäre es denkbar, eine Ortung ohne das Funksignal zu programmieren. Dies scheitert bei den Modulen aus dieser Arbeit allerdings daran, daß die Uhren der einzelnen Empfänger nicht präzise genug synchronisiert werden können. Der Schwerpunkt der Arbeit lag aber darauf, ein System mit mobilem Empfänger zu entwickeln.

Dieses Kapitel faßt zusammen, was ein Benutzer für den Einsatz des Systems benötigt. Dabei behandelt es Aspekte vom Aufbau bis hin zur Koordinatenberechnung und erreichbaren Genauigkeit.

8.1 Installation des Systems

Der Aufbau des Ortungssystems beginnt mit der Entscheidung, wie viele Leuchtfeuer aufgestellt und wie diese angeordnet werden sollen. Eine Grundlage können die in Kapitel 4 beschriebenen Szenarien darstellen. Dort wird auch beschrieben, worauf bei der Anordnung sinnvollerweise geachtet werden muß, damit die Koordinatenberechnung problemlos möglich ist.

8.1.1 Aufbau mit mobilem Empfänger

Die Installation des Leuchtfeuernetzwerkes beginnt mit der Steuereinheit. Sie bezieht ihre Stromversorgung aus einem Steckernetzteil mit 8-20 V, wobei die Reichweite der Sender bei höheren Spannungen etwas größer ist. Kabel mit Westernsteckern an den Enden stellen die Verbindungen zu den Leuchtfeuern her, die in einer Kette verbunden sind. Die Ausgangsbuchse der Steuereinheit wird mit dem Eingang eines Leuchtfeuers verbunden. An dessen Ausgang kann der Eingang des nächsten Leuchtfeuers angeschlossen werden. Diese Anordnung setzt sich fort, bis die Kette durch einen Terminator abgeschlossen wird. Dabei handelt es sich um das kurze Kabelstück mit einem rotem Knopf am Ende, der in die Ausgangsbuchse des letzten Leuchtfeuers gesteckt wird. Das Beispiel in Abbildung 8.1 zeigt die Verkabelung für zwei Leuchtfeuer.

Abbildung 8.1: Installation der Leuchtfeuer

Die Numerierung der einzelnen Module erfolgt über die Verkabelung. Das Leuchtfeuer mit dem Terminator bekommt die Nummer 0, zur Steuereinheit hin wachsen die Nummern in Einerschritten an. Beim Aufbau sollte darauf geachtet werden, daß die Sender etwa auf der selben Höhe stehen, in der sich auch der Empfänger auf dem Roboter befindet. Der Luftraum zwischen den Leuchtfeuern und dem Roboter muß frei von Hindernissen sein, die den Ultraschall aufhalten oder umlenken könnten. Des weiteren muß die Position der Leuchtfeuer sehr präzise mit den Vorgabewerten übereinstimmen, weil sonst die Koordinatenberechnung mit unnötig verfälschten Werten zu kämpfen hat. Wird die Steuereinheit jetzt über den Schalter an der Seite eingeschaltet, initialisiert sich das System und das LC-Display zeigt die Anzahl der angeschlossenen Leuchtfeuer und das aktive Zeitschema für das Senden der Impulse an.

Stimmt die Verkabelung nicht, können die Anzeigen der beteiligten Module bei der Fehlersuche helfen.

- **Rote LED leuchtet ständig**: Einerseits wird das Modul mit Strom versorgt und funktioniert, andererseits gibt es ein Problem bei der Kommunikation. Ursachen können der Datenbus der Leuchtfeuer oder die Verbindung zum RCX sein.
- **Rote LED blitzt auf**: Dieses Modul sendet oder empfängt gerade einen Ultraschallimpuls.
- **Grüne LED blitzt auf**: Es wird gerade ein Funksignal gesendet oder empfangen. Bei der Steuereinheit und den Leuchtfeuern blinken die grünen LEDs immer gleichzeitig auf, obwohl der einzige Funksender in diesem Gespann in der Steuereinheit steckt.

Noch hilfreicher sind bei der Fehlersuche die Meldungen auf dem Display der Steuereinheit.

- **SHO Beacon Controller**: Diese Meldung ist nach dem Einschalten zu sehen, bis das System vollständig initialisiert ist. Der gesamte Vorgang dauert etwa 2 Sekunden.
- **No beacons**: Es sind keine Leuchtfeuer angeschlossen, die Steuereinheit ist direkt terminiert.
- **Wiring error**: Die Verkabelung zu den Leuchtfeuern ist nicht in Ordnung. Wahrscheinlich sind irgendwo Eingang und Ausgang vertauscht, oder der Terminator ist nicht angeschlossen. Der Fehler liegt eventuell bei der ersten Station, deren rote LED nicht leuchtet.
- **Using N beacon(s)**: Das System ist korrekt verkabelt und arbeitet normal.

Wenn das System arbeitet, zeigt die untere Zeile des LC-Displays das gerade aktive Sendeschema in der Form $N \times T_A + T_B$ an. Dabei ist N die Anzahl N der aktiven Leuchtfeuer und T_A gibt an, wie lange die Pause zwischen zwei gesendeten Impulsen ist. Da die Leuchtfeuer reihum senden, wird am Ende einer Runde zusätzlich noch die Zeit T_B abgewartet. Bei einem einzigen Leuchtfeuer wird im Display nur T_A angezeigt und die Steuereinheit verzichtet auf die zusätzliche Pause am Rundenende. Der Benutzer kann mit den beiden roten Tastern im laufenden Betrieb auf ein schnelleres oder langsameres Zeitschema umschalten. Die Steuereinheit schaltet dann auf eines von mehreren vorprogrammierten Zeitschemen um. Die angezeigte Zeit entspricht genau der mittleren Zeit für einen Ortungsvorgang, also z.B. 3×150ms+50ms=500ms. Je kürzer die Pausen sind, desto schneller kann der Roboter geortet werden, allerdings gibt das Echo im Raum eine feste Untergrenze vor. Die Berechnung der Zeitschemen wird in Abschnitt 8.5 besprochen.

Abbildung 8.2: Roboter mit Empfänger

Der Empfänger wird wie ein normaler Legostein auf den Roboter gesteckt und an einen der Sensoreingänge angeschlossen. Es muß nur darauf geachtet werden, daß sich keine größeren Aufbauten um den Sensor herum befinden, die den Ultraschall ablenken könnten. Falls der Roboter beim Fahren starke

Vibrationen erzeugt, die den Empfänger durchrütteln und zu Fehlern führen, kann ein entsprechender Dämpfer zwischen den Roboter und den Empfänger gesetzt werden. Dabei handelt es sich um zwei Legoplatten, zwischen denen ein Stück Neopren eingeklebt ist. Diese einfache Konstruktion schwächt die Vibrationen stark genug ab, so daß der Empfänger auch während der Fahrt normal funktioniert.

8.1.2 Installation mit mobilem Sender

Die Leuchtfeuer können auch die Rolle der Empfänger übernehmen, wenn stattdessen ein Sender auf dem Roboter installiert wird, der Aufbau ist nahezu identisch. Der Sender kann auf den Roboter gesteckt und an einen Motorausgang angeschlossen werden. Zwei Leuchtfeuer werden an festen Orten aufgestellt und mit einem stationären RCX verbunden. Dieser kann die Meßdaten beider Empfänger auswerten und die resultierenden Daten an den Roboter zurücksenden.

Abbildung 8.3: Zwei Empfänger als Leuchtfeuer

Alternativ kann jedes Leuchtfeuer auch einen eigenen RCX enthalten, dann müssen deren Sendevorgänge koordiniert werden, damit es nicht zu Kollisionen kommen kann. Es können auch mehr als zwei Empfänger eingesetzt werden, sofern diese vorhanden sind. Im Rahmen dieser Arbeit wurden allerdings nur zwei Exemplare gebaut.

8.2 Ansteuerung mit brickOS

Damit Empfänger und Sender vom RCX angesteuert werden können, muß Unterstützung durch brickOS für diese Module vorhanden sein. Der Sender wird über Veränderungen der Pulsweitenmodulation gesteuert und braucht daher nur einige Makros für die Ansteuerung. Der Sensor ist wesentlich komplizierter, hier muß der Sensorcode von brickOS entsprechend erweitert werden, damit er die Meßdaten zu einem Paket zusammensetzen und einer Callback-Funktion der Anwendung übergeben kann. Alle Patches für brickOS 0.9.0 sind in den Anhängen B.8 und B.9 dokumentiert.

8.2.1 Ansteuerung des Empfängers

Das erweiterte brickOS unterstützt eine neue Klasse von aktiven Sensoren, die ihre Daten als seriellen Datenstrom an den RCX senden. Der Empfänger des Ortungssystems gehört zu dieser Klasse. Jedesmal, wenn er einen Ultraschallimpuls erkennt, wartet er das Ende des Impulses ab und überträgt dann ein Datenpaket mit einer Größe von 10 Bytes seriell an den RCX. Der Sensorcode von brickOS sammelt auf der untersten Schicht die einzelnen Bits ein und fügt sie zu Bytes zusammen. Diese werden an

einen Handler für Positionsdaten weitergegeben, der das Paket zusammenführt und an eine Callback-Funktion der Anwendung weitergibt. Die Meßdaten umfassen eine Reihe von Werten, Abbildung 8.4 zeigt den Aufbau des Paketes.

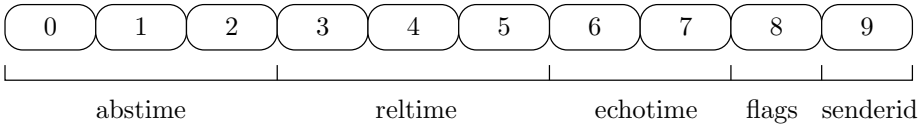


Abbildung 8.4: Aufbau des Meßdatenpaketes vom Empfänger

Die Felder mit mehreren Bytes enthalten Werte im Big-Endian-Format. Bits im Flags-Feld entscheiden darüber, welche Angaben gültig sind und welche falsche oder veraltete Werte enthalten.

- **abstime** gibt die absolute Laufzeit des Ultraschalls vom Sender zum Empfänger in Mikrosekunden an. Der Wert umfaßt 24 Bit, typische Werte liegen zwischen 100 und 32000. Dieser Wert wird mit Hilfe des Funksignals ermittelt, er ist dementsprechend nur gültig, wenn das Flag RFVALID gesetzt ist.
- **reltime** enthält die Zeit in Mikrosekunden, die zwischen dem Eintreffen des letzten und des aktuellen Ultraschallsignals vergangen ist. Auch dieser Wert ist mit 24 Bit gespeichert, die Werte beginnen bei 50000 und können abhängig vom Sendezeitschema bis über eine Million anwachsen. Definiert ist der Wert nur, wenn das Flags USVALID gesetzt ist.
- **echotime** gibt an, wie viele Millisekunden das Echo des aktuellen Ultraschallimpulses im Raum noch nachweisbar war, nachdem der Detektor ausgelöst hat. Dieser Wert ist zu einen wichtig, um das schnellste erlaubte Sendetiming zu ermitteln, zum anderen kann er aber auch zum Erkennen von Störimpulsen dienen, die nicht von den Leuchtfeuern kommen. Typische Werte liegen zwischen 30 und 60, was aber stark von der Akustik des Raumes abhängt. Um schnellere Impulsfolgen zu erlauben, kann der Empfänger auf die Messung dieser Zeit verzichten, was eine geänderte Firmware erfordert. Das Flag ECVALID ist gesetzt, wenn dieser Wert gültig ist.
- **senderid** enthält die Identifikation des Senders, von dem die aktuellen Daten stammen. Die untersten beiden Bits enthalten die Nummer der Station von 0 bis 3, das Bit darüber ist gesetzt, wenn es sich bei der Quelle um ein Leuchtfeuer (nicht einen mobilen Sender) handelt. Die Identifikation wird per Funk übermittelt, sie gilt also nur bei gesetztem RFVALID.
- **flags** ist die Bitmaske mit den verschiedenen Gültigkeitsflags. Dies sind im einzelnen RFVALID (Bit 0), USVALID (Bit 1) und ECVALID (Bit 2).

Der Empfänger wird wie jeder andere Sensor auch an einen der drei Eingänge des RCX angeschlossen und kann dann über das Interface in `position.h` gesteuert werden. Im Programm muß der Sensor als erstes konfiguriert werden, diese Aufgabe übernimmt die Routine `posrecv_enable`, hier für Sensor 1.

```
posrecv_enable(&SENSOR_1);
```

Anschließend ist der Sensor einsatzbereit. Wenn er einen Impuls erkannt hat, sendet er ein Paket mit den Meßdaten an den RCX, das brickOS an eine vorher gesetzte Callback-Funktion weitergeben kann.

```
void handler(unsigned char *packet)
{
  ...
}

posrecv_set_handler(&SENSOR_1, handler);
```

Die Callback-Funktion wird direkt aus dem Interrupt-Handler für die Sensoren heraus aufgerufen. Bei ihrer Programmierung muß daher beachtet werden, daß sie sich so schnell wie möglich beendet und nur die nötigsten Systemfunktionen aufruft. Idealerweise erstellt sie nur eine Kopie der Daten, wo sie dann vom Rest des Programms ausgewertet werden können.

```
posrecv_disable(&SENSOR_1);
```

Wird der Sensor nicht mehr benötigt, kann er mit `posrecv_disable` wieder abgeschaltet werden. Die übrigen Funktionen dienen dazu, auf die einzelnen Felder in den Meßdaten zugreifen zu können. Sie bekommen dazu einen Zeiger auf das Paket übergeben.

```
unsigned long posrecv_abstime(unsigned char *packet);
unsigned long posrecv_reltime(unsigned char *packet);
unsigned int  posrecv_echotime(unsigned char *packet);
unsigned char posrecv_senderid(unsigned char *packet);

unsigned char posrecv_rfvalid(unsigned char *packet);
unsigned char posrecv_usvalid(unsigned char *packet);
unsigned char posrecv_ecvalid(unsigned char *packet);
unsigned char posrecv_frombeacon(unsigned char *packet);
```

Die erste Funktionsgruppe liefert die Inhalte der jeweiligen Felder aus dem Paket, während die zweite Gruppe über boolesche Rückgabewerte angibt, ob das betreffende Bit in den Flags oder in der Identifikation des Senders gesetzt ist. Das Bit zur Unterscheidung von Leuchtfeuern und mobilen Sendern wird von `posrecv_senderid` entfernt und muß über `posrecv_frombeacon` abgefragt werden.

Das folgende Programm stellt ein Minimalgerüst für die Entwicklung eigener Programme dar, es geht von einem Empfänger an Sensoreingang 1 aus. Der Handler kopiert die Daten in einen globalen Datenpuffer, der über einen Mutex-Mechanismus abgesichert ist. Seine zwei Variablen stellen sicher, daß niemals das Hauptprogramm und der Handler gleichzeit auf den Paketpuffer zugreifen können.

```
#include <position.h>

unsigned char lastpacket[POSRECV_PACKETSIZE];
volatile int mutex_newdata = 0;
volatile int mutex_processed = 1;

void handler(unsigned char *packet)
{
  if (mutex_processed != 0) {
    memcpy(lastpacket, packet, POSRECV_PACKETSIZE);
    mutex_processed = 0;
    mutex_newdata = 1;
  }
}

int main(void)
{
  posrecv_enable(&SENSOR_1);
  posrecv_set_handler(&SENSOR_1, handler);
```

```
  while (...) {
    if (mutex_newdata != 0) {
      ... Code zum Auswerten des Paketes ...
      mutex_newdata = 0;
      mutex_processed = 1;
    }
  }
  posrecv_disable(&SENSOR_1);
  return(0);
}
```

Das Hauptprogramm greift mit den üblichen Funktionen auf den Inhalt des kopierten Paketes zu. Der folgende Code gibt die Entfernung zum Leuchtfeuer 0 in Zentimetern mit einer Nachkommastelle aus.

```
if (posrecv_rfvalid(lastpacket)) {
  if ((posrecv_frombeacon(lastpaket)) && ((posrecv_senderid(lastpacket)) == 0)) {
    lcd_int((posrecv_abstime(lastpacket)*343)/1000-45);
    dlcd_show(LCD_2_DOT);
  }
} else cputs("----");
```

Der Code prüft zunächst über das Flag RFVALID, ob die Angaben in senderid und abstime überhaupt gültig sind. Wenn dies der Fall ist und der Impuls von Leuchtfeuer 0 kommt, wird die Entfernung auf dem Display ausgegeben und das Komma gesetzt. Die Formel geht davon aus, daß die Schallgeschwindigkeit 343 m/s beträgt und die Entfernung durch die Reflektoren um 45 mm verlängert wird.

8.2.2 Ansteuerung des Senders

Der Sender kann an einen beliebigen Motorausgang angeschlossen werden und wird durch Veränderung der Geschwindigkeit und damit der Pulsweitenmodulation des Ausgangs gesteuert. Das API umfaßt drei Funktionen pro Motorausgang, zusätzlich wird noch die Konstante `POSSEND_BURSTTIME` definiert. Ein an den Ausgang A des RCX angeschlossener Sender kann mit dem Befehl `possend_a_enable` eingeschaltet und in den Bereitschaftsmodus versetzt werden. Etwa 250 ms nach diesem Befehl ist der Sender einsatzbereit, dann kann mit `possend_a_burst` das Senden eines Impulses ausgelöst werden. Die Funktion braucht dafür die in `POSSEND_BURSTTIME` angegebene Anzahl von Millisekunden, der eigentliche Impuls wird irgendwo mitten in dieser Zeitspanne gesendet. Wenn der Sender nicht mehr gebraucht wird, schaltet ein Aufruf von `possend_a_disable` ihn wieder ab. Die Befehle für die anderen Ausgänge lauten entsprechend. Ein Programmfragment, das alle 200 ms einen Impuls senden soll, sieht also folgendermaßen aus.

```
#include <position.h>

possend_a_enable();
msleep(250);
while (...) {
  possend_a_burst();
  msleep(200-POSSEND_BURSTTIME);
}
possend_a_disable();
```

Der Sendezeitpunkt kann durch die Art der Kommunikation zwischen RCX und Sender nicht auf die Millisekunde genau gesteuert werden, die 200 ms werden also nur im zeitlichen Mittel erreicht.

8.3 Auswertung der Meßdaten

In Kapitel 4 wurde bereits ausführlich dargelegt, wie aus den Meßdaten des Sensors die Position des Roboters berechnet werden kann. Allerdings ist die Implementierung der Formeln auf dem RCX nicht unproblematisch. Dort sind die Datentypen float und double beide als Fließkommazahlen mit einer Breite von 32 Bit ausgelegt, deren Mantisse 23 Bit umfaßt (siehe [Krimer]). Werden die Formeln direkt in den Code übernommen, reicht die Genauigkeit bei weitem nicht aus, wie schon eine einfache Abschätzung zeigt. Die Schallgeschwindigkeit c liegt um 340 m/s, für ihre Speicherung mit einer Genauigkeit von einem Meter pro Sekunde werden also 9 Bit benötigt. Bei der absoluten Laufzeit des Schalls können Werte zwischen 0 und 32000 µs auftreten, wofür 15 Bit erforderlich sind. Die Auswertung des häufig vorkommenden Term c^2t^2 mit voller Genauigkeit belegt dann bereits

$$2 \cdot 9 + 2 \cdot 15 \quad = \quad 48$$

signifikante Binärstellen. Unter brickOS kommt es daher sehr leicht zu Rundungsfehlern, Überläufen und numerischer Auslöschung. Ein weiteres Problem ist, daß es dort keine funktionierende Bibliothek mit mathematischen Routinen gibt, so daß einfache Funktionen wie die Wurzelberechnung von Hand programmiert werden müssen. Der resultierende Code ist langsam und groß, kann numerisch instabil sein kann und liefert trotzdem keine akzeptable Genauigkeit. Der einzige Ausweg aus diesem Problem ist, auf Festkommaarithmetik umzusteigen, was eine Reihe von Vorteilen mit sich bringt.

- Wertebereich und Genauigkeit sind bei Eingabewerten, Zwischenergebnissen und den Resultaten genau definiert. Sie sind statisch und hängen nur von den Vorgaben des Programmierers ab.
- Damit ist es auch möglich, Überläufe sicher zu verhindern.
- Der resultierende Code ist deutlich schneller und kompakter als der auf Fließkommazahlen und den nötigen Emulationsfunktionen basierende Code.
- Rundungsfehler können statisch und relativ einfach abgeschätzt werden.

In den folgenden Abschnitten werden Rechenverfahren für die gängigsten Probleme aufgeführt. Sie gehen von der selben Anordnung der Leuchtfeuer wie in Kapitel 4 aus, insbesondere bilden L_1 und L_2 die Basis jeder Koordinatenberechnung. Sie werden bei Bedarf um weitere Leuchtfeuer ergänzt.

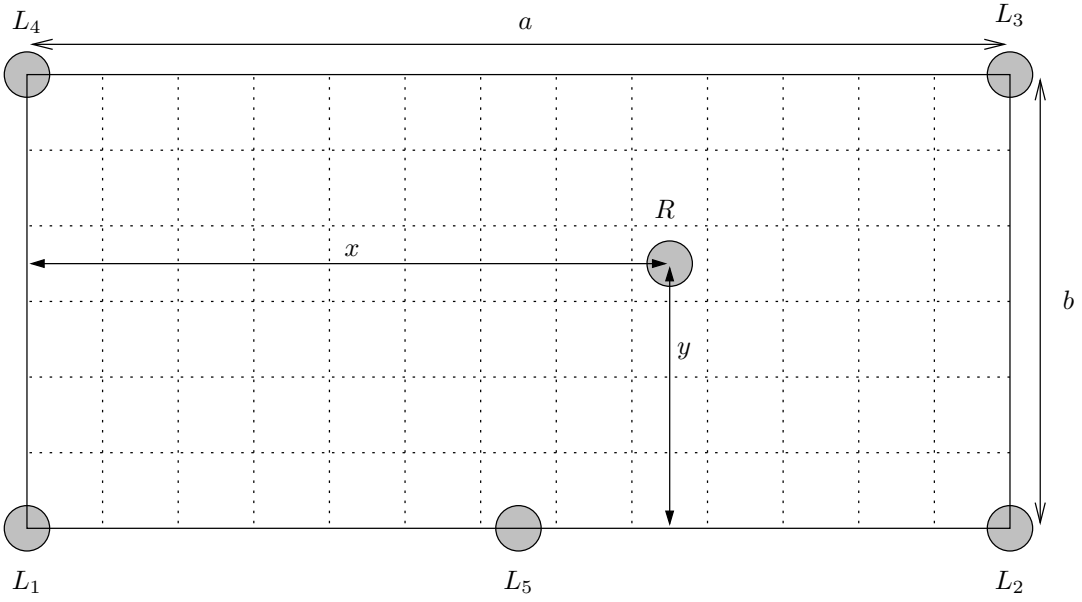


Abbildung 8.5: Anordnung der Leuchtfeuer für die Berechnung der Koordinaten

8.3.1 Gemeinsame Voraussetzungen

Vor jeder Festkommaberechnung müssen die verschiedenen physikalischen Größen skaliert und auf ganze Zahlen gerundet werden. Der Faktor wird dabei so gewählt, daß für die gewünschte Genauigkeit ausreichend viele Dezimalstellen erhalten bleiben. Auf diese Weise entsteht aus jeder physikalischen Größe g die entsprechende Festkommadarstellung $\overline{g}$.

- Die Schallgeschwindigkeit $\overline{c}$ wird in Dezimetern pro Sekunde angegeben. Sie liegt damit zwischen 3200 und 3600, bei 20 °C Raumtemperatur beträgt der Wert etwa 3430.
- Mit $\overline{T}$ wird die Raumtemperatur bezeichnet, gemessen in Zehntelgrad auf der Celsius-Skala. Der Wertebereich liegt zwischen -200 und 400, 200 entspricht der Raumtemperatur von 20 °C.
- $\overline{t_i}$ sind die absoluten Laufzeiten des Schalls zwischen dem Leuchtfeuer i und dem Roboter im Mikrosekunden. Bei einer Reichweite von 10 m und unter Berücksichtigung der unterschiedlichen Schallgeschwindigkeiten liegen gültige Werte zwischen 100 und 32000.
- Die Koordinaten $\overline{x}$ und $\overline{y}$ des Roboters werden in Millimetern angegeben. Das gleiche gilt für $\overline{a}$ und $\overline{b}$, welche die Größe des Feldes beschreiben. Bei einer Reichweite von 10 Metern kann ein quadratisches Feld maximal eine Kantenlänge von 7000 mm haben.
- Der Schall legt auf dem Weg vom Sender zum Empfänger eine Strecke zurück, die länger als die Verbindungslinie der beiden Stationen ist. Die Differenz kommt durch den Umweg über die beiden Reflektoren zustande, sie wird mit $\overline{z}$ bezeichnet und in Millimetern gemessen. Erfahrungsgemäß ist diese Strecke 45 mm lang. Der Schall legt diesen Umweg in der Zeit $\overline{t_z}$ zurück, die in Mikrosekunden gemessen wird.

Tabelle 8.1 faßt alle Umrechnungsformeln und die erlaubten Wertebereiche zusammen. Die verschiedenen Algorithmen sind für diese Rahmenbedingungen ausgelegt, das umgebende Programm muß daher sicherstellen, daß Eingabewerte nicht außerhalb der erlaubten Bereiche liegen können. Anderenfalls kann es zu Überläufen und ähnlichen Fehlern kommen.

Umrechnung			**Wertebereich**			
$\overline{c}$	=	$10 \cdot c$	3200	$\leq \overline{c} \leq$	3600
$\overline{T}$	=	$10 \cdot T$	-200	$\leq \overline{T} \leq$	400
$\overline{x}$	=	$10^3 \cdot x$	0	$\leq \overline{x} \leq$	7000
$\overline{y}$	=	$10^3 \cdot y$	0	$\leq \overline{y} \leq$	7000
$\overline{a}$	=	$10^3 \cdot a$	0	$\leq \overline{a} \leq$	7000
$\overline{b}$	=	$10^3 \cdot b$	0	$\leq \overline{b} \leq$	7000
$\overline{z}$	=	$10^3 \cdot z$	0	$\leq \overline{z} \leq$	100
$\overline{t_i}$	=	$10^6 \cdot t_i$	100	$\leq \overline{t_i} \leq$	32000
$\overline{t_z}$	=	$10^6 \cdot t_z$	0	$\leq \overline{t_z} \leq$	300

Tabelle 8.1: Wertebereiche und Skalierungen für Festkommaarithmetik

8.3.2 Bestimmung der Schallgeschwindigkeit

Die Schallgeschwindigkeit c geht zwangsläufig in jede Koordinatenberechnung mit ein. Sie ändert sich mit der Lufttemperatur, daher muß c vorgegeben sein oder bestimmt werden, bevor die eigentlichen Koordinaten ausgerechnet werden können. Der erste Ansatzpunkt dafür ist Gleichung (4.3), die den Zusammenhang zwischen T und der Schallgeschwindigkeit in Luft beschreibt.

Im Bereich der normalen Umgebungstemperaturen reicht allerdings eine Näherungsformel aus.

$$c = T \cdot 0,583 \frac{\mathrm{m}}{^\circ\mathrm{C} \cdot \mathrm{s}} + 331,6 \frac{\mathrm{m}}{\mathrm{s}} \tag{8.1}$$

Durch Multiplikation mit 10 und geeignetes Umstellen lassen sich alle vorkommenden Größen so umformen, daß die Gleichung auf das Einheitensystem der Festkommadarstellung übergeht.

$$\begin{aligned} 10c &= 10T \cdot 0,583 \frac{\mathrm{m}}{^\circ\mathrm{C} \cdot \mathrm{s}} + 3316 \frac{\mathrm{m}}{\mathrm{s}} \\ \overline{c} &= \overline{T} \cdot \frac{583}{1000} \frac{\mathrm{m}}{^\circ\mathrm{C} \cdot \mathrm{s}} + 3316 \frac{\mathrm{m}}{\mathrm{s}} \end{aligned} \tag{8.2}$$

Diese Darstellung kann direkt in ein entsprechendes Stück C-Code umgesetzt werden, das nur mit Operationen auf Ganzzahlen auskommt. Die Umkehrabbildung wird auf die selbe Weise gebildet.

```
c = ((long)T*583)/1000+3316;
T = (((long)c-3316)*1000)/583;
```

Der Type Cast-Operator ist unter brickOS nötig, damit die Zwischenergebnisse der Rechnung mit 32 Bit gespeichert werden. Ein 16 Bit breiter Integer könnte bei der Multiplikation T*583 bereits überlaufen. Wenn es auf eine sehr hohe Genauigkeit ankommt, kann die ganzzahlige Division durch eine exakt rundende Konstruktion ersetzt werden (`sgn` entspricht der Signumsfunktion).

```
c = ((long)T*583+sgn(T)*1000/2)/1000+3316;
T = (((long)c-3316)*1000+sgn(c)*583/2)/583;
```

Falls die Umgebungstemperatur im Einsatzgebiet des Roboters konstant ist, kann die Schallgeschwindigkeit mit diesen Formeln statisch ausgerechnet und als Konstante in den Programmcode gesetzt werden. Besser ist es natürlich, die Temperatur zur Laufzeit zu messen und die Schallgeschwindigkeit entsprechend anzupassen. Der Temperatursensor von Lego ist für diese Aufgabe gut geeignet.

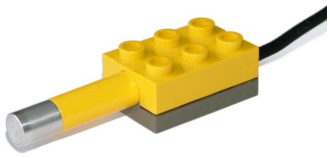


Abbildung 8.6: Mindstorms Temperatursensor

Aus dem vom Sensor gelieferten Rohwert n muß zunächst die Temperatur ausgerechnet werden. Der Bytecode-Interpreter von Lego benutzt dafür laut [Gasperi] eine einfache, aber ungenaue Formel.

$$T = \frac{785 - n}{8} ^\circ\mathrm{C} = 98,125 ^\circ\mathrm{C} - n \cdot 0,125 ^\circ\mathrm{C} \tag{8.3}$$

BrickOS enthält eine präzisere Formel, die sich gut in die Festkommadarstellung überführen läßt.

$$T = 93.8136 ^\circ\mathrm{C} - n \cdot 0.122241 ^\circ\mathrm{C} \tag{8.4}$$

$$\begin{aligned} 10T &\approx 938 ^\circ\mathrm{C} - n \cdot \frac{1000}{818} ^\circ\mathrm{C} \\ \overline{T} &= 938 ^\circ\mathrm{C} - n \cdot \frac{1000}{818} ^\circ\mathrm{C} \end{aligned} \tag{8.5}$$

Daraus ergibt sich ein Codefragment für brickOS, das die Temperatur in Zehntelgrad berechnen kann.

```
T = 938-((long)ds_scale(SENSOR_1)*1000)/818;
```

Es ist aber auch möglich, die Schallgeschwindigkeit ohne Hilfe zusätzlicher Sensoren alleine mit dem Ortungssystem zu ermitteln. Dazu werden die beiden Leuchtfeuer L_1 und L_2 benötigt, die an einem Rand des Feldes im Abstand a aufgestellt sind. Der Empfänger wird irgendwo auf der Verbindungslinie plaziert und mißt dort die absoluten Laufzeiten t_1 und t_2, die genau den den Strecken x+z und (a-x)+z entsprechen. Durch Addition der beiden Messungen entsteht eine Formel, die x nicht mehr enthält und so unabhängig von der genauen Position des Empfängers ist.

$$\begin{aligned} c(t_1 + t_2) &= a + 2z \qquad (8.6) \\ 10 \cdot c(10^6 t_1 + 10^6 t_2) &= (10^3 a + 2 \cdot 10^3 z) \cdot 10^4 \\ \overline{c}\left(\overline{t_1} + \overline{t_2}\right) &= (\overline{a} + 2\overline{z}) \cdot 10^4 \\ \overline{c} &= \frac{(\overline{a} + 2\overline{z}) \cdot 10000}{\overline{t_1} + \overline{t_2}} \qquad (8.7) \end{aligned}$$

Die Implementierung dieser Formel in brickOS ergibt sich direkt aus der letzten Gleichung.

```
c = (((long)a+2*z)*10000)/((long)t1+t2);
```

Eine vollkommen andere Methode zur Bestimmung der Schallgeschwindigkeit wird im Kapitel 4 bei Szenario 3 erwähnt. Stellt man in die Mitte zwischen den Leuchtfeuern L_1 und L_2 zusätzlich noch das Leuchtfeuer L_5 auf, steht dem Empfänger ein Meßwert mehr zur Verfügung. Durch die zusätzliche Information ist es möglich, die Schallgeschwindigkeit unabhängig von den Koordinaten auszurechnen. Die dafür nötige Gleichung (4.23) kann einfach in Festkommadarstellung umgeformt werden.

$$\begin{aligned} c^2 &= \frac{a^2}{2\left(t_1^2 + t_2^2 - 2t_5^2\right)} \qquad (8.8) \\ c &= \frac{a}{\sqrt{2\left(t_1^2 + t_2^2 - 2t_5^2\right)}} \\ 10 \cdot c &= \frac{10^7 a}{10^6\sqrt{2\left(t_1^2 + t_2^2 - 2t_5^2\right)}} \\ 10 \cdot c &= \frac{\left(10^3 a\right) \cdot 10^4}{\sqrt{2\left((10^6 t_1)^2 + (10^6 t_2)^2 - 2(10^6 t_5)^2\right)}} \\ \overline{c} &= \frac{\overline{a} \cdot 10000}{\sqrt{2(\overline{t_1}^2 + \overline{t_2}^2 - 2\overline{t_5}^2)}} \qquad (8.9) \end{aligned}$$

Für die Wurzelberechnung unter brickOS kann die Funktion `sqrt32` aus Anhang B.10 benutzt werden. Sie zieht in konstanter Zeit die Wurzel aus einer beliebigen natürlichen Zahl von 32 Bit und gibt ein richtig gerundetes Ergebnis zurück.

```
#define sqr(x) ((x)*(x))

c = ((long)a*10000)/(sqrt32(2*(sqr((long)t1)+sqr((long)t2)-2*sqr((long)t5))));
```

Wenn die Schallgeschwindigkeit erst einmal bestimmt ist, können aus ihr und den Meßwerten des Empfängers auch die Koordinaten des Roboters berechnet werden.

8.3.3 Berechnung der Koordinaten

Beim Einsatz der Leuchtfeuer L_1 und L_2 existiert mit Gleichung (4.21) eine sehr einfache Formel zur Berechnung der X-Koordinaten. Sie wird nach dem bekannten Muster für die Berechnung umgeformt.

$$\begin{aligned} x &= \frac{a^2 + c^2(t_1^2 - t_2^2)}{2a} \\ x &= \frac{a}{2} + \frac{c^2(t_1^2 - t_2^2)}{2a} \\ 10^3 x &= \frac{10^3 a}{2} + \frac{10^{14} c^2(t_1^2 - t_2^2)}{10^{11} \cdot 2a} \\ 10^3 x &= \frac{10^3 a}{2} + \frac{(10c)^2\left((10^6 t_1)^2 - (10^6 t_2)^2\right)}{10^8 \cdot 2(10^3 a)} \\ \overline{x} &= \frac{\overline{a}}{2} + \frac{\overline{c}^2(\overline{t_1}^2 - \overline{t_2}^2)}{10^8 \cdot 2\overline{a}} \end{aligned} \tag{8.10}$$

In dem zweiten Bruch auf der rechten Seite sind mit der Feldbreite $\overline{a}$ und der Schallgeschwindigkeit $\overline{c}$ mehrere Konstanten vorhanden, die sich zu einem Wert d_a zusammenfassen lassen.

$$d_a = \frac{2\overline{a} \cdot 10^8}{\overline{c}^2} \tag{8.11}$$

$$\overline{x} = \frac{\overline{a}}{2} + \frac{\overline{t_1}^2 - \overline{t_2}^2}{d_a} \tag{8.12}$$

Die Y-Koordinate kann nach dem selben Schema berechnet werden, wenn die Leuchtfeuer L_2 und L_3 vorhanden sind. Der Rechenweg führt auf die folgenden Gleichungen.

$$d_b = \frac{2\overline{b} \cdot 10^8}{\overline{c}^2} \tag{8.13}$$

$$\overline{y} = \frac{\overline{b}}{2} + \frac{\overline{t_2}^2 - \overline{t_3}^2}{d_b} \tag{8.14}$$

Bei der Implementierung dieser Formeln muß berücksichtigt werden, daß die vom Empfänger gelieferten Laufzeiten noch den Offset t_z enthalten. Dieser Wert muß vor der Koordinatenberechnung subtrahiert werden. Im Einheitensystem der Festkommadarstellung kann er einfach ermittelt werden.

$$\begin{aligned} t_z &= \frac{z}{c} \\ 10^6 t_z &= \frac{10^4 \cdot (10^3 z)}{10c} \end{aligned} \tag{8.15}$$

$$\overline{t_z} := \frac{10^4 \cdot \overline{z}}{\overline{c}} \tag{8.16}$$

Bei der direkten Berechnung von d_a und d_b reichen 32 Bit nicht aus, um den Zähler darzustellen. Dieses Problem kann aber einfach umgangen werden, indem die Rechenschritte aufgeteilt und in der richtigen Reihenfolge wieder zusammengesetzt werden. Der vollständige Code lautet damit wie folgt.

```
tz = ((long)z*10000)/c;
da = ((((long)2*a*100000)/c)*1000)/c;
db = ((((long)2*b*100000)/c)*1000)/c;
x = a/2+(sqr((long)t1-tz)-sqr((long)t2-tz))/da;
y = b/2+(sqr((long)t2-tz)-sqr((long)t3-tz))/db;
```

Die Konstanten t_z, d_a und d_b müssen natürlich nicht jedesmal neu berechnet werden.

Wenn die Ortung mit nur zwei Leuchtfeuern auskommen soll, muß die Y-Koordinate anders ermittelt werden. Die einzige Möglichkeit hierfür wird durch Gleichung 4.12 beschrieben.

$$y = \sqrt{c^2 t_1^2 - x^2} \tag{8.17}$$

Für die Darstellung des Radikanten in dieser Form werden allerdings sehr viele Bits gebraucht, daher bietet sich eine Zerlegung nach der dritten binomischen Formel in zwei Teilterme an.

$$y = \sqrt{ct_1 + x} \cdot \sqrt{ct_1 - x} \tag{8.18}$$

Die Umformung auf das andere Einheitensystem läuft nach dem bekannten Schema ab. Diesmal wird die Erweiterung aber so angelegt, daß die Terme unter den Wurzeln jeweils mit 10 erweitert werden. Dadurch wird der Wertebereich voll ausgenutzt und die Genauigkeit verbessert.

$$\begin{aligned}
10^3 y &= \frac{10^8 \sqrt{ct_1 + x} \cdot \sqrt{ct_1 - x}}{10^5} \\
10^3 y &= \frac{\sqrt{10^8 (ct_1 + x)} \cdot \sqrt{10^8 (ct_1 - x)}}{10^5} \\
10^3 y &= \frac{\sqrt{10(10c)(10^6 t_1) + 10^5 (10^3 x)} \cdot \sqrt{10(10c)(10^6 t_1) - 10^5 (10^3 x))}}{10^5} \\
\overline{y} &= \frac{\sqrt{10\overline{ct_1} + 10^5 \overline{x}} \cdot \sqrt{10\overline{ct_1} - 10^5 \overline{x}}}{10^5}
\end{aligned} \tag{8.19}$$

Ein kurzes Stück C-Code implementiert die vollständige Berechnung der beiden Koordinaten. Auch hier müssen die Konstanten t_z und d_a nicht ständig neu berechnet werden.

```
tz = ((long)z*10000)/c;
da = ((((long)2*a*100000)/c)*1000)/c;
x = a/2+(sqr((long)t1-tz)-sqr((long)t2-tz))/da;
y = ((long)sqrt32(10*c*((long)t1-tz)+(long)x*100000)
          *sqrt32(10*c*((long)t1-tz)-(long)x*100000))/100000;
```

8.4 Ortung ohne Funksignal

Die bisher vorgestellten Verfahren zur Koordinatenberechnung funktionieren nur mit Hilfe des Funksignals. Es wird zur Identifikation des Senders und zur Bestimmung der absoluten Laufzeit benutzt und geht daher als implizite Voraussetzung an mehreren Stellen ein. Wenn das Ortungssystem ohne das zusätzliche Signal arbeiten soll, stehen stehen diese Informationen nicht mehr zur Verfügung. Die Felder abstime, senderid und frombeacon können nicht mehr benutzt werden, stattdessen enthalten nur noch reltime und echotime gültige Werte. Die Berechnung der Koordinaten ist dementsprechend schwieriger, erfordert ein Leuchtfeuer mehr als die bisherigen Techniken und ist mit etwas größeren Fehlern verbunden, sie funktioniert aber ebenfalls. Die wichtigste Voraussetzung ist dabei, daß die Steuereinheit die Leuchtfeuer nach einem sehr präzisen Zeitschema reihum senden läßt und am Ende einer Runde eine ausreichend lange Pause verstreichen läßt, bevor sie wieder von vorne anfängt.

$$N \times T_A + T_B \tag{8.20}$$

Die Sendezeitschemen werden mit dem obigen Ausdruck beschrieben. Die Angabe besagt, daß N Leuchtfeuer vorhanden sind, die im Abstand von T_A reihum senden. Nach dem letzten Leuchtfeuer wird zusätzlich noch die Zeit T_B gewartet, bevor der Zyklus wieder von vorne beginnt. Die Gesamtdauer

einer Runde entspricht damit genau dem Wert des Ausdrucks (8.20). Tabelle 8.2 zeigt als Beispiel, wie das Sendeschema mit N=3, T_A=120 ms und T_B=30 ms realisiert ist.

Zeit	Aktion	Pause danach
0 ms	Leuchtfeuer 0 sendet	120 ms
120 ms	Leuchtfeuer 1 sendet	120 ms
240 ms	Leuchtfeuer 2 sendet	120 + 30 ms
390 ms	Leuchtfeuer 0 sendet	120 ms
...	...	...

Tabelle 8.2: Beispiel für ein Sendetiming

Das im Rahmen dieser Arbeit gebaute Ortungssystem kann auch komplett ohne Funksignale arbeiten. Die Hauptfunktion des Empfängers ist das Messen der Zeit, die zwischen zwei Impulsen vergeht. Beim Eintreffen eines Funksignals wird nur die übermittelte Identifikation gespeichert und eine Zwischenzeit genommen, mit der später die absolute Laufzeit berechnet werden kann. Ohne diesen Mechanismus sind die entsprechenden Daten nicht verfügbar, der Rest der Messungen funktioniert aber trotzdem problemlos. Um eine Koordinatenberechnung unter diesen Bedingungen zu testen, muß der Funksender aber nicht ausgelötet oder in der Firmware deaktiviert werden, auch wenn dies problemlos möglich wäre. Stattdessen kann der Benutzer aber auch auf die Auswertung der fraglichen Felder verzichten und nur auf den Rest zugreifen. Die Wirkung ist identisch, da das Funksignal auch nicht versteckt in die übrigen Messungen eingeht.

8.4.1 Identifikation des Senders

Beim Verzicht auf das Funksignal ist es nicht mehr einfach möglich, jeden Impuls eindeutig einem der Sender zuzuordnen. Die Sender können in dem Ultraschallsignal keine Hinweise unterbringen, daher kann die Entscheidung nur über das Sendezeitschema getroffen werden. An dieser Stelle bekommt die Extrapause T_B ihre Bedeutung. Der vom Empfänger gemessene Laufzeitunterschied zwischen dem letzten und ersten Leuchtfeuer ist immer größer als der Abstand zwischen zwei beliebigen anderen, nacheinander sendenden Leuchtfeuern. Daher muß der Impuls, der die längste Zeitspanne vor sich hat, von Leuchtfeuer 0 kommen. Die darauffolgenden Impulse werden entsprechend fortlaufend numeriert. Abhängig von der Position des Roboters können die Meßwerte in gewissen Bereichen schwanken. Daher muß die Extrapause lang genug gewählt werden, so daß die Pause vor Leuchtfeuer 0 unter allen Umständen länger als die übrigen ist. Details zu ihrer Berechnung werden in Abschnitt 8.5 beschrieben.

Ein einfacher Algorithmus kann also die Zuordnung der Meßwerte zu Leuchtfeuern übernehmen. Gleichzeitig können die Meßwerte auf Gültigkeit geprüft werden. Sind alle Meßwerte einer Runde gültig, entspricht ihre Summe etwa der Rundendauer $N \cdot T_A + T_B$.

Voraussetzungen

- Die Parameter N, T_A und T_B werden als Konstanten vorgegeben. Sie lassen sich nur mit einigem Aufwand durch eine reine Analyse der Meßdaten herausfinden. Insbesondere bei der Aufteilung der Rundendauer in T_A und T_B müssen Annahmen über die möglichen Zeitschemen und die ungefähre Position des Empfängers gemacht werden. Es ist daher wesentlich einfacher und zuverlässiger, die Parameter genau wie die Feldgröße als Konstanten im Programmcode vorzugeben.
- Das Programm sammelt laufend die Meßwerte des Empfängers in einem Puffer von N Elementen. Dabei wird jedesmal der erste Wert verworfen, die nachfolgenden Werte rücken auf und der neue Meßwert wird hinten angehängt.

Suche nach einer gültigen Meßreihe

- Die Meßwerte im Puffer heißen $\mathrm{rel}_0 \ldots \mathrm{rel}_{N-1}$, wobei rel_0 der älteste Wert ist.
- Für jeden neuen Wert werden zwei Kriterien überprüft. (8.21) stellt sicher, daß es sich um eine gültige Reihe handelt, während (8.22) für die Zuordnung zu den Sendern zuständig ist.

$$2000 \geq \left| N \cdot T_A + T_B - \sum_{i=0}^{N-1} rel_i \right| \tag{8.21}$$

$$rel_0 > rel_i \quad \forall i \in [1, \ldots, N-1] \tag{8.22}$$

- Diese Integritätsprüfungen können natürlich auch bei Verwendung des Funksignals zum Erkennen von Fehlern genutzt werden. Die Fehlergrenze von 2000 ist ein Kompromiß, der experimentell gefunden wurde. Der Wert muß groß genug sein, so daß kleinere Schwankungen und Meßfehler noch toleriert werden. Andererseits steigt zusammen mit ihm auch die Wahrscheinlichkeit, daß Fehler nicht erkannt werden.
- Wenn beide Bedingungen zutreffen, enthält jedes $\mathrm{rel_i}$ mit sehr hoher Wahrscheinlichkeit die gültigen Meßdaten von Leuchtfeuer i.

Wenn der Algorithmus erfolgreich ausgeführt wurde, stehen die Laufzeitunterschiede für alle Leuchtfeuer fest und können eindeutig einer Quelle zugeordnet werden. Die dabei ermittelten Zahlen $\mathrm{rel_i}$ werden anschließend weiterverarbeitet, um zu einer Position zu kommen.

8.4.2 Rekonstruktion der absoluten Laufzeiten

Der zweite Schritt der Auswertung beschäftigt sich damit, die absoluten Laufzeiten aus den gemessenen Unterschieden zu rekonstruieren. Die dafür nötigen Formeln wurden in Kapitel 4 hergeleitet, sie sind aber noch nicht direkt verwendbar. Erst müssen die Meßdaten noch in die Form umgerechnet werden, die von den Gleichungen erwartet wird. Insbesondere erwarten die Formeln Laufzeitunterschiede, die relativ zum ersten Leuchtfeuer gemessen wurden, während der Empfänger immer die Unterschiede von einem Impuls zum nächsten mißt. Außerdem müssen noch die Pausen $\mathrm{T_A}$ und $\mathrm{T_B}$ herausgerechnet und unterschiedliche Geschwindigkeiten der Uhren in Sender und Empfänger herausgerechnet werden.

Voraussetzungen

- Es liegen gültige Meßwerte $\mathrm{rel}_0 \ldots \mathrm{rel}_{N-1}$ vor. Dabei ist $\mathrm{rel_i}$ der Laufzeitunterschied, den der Empfänger beim Ankommen des Impulses von Leuchtfeuer i gemessen hat. Die Werte sind also ein gültiger Datensatz, wie er vom vorigen Algorithmus erzeugt wurde.

Vorbereitung der Meßwerte

- Die Uhren von Steuereinheit und Empfänger enthalten Taktgeneratoren, die auf Quarzen basieren. Dennoch können sie im Bereich von einigen hundert ppm unterschiedlich schnell laufen, was sich in Meßfehlern von entsprechend vielen Mikrosekunden äußert. Die Abweichung muß daher so gut wie möglich herausgerechnet werden. Es reicht dafür bereits aus, die Werte $\mathrm{rel_i}$ so zu skalieren, daß ihre Summe $\mathrm{N{\cdot}T_A{+}T_B}$ beträgt.

$$s = \sum_{i=0}^{N-1} rel_i \tag{8.23}$$

$$rel_i := rel_i \cdot \frac{N \cdot T_A + T_B}{s} \tag{8.24}$$

Die so korrigierten Meßwerte können in den nächsten Schritten weiterverarbeitet werden.

- In den gemessenen Zeiten sind noch die beiden Pausen T_A und T_B enthalten. Die Formeln zur Koordinatenberechnung gehen von Impulsen aus, die gleichzeitig gesendet und zu unterschiedlichen Zeiten empfangen wurden. Daher müssen die Differenzen subtrahiert werden, um die Meßwerte in die erforderliche Darstellung zu bringen.

$$rel_0 := rel_0 - T_A - T_B \tag{8.25}$$
$$rel_i := rel_i - T_A \quad \forall i \in [1, \ldots, N-1] \tag{8.26}$$

  Die von den Reflektoren verursachte Streckenverlängerung z spielt hier übrigens keine Rolle, sie wurde bereits bei der Bildung der Laufzeitdifferenzen subtrahiert.

Ausrechnen der absoluten Laufzeiten

- Die Formeln aus Kapitel 4 gehen davon aus, daß alle Laufzeitunterschiede relativ zum ersten Leuchtfeuer gemessen werden, während der Empfänger immer von einem Leuchtfeuer zum nächsten mißt. Die Meßwerte rel_i entsprechen nach den vorangehenden Korrekturen jeweils genau einer Laufzeitdifferenz t_{jk} im Sinne der Formeln, allerdings in der Regel nicht der richtigen. Für die weitere Berechnung werden nur Ausdrücke der Form t_{m1} benötigt. Diese lassen sich aber durch ein paar einfache Rechenregeln herstellen.

$$t_{ji} = -t_{ij} \tag{8.27}$$
$$t_{ik} = t_{ij} + t_{jk} \tag{8.28}$$

  Der genaue Rechenweg hängt von der tatsächlichen Anordnung der Leuchtfeuer ab, Beispielrechnungen für ein konkretes Szenario folgen später in diesem Abschnitt.
- Über die Formel (4.39) kann die absolute Laufzeit t_1 des Schalls vom ersten Leuchtfeuer zum Empfänger berechnet werden.

$$t_1 = \frac{a^2 - 2c^2(t_{21}^2 - 2t_{51}^2)}{4c^2(t_{21} - 2t_{51})} \tag{8.29}$$

  Aus ihr können über die Differenzen alle übrigen absoluten Laufzeiten berechnet werden.

$$t_i = t_{i1} + t_1 \tag{8.30}$$

Mit den absoluten Laufzeiten stehen dem Programm alle nötigen Informationen zur Berechnung der Koordinaten nach dem bekannten Schema zur Verfügung. Allerdings wird die Genauigkeit prinzipiell nie so gut sein, wie sie es mit Hilfe des Funksignals ist. Dennoch ist es mit relativ wenig Mehraufwand möglich, die Leuchtfeuer sicher zu identifizieren und die absoluten Laufzeiten einigermaßen genau zu ermitteln. Wenn der Einsatz eines Funksignals zu aufwendig oder aus anderen Gründen nicht möglich ist, stellt die beschriebene Technik eine gute Alternative mit brauchbarer Genauigkeit dar.

8.4.3 Durchführung für Szenario 4

Die Rechenschritte für die Ortsbestimmung müssen grundsätzlich immer an ein konkretes Szenario angepaßt werden, unter anderem weil die Leuchtfeuer im tatsächlichen Aufbau anders als in der Vorlage numeriert sind. Die nötigen Schritte sollen hier am Beispiel des Szenarios 4 aus Kapitel 4 vorgestellt werden, dazu kommen die nötigen Anpassungen der Formeln für die Implementierung mit Festkommazahlen.

Das Szenario benutzt die drei Leuchtfeuer L_1, L_2 und L_5. Für den Aufbau wird man die Leuchtfeuer-Module 0, 1 und 2 in genau dieser Reihenfolge entlang der X-Achse aufstellen, um mir kurzen Kabeln

auszukommen und die Steuereinheit am Rand des Feldes plazieren zu können. Tabelle 8.3 faßt alle Änderungen der Benennung zusammen, die sich aus dieser Anordnung ergeben.

Bezeichnung in den Formeln	**Entsprechung im Aufbau**
L_1	Leuchtfeuer 0
L_5	Leuchtfeuer 1
L_2	Leuchtfeuer 2
t_1	t_0
t_5	t_1
t_2	t_2
t_{51}=t_5-t_1	t_{10}=t_1-t_0
t_{21}=t_2-t_1	t_{20}=t_2-t_0

Tabelle 8.3: Bezeichnungen im Aufbau von Szenario 4

Die Aufwertung beginnt damit, daß ein gültiger Satz von drei Meßwerten $rel_0 \ldots rel_2$ aufgenommen wird. Dies bedeutet im Einzelnen, daß die Werte bereits den Leuchtfeuern zugeordnet sind und ihre Summe maximal 2000 µs von der Rundendauer $N \cdot T_A + T_B$ abweicht. Als erstes werden die Werte normiert, wobei unterschiedlich schnell laufende Uhren in Sender und Empfänger ausgeglichen und die Pausen des Sendeschemas abgezogen werden. Dafür muß als Formel (8.24) in eine Form gebracht werden, die sich in Festkommadarstellung überführen läßt.

$$s = \sum_{i=0}^{N-1} rel_i \tag{8.31}$$

$$\begin{aligned} rel_i &:= rel_i \cdot \frac{N \cdot T_A + T_B}{s} && (8.32)\\ &= rel_i \left(\frac{s + N \cdot T_A + T_B - s}{s} \right) \\ &= rel_i \left(1 + \frac{N \cdot T_A + T_B - s}{s} \right) \\ &= rel_i + rel_i \cdot d_q && (8.33) \end{aligned}$$

$$\text{mit} \quad d_q := \frac{N \cdot T_A + T_B - s}{s} \tag{8.34}$$

Mit Hilfe der Konstanten d_q können die Korrekturschritte einfach ausgedrückt werden. Sie gibt den Gangunterschied der Uhren in ppm an und muß daher noch mit einer Million multipliziert werden, damit ausreichend viele Nachkommastellen erhalten bleiben. Der Wert darf sinnvollerweise nur berechnet werden, während der Roboter steht. Anderenfalls überwiegen die Meßfehler durch die Bewegung.

```
dq=(((long)N*TA+TB-((long)rel0+rel1+rel2))*1e6)/((long)rel0+rel1+rel2);

rel0+=((long)rel0*dq)/1e6-TA-TB;
rel1+=((long)rel1*dq)/1e6-TA;
rel2+=((long)rel2*dq)/1e6-TA;
```

Nach den Korrekturen stellen die Werte die folgenden Laufzeitunterschiede dar.

$$rel_0 = t_0 - t_2 \tag{8.35}$$

$$rel_1 = t_1 - t_0 \tag{8.36}$$

$$rel_2 = t_2 - t_1 \tag{8.37}$$

Für die Berechnung von t_1 müssen daraus noch die Werte t_{10} und t_{20} berechnet werden.

$$t_{10} = t_1 - t_0 = rel_1 \tag{8.38}$$
$$t_{20} = t_2 - t_0 = -rel_0 \tag{8.39}$$

Damit sind alle Voraussetzungen für die Berechnung von t_0 zusammen. Der Meßwert rel_2 des dritten Leuchtfeuers geht zwar nicht direkt in die Auswertung ein, er ist aber trotzdem unverzichtbar. Ohne ihn wäre $rel_0 = t_0\text{-}t_1 = \text{-}rel_1$ und es gäbe keine Möglichkeit, t_{20} zu berechnen. Der Weg zu den absoluten Laufzeiten führt über die Formel (8.29), die an die Bezeichnungen des Szenarios angepaßt und für die Festkommadarstellung umgeformt werden muß.

$$t_0 = \frac{a^2 - 2c^2(t_{20}^2 - 2t_{10}^2)}{4c^2(t_{20} - 2t_{10})} \tag{8.40}$$

Durch quadratische Ergänzung wird ein Term aus dem Zähler des Bruches herausgeholt.

$$t_0 = \frac{a^2 - c^2t_{20}^2 - c^2(t_{20} + 2t_{10})(t_{20} - 2t_{10})}{4c^2(t_{20} - 2t_{10})}$$
$$t_0 = \frac{a^2 - c^2t_{20}^2}{4c^2(t_{20} - 2t_{10})} - \frac{1}{4}(t_{20} + 2t_{10})$$

Der verbleibende Zähler muß über die dritte binomische Formel in zwei Terme aufgespalten werden, weil der Wert des Ausdrucks sonst zu groß werden kann.

$$t_0 = \frac{1}{4}\left(\frac{\left(\frac{a}{c} + t_{20}\right) \cdot \left(\frac{a}{c} - t_{20}\right)}{t_{20} - 2t_{10}} - t_{20} - 2t_{10}\right)$$

Die Größen in der Formel werden in das Einheitensystem der Festkommadarstellung transformiert.

$$10^6 t_0 = \frac{1}{4}\left(\frac{10^6\left(\frac{a}{c} + t_{20}\right) \cdot 10^6\left(\frac{a}{c} - t_{20}\right)}{10^6(t_{20} - 2t_{10})} - 10^6 t_{20} - 2(10^6 t_{10})\right)$$
$$10^6 t_0 = \frac{1}{4}\left(\frac{\left(10^4\frac{10^3 a}{10c} + 10^6 t_{20}\right) \cdot \left(10^4\frac{10^3 a}{10c} - 10^6 t_{20}\right)}{10^6 t_{20} - 2(10^6 t_{10})} - 10^6 t_{20} - 2(10^6 t_{10})\right)$$
$$\overline{t_0} = \frac{1}{4}\left(\frac{\left(10000\frac{\overline{a}}{\overline{c}} + \overline{t_{20}}\right) \cdot \left(10000\frac{\overline{a}}{\overline{c}} - \overline{t_{20}}\right)}{\overline{t_{20}} - 2\overline{t_{10}}} - \overline{t_{20}} - 2\overline{t_{10}}\right) \tag{8.41}$$

Die anderen absoluten Laufzeiten können einfach aus t_0 bestimmt werden. Die Koordinaten ergeben sich daraus nach dem bekannten Schema.

```
t10 = rel1;
t20 = -rel0;
t0 = (((((((long)10000*a)/c)+t20)*((((long)10000*a)/c)-t20))
     /((long)t20-2*t10))-t20-2*t10)/4;
t1 = t10+t0;
t2 = t20+t0;

da = ((((long)2*a*100000)/c)*1000)/c;

x = a/2+(sqr((long)t0)-sqr((long)t2))/da;
y = ((long)sqrt32(10*c*((long)t0)+(long)x*100000)
          *sqrt32(10*c*((long)t0)-(long)x*100000))/100000;
```

8.4.4 Durchführung für Szenario 6

Die Genauigkeit der Koordinatenberechnung in Szenario 4 kann durch den Einsatz eines weiteren Leuchtfeuers gesteigert werden. Insbesondere wird der Fehler in Y-Richtung dadurch geringer, allerdings dauert eine Runde des Ortungssystems etwas länger. Das zusätzliche Leuchtfeuer wird in der oberen rechten Ecke des Feldes aufgestellt und so verkabelt, daß die Nummern der anderen Leuchtfeuer unverändert bleiben. Tabelle 8.4 faßt die Änderungen zusammen.

Bezeichnung in den Formeln	**Entsprechung im Aufbau**
L_1	Leuchtfeuer 0
L_5	Leuchtfeuer 1
L_2	Leuchtfeuer 2
L_3	Leuchtfeuer 3
t_1	t_0
t_5	t_1
t_2	t_2
t_3	t_3
t_{51}=t_5-t_1	t_{10}=t_1-t_0
t_{21}=t_2-t_1	t_{20}=t_2-t_0
t_{31}=t_3-t_1	t_{30}=t_3-t_0

Tabelle 8.4: Bezeichnungen im Aufbau von Szenario 6

Die Berechnungen laufen exakt wie bei Szenario 4 ab, es müssen nur die Meßwerte etwas anders umgerechnet werden. Von den Sensoren kommen die folgenden Laufzeitunterschiede.

$$rel_0 = t_0 - t_3 \tag{8.42}$$
$$rel_1 = t_1 - t_0 \tag{8.43}$$
$$rel_2 = t_2 - t_1 \tag{8.44}$$
$$rel_3 = t_3 - t_2 \tag{8.45}$$

Für die Berechnung der absoluten Laufzeiten werden sie in eine andere Darstellung umgerechnet.

$$t_{10} = t_1 - t_0 = rel_1 \tag{8.46}$$
$$t_{20} = t_2 - t_0 = rel_2 + rel_1 \tag{8.47}$$
$$t_{30} = t_3 - t_0 = rel_3 + rel_2 + rel_1 \tag{8.48}$$

Der komplette C-Code zur Berechnung der Koordinaten lautet dann wie folgt.

```
t10 = rel1;
t20 = rel2+rel1;
t30 = rel3+rel2+rel1;
t0 = (((((((long)10000*a)/c)+t20)*((((long)10000*a)/c)-t20))
     /((long)t20-2*t10))-t20-2*t10)/4;
t1 = t10+t0;
t2 = t20+t0;
t3 = t30+t0;
da = ((((long)2*a*100000)/c)*1000)/c;
db = ((((long)2*b*100000)/c)*1000)/c;
x = a/2+(sqr((long)t0)-sqr((long)t2))/da;
y = b/2+(sqr((long)t2)-sqr((long)t3))/da;
```

8.5 Anforderungen an das Sendetiming

Für die Ortung ist es wünschenswert, daß die Leuchtfeuer in einer möglichst schnellen Folge senden. Dadurch wird der Roboter in kurzen Intervallen mit neuen Positionsangaben versorgt, außerdem kann er sich während eines Zyklus nur sehr wenig bewegen, was die Meßfehler klein hält. Auf der anderen Seite gibt es aber Fakten, welche die erreichbare Geschwindigkeit begrenzen.

8.5.1 Anforderungen an T_A

Die Steuereinheit muß zwischen zwei Impulsen lange genug warten, daß das Echo des ersten Impulses genug abklingen kann. Nur so ist der Detektor in der Lage, die beiden Impulse auseinanderzuhalten. Außerdem muß ihm noch ausreichend Zeit für die Auswertung der Daten und die Übertragung zum RCX bleiben. Im Einzelnen laufen beim Senden eines Impulses die folgenden Schritte ab.

Dauer	Aktion des Empfängers
5 ms	Das Funksignal mit der Identifikation des Senders wird zum Empfänger gesendet. Dabei werden 10 Bit mit einer Datenrate von 2 kBit/s übertragen.
etwa 45 ms	Innerhalb dieser Zeit ist der Schall im Raum nachweisbar, der Impuls erreicht den Empfänger und löst den Detektor aus. Wie lang die Zeit genau ist, hängt von den akustischen Eigenschaften des Raumes ab. Das Minimum liegt bei 30 ms, nach oben sind Werte bis 75 ms beobachtet worden.
10 ms	Der Detektor erwartet, daß nach dem Abklingen des Echos 10 ms lang keinerlei Impulse registriert werden. Dementsprechend wartet er so lang, bis dies der Fall ist. Dadurch wird sichergestellt, daß zwei aufeinanderfolgende Übertragungen sich nicht überlappen können.
15 ms	Schließlich müssen die gesammelten Daten noch an den RCX übertragen werden. Abhängig von der Abfrageschleife von brickOS dauert dies einige Millisekunden, bis alle 90 Bit im seriell codierten Protokoll angekommen sind.
75 ms	**Gesamtdauer**

Tabelle 8.5: Zeitlicher Ablauf einer Entfernungsmessung per Ultraschall

Die Angaben sind nur als grobe Richtwerte zu sehen, in der Praxis unterliegen sie deutlichen Schwankungen und müssen experimentell ermittelt werden. Dazu kann insbesondere die Echodauer beitragen, die der Sensor mißt und als Meßwert an die Anwendung weitergibt. In der Praxis sind Impulsfolgen bis herunter auf T_A=80 ms erfolgreich getestet worden. Die Messung der Echodauer kann im Programmcode des Empfängers deaktiviert werden, so daß die Datenübertragung schon beginnt, während das Echo noch nachweisbar ist. Auf diese Weise ist in einem geeigneten Raum auch ein Sendeschema mit T_A=70 ms möglich. Bei den genannten Zahlen handelt es sich aber um Untergrenzen, der tatsächliche Wert von T_A sollte zur Sicherheit etwas größer sein.

8.5.2 Anforderungen an T_B

Die zusätzliche Pause T_B am Ende einer Runde wird nur gebraucht, wenn der Empfänger die Sender auch ohne Hilfe des Funksignals identifizieren soll. Anderenfalls muß die Steuereinheit keine Extrapause einlegen, entsprechend kann die Zeit T_B problemlos auf 0 gesetzt werden.

Die Sender können ohne Funksignal nach einem einfachen Kriterium unterschieden werden. Zwischen dem Feuern des letzten und ersten Leuchtfeuers vergeht die Zeit T_A+T_B, bei den anderen ist es nur T_A. Als Folge ist auch der gemessene Laufzeitunterschied zum Vorgänger am größten, wenn der Impuls von Leuchtfeuer 0 kommt. Die Meßwerte verschieben sich allerdings abhängig von der Position des Roboters in gewissen Grenzen. Daher muß T_B groß genug gewählt werden, damit das Kriterium auch unter diesen Bedingungen noch funktioniert

Innerhalb eines quadratischen Feldes der Seitenlänge a ist der größte mögliche Abstand zwischen Sender und Empfänger $a\sqrt{2}$. Dementsprechend können die folgenden absoluten Laufzeiten auftreten.

$$0 \;\leq\; t_i \;\leq\; M \quad \text{mit } M = \frac{a}{c}\sqrt{2} \tag{8.49}$$

Der Faktor M begrenzt auch die auftretenden Laufzeitdifferenzen t_{ij}=t_i-t_j.

$$-M \;\leq\; t_{ij} \;\leq\; M \tag{8.50}$$

Wenn zwei Leuchtfeuer im Abstand T_A senden, wird der Empfänger Laufzeitunterschiede zwischen T_A-M und T_A+M feststellen. Beim Sendeabstand T_A+T_B liegt der Meßwert zwischen T_A+T_B-M und T_A+T_B+M. Damit Leuchtfeuer 0 sicher erkannt werden kann, müssen die Intervalle disjunkt sein.

$$\begin{aligned} T_a + T_b - M &> T_a + M \\ T_b &> 2M \\ T_b &> a \cdot \frac{2\sqrt{2}}{c} \\ T_b &> a \cdot 10\frac{\text{ms}}{\text{m}} \end{aligned}$$

Wenn das Ortungssystem also ohne das Funksignal arbeiten soll, muß T_B mindestens 10 ms pro Meter Kantenlänge des abzudeckenden Feldes betragen.

8.6 Berücksichtigung der Bewegung

Das Ortungssystem funktioniert auch problemlos, während der Roboter sich bewegt, allerdings leidet die Genauigkeit darunter. Der Empfänger wird die nötigen Impulse an etwas unterschiedlichen Orten empfangen, was die Formeln aber nicht berücksichtigen. Der Fehler ist um so größer, je schneller der Roboter sich bewegt und je länger eine Runde des Sendeschemas dauert. Außerdem wirkt er sich beim Verzicht auf das Funksignal deutlich stärker aus, als wenn das System mit absoluten Laufzeiten arbeitet. Der Benutzer hat einen gewissen Einfluß auf die Genauigkeit. Er kann den Roboter langsamer fahren oder sogar anhalten lassen, wenn besonders präzise Koordinaten benötigt werden. Eine andere Lösung kann sein, alle Meßwerte durch Extrapolation auf einen gemeinsamen Stand zu bringen.

Umgekehrt kann es für das Programm auch interessant sein, in welche Richtung und wie schnell der Roboter sich bewegt. Dazu wird im einfachsten Fall der Bewegungsvektor zwischen zwei Positionen berechnet. Wenn die Orientierung auch im Stand festgestellt werden soll, können zwei Empfänger an entgegengesetzten Enden des Roboters aufgestellt und die Differenz ihrer Koordinaten ermittelt werden.

8.7 Beispielprogramme

Eine Reihe von Beispielprogrammen soll dabei helfen, die Funktionen des Systems zu testen, Meßwerte zu kontrollieren und die Implementierung der Koordinatenberechnung in einem funktionierenden Kontext zu zeigen.

Ihr Quellcode befindet sich auf der beiliegenden CD, die folgende Liste zählt diese Programme auf und beschreibt kurz ihre Funktion.

- **skeleton.c** enthält das Grundgerüst mit Mutex-Mechanismus zur Ansteuerung des Sensors. Es ist als Grundlage für die Entwicklung eigener Programme gedacht, als Anwendungsbeispiel wird die gemessene Entfernung zum Sender 0 auf dem Display angezeigt.
- **monitor.c** ist ein vielseitig einsetzbares Programm zur Analyse des Systems und der Meßwerte. Wenn gestartet ist, zeigt es die Meßwerte eines angeschlossenen Empfängers an. Mit der PRGM-Taste kann ausgewählt werden, ob die Informationen von beliebigen oder einem bestimmten Sender angezeigt werden sollen, VIEW wechselt zwischen den verschiedenen Feldern. Die Anzeige wird jedesmal aktualisiert, wenn neue Daten eintreffen, die den Filterkriterien entsprechen.
- **temperature.c** erlaubt die Bestimmung der Schallgeschwindigkeit über den Temperatursensor von Lego. Die Anzeige kann mit VIEW zwischen der Temperatur und der Schallgeschwindigkeit umgeschaltet werden.
- **calibrate.c** implementiert den Algorithmus zur Messung der Schallgeschwindigkeit mit Hilfe von zwei Leuchtfeuern mit bekanntem Abstand. Nach der Messung von jeweils 8 Werten gibt das Programm wahlweise die Schallgeschwindigkeit oder die Temperatur auf dem LCD aus, wobei die Umschaltung wieder über VIEW erfolgt.
- **possend.c und posrecv.c** sind ein Gespann von Programmen, mit denen die Meßwerte des Empfängers per Infrarot an einen PC zur Auswertung gesendet werden können. Dazu läuft possend auf dem RCX und posrecv auf dem PC, die Geschwindigkeit reicht aus, um alle 150 ms einen Impuls zu übertragen. Die Daten können in zwei verschiedenen Formaten angezeigt werden, die erste stellt alle Daten dar, wobei eine Zeile einem Paket entspricht.

  ```
  BCO 0  |  ABS     3303 us  |  REL  1100563 us  |  ECHO    63 ms  |  EUR
  BCO 1  |  ABS     2797 us  |  REL   999555 us  |  ECHO    61 ms  |  EUR
  --- -  |  ABS -------- us  |  REL   266923 us  |  ECHO   118 ms  |  EU.
  ```

  Alternativ kann auch ein Modus gewählt werden, in dem nur die absoluten Laufzeiten angezeigt werden. Auch hier entspricht jede Zeile einem Meßwert, jede der vier Spalten enthält die Daten eines Leuchtfeuers.

  ```
  3302 us  |                |                |
           |       2804 us  |                |
  ```

- **coordinates.c** implementiert die Koordinatenberechnung mit Hilfe des Funksignals nach den Szenarien 1-3. Mit der PRGM-Taste kann ein Szenario ausgewählt werden, VIEW schaltet zwischen der Anzeige von X-Koordinate, Y-Koordinate und Schallgeschwindigkeit um.
- **rcxbeacon.c und tworecv.c** demonstrieren die Koordinatenberechnung mit einem mobilen Sender. Der RCX mit dem Programm rcxbeacon steuert den Sender an und sendet regelmäßig Impulse aus. Die Auswertung wird von tworecv auf einem anderen RCX übernommen. Dort müssen zwei Empfänger mit definiertem Abstand angeschlossen werden, damit die Berechnung funktionieren kann. Mit VIEW kann ausgewählt werden, ob X oder Y angezeigt werden soll.
- **relative.c** zeigt, wie die Koordinatenberechnung auch ohne das Funksignal durchgeführt werden kann. Es ist eine Umsetzung von Szenario 4, die Koordinaten werden im LCD dargestellt, mit VIEW kann zwischen beiden umgeschaltet werden.

Kapitel 9

Ergebnisse

Die Entwicklung des vorliegenden Ortungssystems hat sich als sehr aufwendig herausgestellt. Es gab viele Probleme von der Konzeption über die Konstruktion bis hin zur Programmierung, deren Lösung oft mit einigem Aufwand verbunden war. Daher stellt sich natürlich die Frage, ob es eine weniger komplizierte Alternative gegeben hätte.

Am naheliegendsten ist es, einen Ultraschallsensor in der Art des Geräuschsensors aus Anhang A.2 zu bauen. Der RCX könnte dann nur feststellen, ob der Sensor zum Abfragezeitpunkt gerade Ultraschall empfängt oder nicht. Zusätzliche Funksignale werden nicht benutzt, die Messung und Auswertung der Laufzeitunterschiede erfolgt unter brickOS. Diese Architektur ist sehr einfach, die Fehlersimulation zeigt allerdings, daß die erreichbare Genauigkeit katastrophal ist. Durch die relativ selten stattfindenden Auslesevorgänge liegen die Fehler in der Koordinatenberechnung im Bereich von einigen Dezimetern. Es führt also kein Weg an einer Vorverarbeitung der Meßdaten im Sensor vorbei. Damit alleine ist es aber auch nicht getan, wie das Ortungssystem von [Restena] zeigt. Die eingangs angeführte Abbildung 1.5 belegt deutlich, daß die Fehler des Systems ebenfalls bei mehreren Dezimetern liegen müssen, die Webseite des Projektes gibt 20 cm als typische Größe an. Auch dieser Wert ist noch zu schlecht, um das System in der Praxis sinnvoll einsetzen zu können. Erst die seriellen Datenübertragung der vorverarbeiteten Meßwerte in den RCX bringt die nötige Verbesserung. Der Empfänger darf also kaum einfacher aufgebaut sein als er es momentan ist.

Das gleiche gilt für das Sendernetzwerk. Die Leuchtfeuer stellen bereits eine Minimallösung dar, und die Steuereinheit kann nur vereinfacht werden, indem die Variationsmöglichkeiten entfernt werden. Dann müßte das System mit einer festen Anzahl von Leuchtfeuern und einem statischen Timing arbeiten, das langsam genug ist, um auch in Räumen mit stärkerem Echo noch zu funktionieren. Diese Einschränkungen wären aber ein hoher Preis, denen eine geringe Ersparnis an Schaltungsaufwand und Programmkomplexität gegenübersteht.

Denkbar wäre noch ein Verzicht auf das zusätzliche Funksignal. Die Tests mit dem fertigen System haben gezeigt, daß die so erhaltenen Koordinaten eine in vielen Fällen ausreichende Genauigkeit erreichen. Dies war am Anfang des Projektes jedoch nicht absehbar, die Hochrechnungen gingen damals in eine andere Richtung. Andererseits ist die Funkübertragung als Ergänzung des Systems sehr hilfreich, wenn es auf eine hohe Präzision ankommt. Durch den Einsatz fertiger Module ist der Aufwand für diesen zusätzlichen Kanal so gering, daß er vertretbar ist.

9.1 Technische Eckdaten

Es sind im Wesentlichen drei Parameter entscheidend für die Beurteilung des Ortungssystems. Die Reichweite entscheidet darüber, was für ein Gebiet maximal abgedeckt werden kann. Von besonderem

Interesse ist die Genauigkeit, mit der Entfernungen gemessen und die Koordinaten berechnet werden können. Die Geschwindigkeit gibt schließlich an, wie lange eine Ortsbestimmung dauert. Dies hat unter anderem einen Einfluß auf die Genauigkeit, wenn der Roboter sich während einer Messung bewegt.

9.1.1 Reichweite

Sender und Empfänger des Ortungssystems sind so aufeinander abgestimmt, daß sich Ultraschallimpulse noch in einer Entfernung von 10 m zuverlässig nachweisen lassen. Darin sind die Verluste durch Streuung an den Reflektoren bereits berücksichtigt, so daß effektiv ein quadratisches Feld mit einer Kantenlänge von 7 Metern komplett abgedeckt werden kann. Abhängig von den Umgebungsbedingungen sind auch etwas größere Distanzen möglich, dies hängt in erster Linie von dem Funksignal ab, da die Reserven der Ultraschallstrecke etwas größer sind.

9.1.2 Geschwindigkeit

Wie schnell ein Roboter geortet werden kann, hängt von der Anzahl der verwendeten Leuchtfeuer, von dem ausgewählten Sendetiming und auch ein wenig von der Akustik des Raumes ab. Bei zwei Leuchtfeuern und dem schnellsten Timing sind etwas über 6 Ortungen pro Sekunde möglich. Soll die Ortung ohne Auswertung des Funksignals erfolgen, ist die schnellste Lösung ein Szenario mit 3 Leuchtfeuern und dem Zeitschema 110+20 ms. Damit sind fast 3 Messungen pro Sekunde möglich.

9.1.3 Genauigkeit

Die Abbildung 9.1 gibt einen ersten Eindruck davon, wie genau das Ortungssystem Entfernungen messen kann. Im Diagramm sind die Meßwerte (blaue Kurve, linke Achse) und der jeweilige Meßfehler (rote Kurve, rechte Achse) für Entfernungen zwischen 0 und 2 Metern aufgeführt.

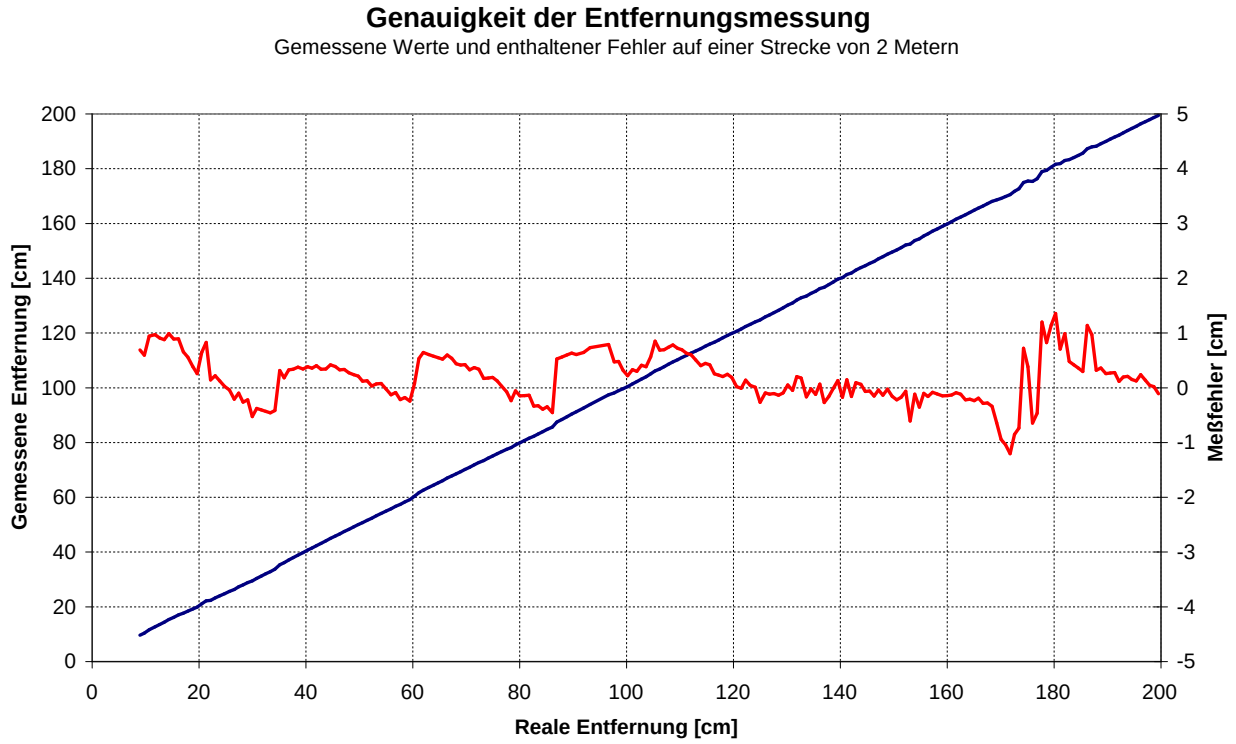


Abbildung 9.1: Genauigkeit des Ortungssystems beim Messen von Entfernungen

Von der Störung bei 180 cm abgesehen liegt die Abweichung bei maximal einem Zentimeter, was etwa der Wellenlänge des Ultraschalls entspricht. Deutlich bessere Ergebnisse lassen sich mit der eingesetzten Elektronik nicht erreichen. Die Genauigkeit reicht aber vollkommen für die verschiedenen Berechnungen aus, wie ein kleines Experiment zeigt. Mit einem in Abschnitt 8.3 beschriebenen Aufbau kann das Ortungssystem benutzt werden, um die Schallgeschwindigkeit und damit auch die Lufttemperatur zu messen. Bei mehreren Versuchsmessungen konnte die Temperatur immer auf 1°C genau bestimmt werden, einige davon fanden im Freien bei Temperaturen um den Gefrierpunkt statt.

Aus den gemessenen Entfernungen und Laufzeitunterschieden können die Koordinaten des Empfängers berechnet werden. Bei ihrer Beurteilung werden zwei verschiedene Genauigkeitsbegriffe benutzt.

- Die **absolute Genauigkeit** gibt an, wie weit die berechneten Koordinaten mit der Position im Bezugssystem übereinstimmen. Ihr Wert ist vor allem interessant, wenn der Roboter einen bestimmten Punkt ansteuern soll oder seine Daten von außen ausgewertet werden sollen.
- Dagegen beschreibt die **relative Genauigkeit**, wie stabil Koordinaten eines festen Punktes bei mehreren Messungen sind und wie genau kleinere Unterschiede in der Position erkannt werden können. Dieser Wert ist vor allem für die Navigation im Nahbereich, das Messen von Geschwindigkeiten und Toleranzgrenzen in den Algorithmen wichtig. Aus einer hohen absoluten folgt immer eine hohe relative Genauigkeit, umgekehrt kann es trotz einer niedrigen absoluten eine brauchbare relative Genauigkeit geben.

Die folgende Auswertung konzentriert sich daher auf den absoluten Fehler. Dazu wurde ein quadratisches Feld mit einer Kantenlänge von 2 m abgesteckt, in dem der Empfänger sich auf einer Kreisbahn bewegen kann. Aus den vom RCX berechneten Positionen ist die Abbildung 9.2 entstanden. Zugrunde liegt das Szenario 2, das 3 Leuchtfeuer benötigt und das Funksignal zur Auswertung benutzt. Mit Fehlern in der Entfernungsmessung von 1 cm können die Koordinaten laut der Simulation bis zu 5 cm abweichen. Der tatsächliche Meßfehler ist jedoch beeindruckend gering.

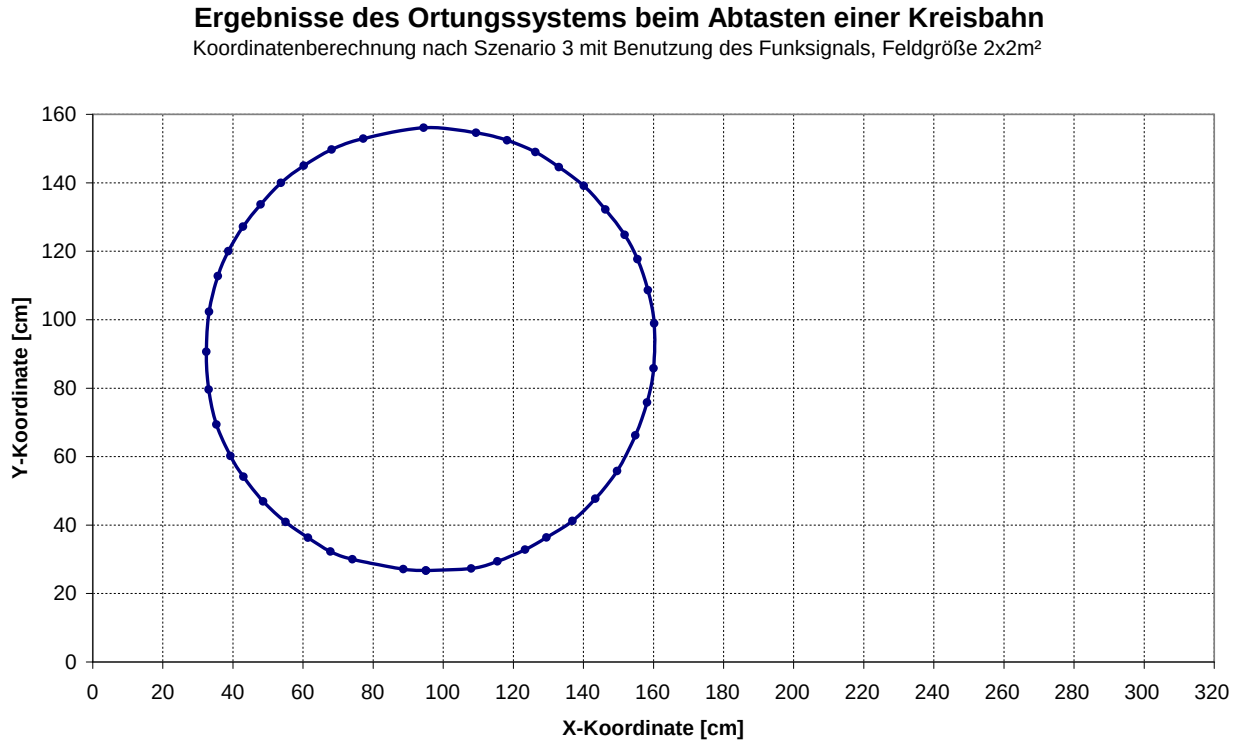


Abbildung 9.2: Koordinatenberechnung beim Zurücklegen einer Kreisbahn (mit Funksignal)

Die Ergebnisse ohne Benutzung des Funksignals sind nicht ganz so gut, aber immer noch ausreichend. Die Ergebnisse in Abbildung 9.3 wurden basierend auf Szenario 4 berechnet, dabei kamen 3 Leuchtfeuer zum Einsatz. Das Szenario wurde bewußt ausgewählt, da es etwas größere Abweichungen in Y-Richtung aufweist, vor allem in der oberen Hälfte des Feldes. Der rote Kreis stellt die exakte Kreisbahn dar.

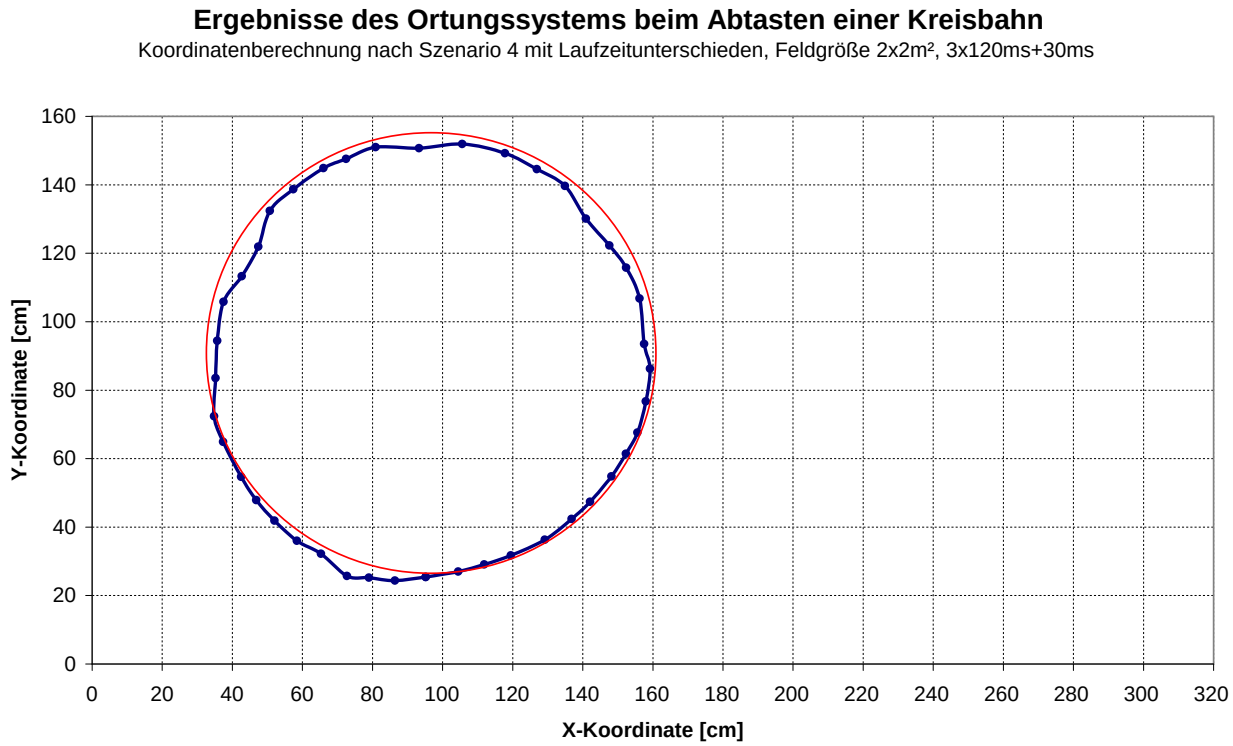


Abbildung 9.3: Genauigkeit der Koordinatenberechnung ohne Funksignal

9.2 Einsatz in den Übungen

Das Ortungssystem mußte seinen erste Bewährungsprobe im Rahmen der Übungen am Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme bestehen. Die Studenten hatten die Aufgabe zu lösen, ihre Roboter durch einen auf dem Boden aufgezeichneten Parcours zu bringen.

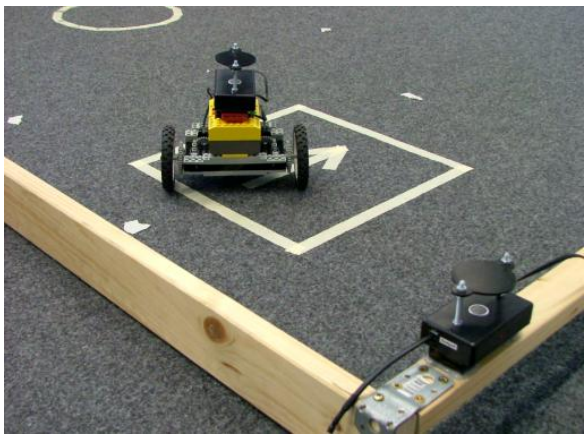


Abbildung 9.4: Aufbau des Ortungssystems für die Übungen

In Abbildung 9.4 ist die Startposition zu sehen, ihre Koordinaten und die der verschiedenen Hindernisse waren fest in den Programmen vorgegeben und mußten entsprechend der Regeln abgefahren werden. Das Ortungssystem hat den Test mit Erfolg bestanden, alle Geräte haben funktioniert und es gab keine technischen Probleme, die auf das System zurückzuführen waren.

Abbildung 9.5: Einsatz des Ortungsystems in den Übungen

In einer anschließenden Umfrage konnten die Studenten Feedback geben und haben sich durchweg sehr positiv geäußert. Einige Anregungen und Verbesserungsvorschläge sind noch unmittelbar vor der Fertigstellung in die Arbeit eingeflossen.

- Die größten Schwierigkeiten gab es beim **Ausrechnen der Koordinaten**. Alle Übungsteilnehmer haben die Formeln mit Fließkommazahlen implementiert und hatten dabei starke Probleme mit Überläufen, Auslöschung und ähnlichen numerischen Fehlern. Die häufig verwendete Lösung, die Meßwerte durch Division zu verkleinern, führte dabei zwangsläufig zu einer reduzierten Genauigkeit. Daraufhin wurden bessere Algorithmen mit Festkommaarithmetik implementiert und mit in die Ausarbeitung aufgenommen, so daß sie späteren Gruppen zur Verfügung stehen.
- Auf den Wunsch einer Gruppe wurde das Timing von Empfänger und Steuereinheit optimiert, so daß etwas **schnellere Feuersequenzen** der Leuchtfeuer möglich wurden. Auf diesem Weg wurde die Grenze von 100 ms auf 70-80 ms gesenkt.
- Für Testzwecke haben sich mehrere Gruppen die Möglichkeit gewünscht, **Debug-Ausgaben** per Infrarot an einen PC zur Auswertung schicken zu können. Mit der lnphost-Bibliothek besteht jetzt diese Möglichkeit, entsprechende Beispielprogramme sind ebenfalls vorhanden.
- In der ersten Version war der **Vibrationsdämpfer** nicht robust genug und ist im Verlauf der Übungen kaputt gegangen. Der Kern aus Filz wurde gegen Neopren getauscht und widersteht jetzt auch den Belastungen beim ruckartigen Auseinanderziehen der Teile.

Mit diesen Verbesserungen kann das System problemlos in weiteren Übungen eingesetzt werden.

9.3 Mögliche Verbesserungen

Die Liste der bekannten Schwachstellen und Verbesserungsmöglichkeiten ist relativ kurz, da viele Punkte noch während der Entwicklungszeit umgesetzt werden konnten. Das größte Potential für Verbesserungen besteht noch bei dem Ultraschalldetektor und der Funkübertragung. Bei beiden können die Empfindlichkeit, die Unterdrückung von Störungen sowie die Auslösegenauigkeit noch verbessert werden, wofür dann aber aufwendige Schaltungstechnik gebraucht wird. Dies ist vor allem bei den Funkmodulen nötig, da sie die größte Schwachstelle des Systems bilden.

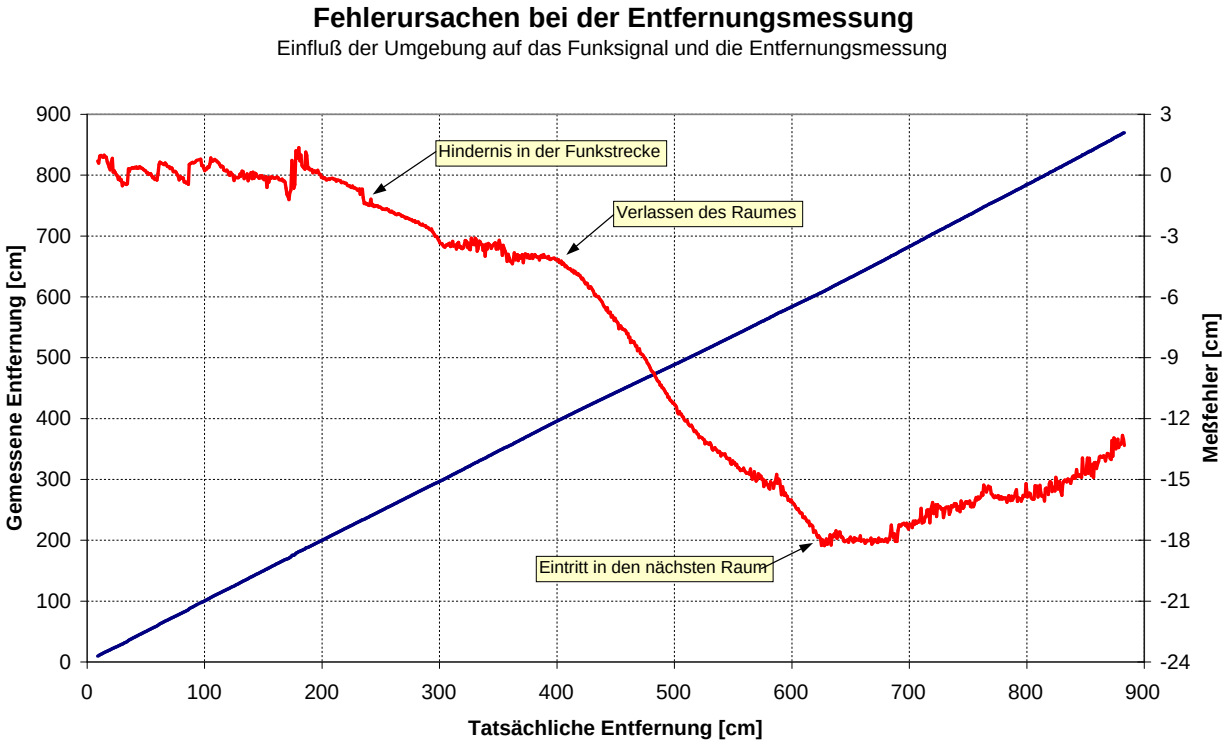


Abbildung 9.6: Fehleranalyse für die Entfernungsmessung

Abbildung 9.6 zeigt die Ergebnisse einer Entfernungsmessung mit einem Leuchtfeuer und einem Empfänger, die genau diese Behauptung belegt. Dargestellt sind die gemessene Entfernung und der dabei auftretende Fehler. Es sind die selben Daten, die auch Abbildung 9.1 zugrunde liegen, hier sind aber alle Entfernungen bis 9 Meter dargestellt. Es fällt auf, daß der Fehler lokal kaum schwankt und alle Werte in einem engen Schlauch liegen. Noch interessanter ist aber die Tatsache, daß die Entfernung ab einer Entfernung von 200 cm zu kurz gemessen wird. Dafür gibt es nur zwei Erklärungen, entweder hat der Ultraschalldetektor ausgelöst, bevor der Impuls den Empfänger erreicht hat, oder der Funkdetektor hat zu spät ausgelöst. Es kommt natürlich nur die zweite Alternative in Frage, und der Kurvenverlauf kann auch belegen, daß die Ursache beim Funksignal liegen muß.

Die Meßreihe ist entstanden, als der Roboter mit dem Empfänger quer durch eine Wohnung gefahren ist und sich dabei vom Sender entfernt hat. Zwischen beiden bestand Sichtkontakt, so daß der direkte Schallweg immer frei war. Die Steuereinheit mit dem Funksender stand hingegen etwas abseits, so daß sich nacheinander ein Möbelstück und schließlich die Wände der Wohnung zwischen ihn und den Empfänger geschoben haben. Diese markanten Hindernisse finden sich im Diagramm an den zu erwartenden Stellen wieder, die Punkte sind entsprechend beschriftet. Mittlerweile gibt es deutlich bessere Funkmodule zu kaufen, die vergleichbar sind und diesen Fehler nicht aufweisen. Glücklicherweise stört der Fehler nur wenig, wenn keine Hindernisse in der Funkstrecke sind.

9.4 Fazit

Im Laufe dieser Arbeit ist ein Ortungssystem für Lego Mindstorms entstanden, das alle in Kapitel 5 genannten Anforderungen erfüllt. Es hat eine Reichweite von 10 Metern und kann Entfernungen auf einen Zentimeter genau messen. Die daraus resultierenden Koordinaten weichen in der Praxis deutlich weniger als 5 cm von den tatsächlichen Werten ab, damit kann das System mit kommerziellen Geräten konkurrieren. Bis zu 6 Positionen pro Sekunde erlauben den Einsatz auf einem mobilen Roboter während der Fahrt und lassen schnelle Reaktionen zu.

Die einzelnen Geräte des Systems sind robust und anpassungsfähig, das modulare Konzept erlaubt es, viele verschiedene Szenarien auszuprobieren. Damit eignet sich das System sehr gut für den Einsatz in Übungen, Praktika, für Demonstrationen oder Präsentationen. Die Hardware ist relativ einfach aufgebaut und besteht fast ausschließlich aus Standardbauteilen, die einfach zu beschaffen sind. Reparaturen und der Nachbau einzelner Komponenten sind damit problemlos möglich, die nötigen Informationen sind in dieser Ausarbeitung enthalten.

Der Stromverbrauch der einzelnen Module ist sehr gering und orientiert sich an dem Einsatz auf einem batteriebetriebenen Mindstorms-Roboter. Alle Schaltungen verbrauchen weniger als 20 mA, der Empfänger liegt sogar deutlich darunter. Die zum RCX übertragenen Meßwerte sind umfangreich genug, um die Grundlage für komplexe Algorithmen zur Filterung und Auswertung zu bilden. Die Technik der seriellen Datenübertragung über das Sensorinterface ist neu. Einige Entwickler haben sich mit den Einschränkungen der Schnittstelle beschäftigt aber keine Lösung gefunden, die mit so geringem Aufwand eine vergleichbare Bandbreite oder Zuverlässigkeit erreicht.

Diese Dokumentation beschäftigt sich neben dem Ortungssystem auch mit vielen Grundlagen zum Thema Mindstorms. Die enthaltenen Informationen können für andere Entwickler eine gute Hilfe sein, die Sammlung ist eine der vollständigsten ihrer Art. Dazu gehört auch die Bibliothek lnphost, die mittlerweile bei Sourceforge gehostet ist (siehe [lnphost]). Sie wird von einer Reihe von Leuten und Einrichtungen mit Erfolg als Alternative zum Lego Network Protocol Daemon benutzt. Genauso sind einige im Rahmen dieser Arbeit entstandene Fehlerkorrekturen und Erweiterungen in brickOS selbst eingeflossen.

Damit wird ein guter Teil der Ergebnisse aus der Arbeit auch nach ihrem Abschluß noch aktiv genutzt, was mich als Entwickler sehr freut. Ich selbst habe während der Bearbeitung des Themas eine Menge an Erfahrungen, Wissen und Fertigkeiten gesammelt. Es war eine schöne Gelegenheit, Gelerntes auszuprobieren und gleichzeitig an der Arbeit zu wachsen, und ich bin mit dem Ergebnis sehr zufrieden.

Teil III

Anhänge

Anhang A

Verwendete Schaltungen

Die folgenden Abschnitte enthalten Bauanleitungen für alle Platinen und Zubehörteile, die in dieser Arbeit vorgestellt wurden. Dazu gehören neben Schaltplänen auch Bauteillisten, das Platinenlayout mit passendem Bestückungsplan und Hinweise zum Aufbau.

A.1 Grundschaltung für Sensoren

Beim Experimentieren mit Mindstorms-Sensoren ist es hilfreich, die Grundschaltung auf einer eigenen Platine aufzubauen. Sie liegt dann als fertiges Modul vor, das beispielweise in eine Steckplatine gesteckt werden kann und so die Basis für einen Sensorprototypen bildet.

Abbildung A.1: Allgemeines Sensorinterface

Die Schaltung für dieses Modul entspricht der Grundschaltung, in die ein Kondensator von 22 µF eingebaut wurde. Diese Kapazität ist für alle denkbaren Sensoren mit einem Stromverbrauch bis 11 mA ausreichend, da die Spannung während der Auslesephase um maximal 50 mV einbrechen kann.

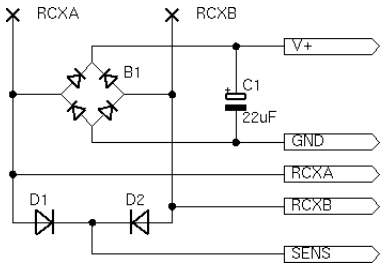


Abbildung A.2: Schaltung des allgemeinen Sensorinterfaces

Der Pfostenstecker auf der Platine liefert alle nötigen Anschlüsse. Die Ausgangsspannung des RCX liegt direkt an den Anschlüssen RCXA und RCXB an. Gleichgerichtet und geglättet dient sie an V+

und GND dazu, den Sensor mit Strom zu versorgen. Der auszulesende Teil des Sensors wird zwischen die Anschlüsse SENS und GND geschaltet.

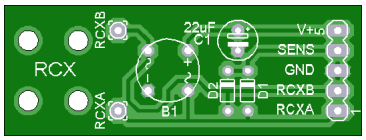


Abbildung A.3: Layout des allgemeinen Sensorinterfaces

Beim Aufbau der Platine ist nur die Kabelführung zu berücksichtigen, die anderen Bauteile werden einfach eingelötet. Die beiden Adern des Kabels werden am Ende aufgetrennt und durch die beiden oberen bzw. durch unteren Löcher geführt. Diese Konstruktion dient als Zugentlastung und verhindert, daß das Kabel einfach aus der Platine gerissen werden kann.

	Halbleiter
2x	Diode 1N4001
1x	Brückengleichrichter B40C800

	Kondensatoren
1x	22 µF, 10 V, Elektrolyt oder Tantal

	Pfostenstecker 2.54 mm, einreihig
1x	5 Stifte abgewinkelt

	Verschiedenes
1x	Fotoplatine in passender Größe, einseitig
1x	Halbes Mindstorms-Kabel

Tabelle A.1: Komponentenliste für das allgemeine Sensorinterface

Die Schaltung kann auch als Basis für Aktoren benutzt werden, sofern diese nur wenige Milliampere an Strom benötigen. Anderenfalls muß ein Elko als Ergänzung zu C_1 extern angeschlossen werden.

A.2 Mindstorms Schallsensor

Der hier vorgestellte Schallsensor ist ein Beispiel für einen einfachen aktiven Mindstorms-Sensor. Mit ihm kann der RCX auf laute Geräusche wie Klatschen reagieren. Die Schaltung stammt aus [Baum] und ist klein genug, daß sie in einen Legostein eingebaut werden kann.

Abbildung A.4: Mindstorms Schallsensor

Das vom Mikrofon kommende Signal wird zunächst von einem Operationsverstärker um den Faktor 45 verstärkt, bevor es einem Detektor zugeführt wird. Die Diode D_3 leitet die positiven Halbwellen des Signals weiter und lädt so den Kondensator C_2 auf. Dieser wird gleichzeitig durch R_6 wieder entladen, so daß sich abhängig von der Lautstärke ein Gleichgewicht einstellt. Je lauter das Geräusch, desto höher die Spannung. Der Ausgabewert des Detektors wird von einem weiteren Operationsverstärker

an das Sensorinterface weitergeleitet. Eine höhere Spannung am Eingang des OPs führt dazu, daß weniger Strom durch R_1 in den Ausgang des Operationsverstärkers und damit nach Masse fließen kann. Infolge dessen steigt der Sensorwert.

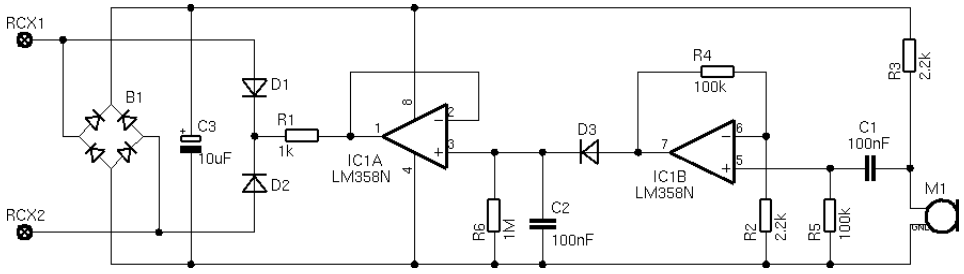


Abbildung A.5: Schaltung des Mindstorms Schallsensors

Zum Aufbau werden die Teile aus Tabelle A.2 benötigt. Der Operationsverstärker ist das wichtigste Bauteil. Es muß ein Modell eingesetzt werden, daß mit geringen Spannungen funktioniert und nur sehr wenig Strom verbraucht. Der LM358N ist ein Bauteil mit genau diesen Eigenschaften.

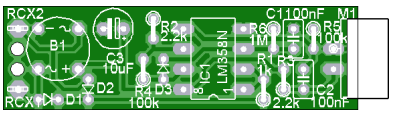


Abbildung A.6: Layout des Mindstorms Schallsensors

	Halbleiter
3x	Diode 1N4148
1x	Brückengleichrichter B40C800
1x	OpAmp LM358N (DIL)

	Widerstände
1x	1 kΩ, 1/8 W
2x	2,2 kΩ, 1/8 W
2x	100 kΩ, 1/8 W
1x	1 MΩ, 1/8 W

	Kondensatoren
2x	100 nF, Keramik, Folie o.ä.
1x	10 µF, 10 V, Elektrolyt oder Tantal

	Verschiedenes
1x	Fotoplatine 13mm x 45mm, einseitig
1x	Elektret-Mikrofonkapsel, mono
1x	Halbiertes Mindstorms-Kabel
1x	Legostein Größe 2x6, schwarz
1x	Legoplatte Größe 2x6, grau

Tabelle A.2: Komponentenliste für den Mindstorms Schallsensor

Beim Aufbau der Schaltung ist vor allem zu beachten, daß sie in den Legostein passen soll. Die Widerstände müssen möglichst dicht auf der Platine sitzen und schräg eingelötet werden, so daß sie weniger Höhe beanspruchen. Der Elko wird je nach Größe liegend eingelötet. Zwei kurze Drahtstücke verbinden das Mikrofon mit der Platine, so daß es in der dafür vorgesehenen Aussparung liegt. Die beiden Leitungen des Mindstorms-Kabels werden von unten durch die Bohrungen am linken Platinenrand geführt, um eine Zugentlastung für das Kabel zu schaffen.

Das Gehäuse des Sensors besteht aus einem Legostein der Größe 2x6 und einer gleich großen Platte. Die Zylinder und Trennstege werden aus dem Inneren des Legosteins herausgefräst, dann wird eine kleine Öffnung für das Kabel auf der einen Seite herausgeschnitten. Auf der anderen Seite wird ein Loch für das Mikrofon gebohrt. Von der Platte müssen alle Knöpfe abgeschnitten werden, und es ist

wahrscheinlich auch nötig, ein wenig Platz für den unteren Rand des Mikrofons zu schaffen. Abbildung A.7 stellt das Ergebnis im Detail dar. Wenn die einzelnen Teile fertig sind und passen, wird das Gehäuse mit Kunststoffkleber verschlossen.

Abbildung A.7: Einbau des Mindstorms Schallsensors in einen Legostein

A.3 Infrarot-Transceiver für den PC

Im Abschnitt 2.6.4 wird die Grundschaltung für einen Infrarot-Transceiver vorgestellt. So ein Baustein versetzt selbst entwickelte Schaltungen in die Lage, mit einem RCX oder einem Tower zu kommunizieren. Im Folgenden soll als einfache Anwendung dieser Technik gezeigt werden, wie ein serieller Tower gebaut werden kann, der voll kompatibel zum Mindstorms Tower ist.

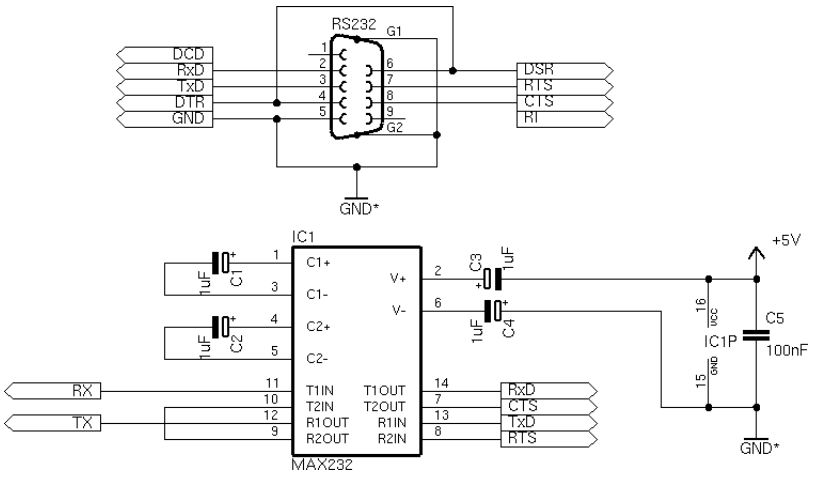


Abbildung A.8: Schaltung zum Anschluß des Transceivers an den PC

Die Grundschaltung muß lediglich um einen Pegelwandler ergänzt werden, der die Spannungspegel in die von RS232 geforderten Werte übersetzt. Dazu eignet sich beispielsweise ein MAX232 von Maxim in Standardbeschaltung, die in Abbildung A.8 zu sehen ist. Von den Steuerleitungen der Schnittstelle sind RTS und CTS verbunden, weil die Software daran den angeschlossenen Tower erkennt. Beide Schaltungen zusammen ergeben einen vollwertigen Ersatz für den Mindstorms-Tower. Er ist im Gegensatz zu diesem ständig empfangsbereit und kann auf den Keepalive-Mechanismus verzichten.

A.4 Prototyp-Platine für PICs mit 8 Pins

Die hier vorgestellten Prototyp-Platinen erlauben es, einfache Programme ohne zusätzliche Hardware laufen zu lassen und zu testen. Als Peripherie sind Leuchtdioden und ein Taster vorhanden, zusätzlich können aber auch externe Bauteile angeschlossen werden. Eine solche Platine für PICs mit 8 Pins ist in Abbildung A.9 zu sehen, sie unterstützt unter anderem die Modelle 12F629 und 12F675.

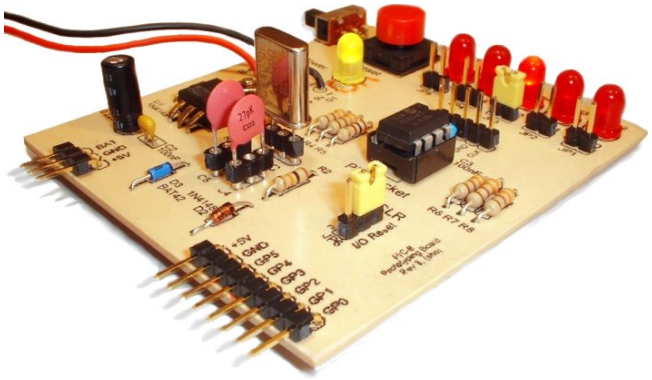


Abbildung A.9: Prototyp-Platine PIC8

Die auf der Platine vorhandenen Baugruppen erlauben eine schnelle Entwicklung eigener Schaltungen.

- Die Platine wird aus einer 9 V Blockbatterie heraus versorgt und erzeugt selbst eine stabilisierte Versorgungsspannung von 5 V. Beide Spannungen sind über Pfostenstecker erreichbar.
- Der PIC wird in einen Sockel gesteckt und kann so relativ einfach ausgetauscht werden.
- Alle 8 Pins des Mikrocontrollers werden über Pfostenstecker nach außen geführt.
- 5 LEDs können einzeln über Jumper mit den Ausgangspins verbunden werden.
- Als externe Taktquellen können ein Resonator oder ein Quarz zusammen mit den passenden Kondensatoren in entsprechende Buchsen gesteckt werden. Bei Benutzung des internen Oszillators sind die Pins nach außen geführt.
- Die integrierte ICSP-Schnittstelle erlaubt eine Neuprogrammierung im laufenden Betrieb.
- Der MCLR-Pin kann über einen Jumper mit dem integrierten Reset-Taster verbunden oder als I/O-Pin benutzt werden. Externe Bauteile können den Pin wie gewünscht ansteuern, sie müssen sich selbst nur vor der Programmierspannung schützen.
- Beim Programmieren wird die Versorgungsspannung durch eine Schottky-Diode davon abgehalten, in den Rest der Schaltung zu gelangen. Ein Jumper kann diese Diode überbrücken, falls der durch die Diode verursachte Abfall der Versorgungsspannung nicht tolerabel ist.

Der Schaltplan in Abbildung A.10 stellt die einzelnen Baugruppen dar. Der Regler VR_1 oben links erzeugt die stabilisierte Versorgungsspannung und läßt LED_1 leuchten, wenn die Schaltung eingeschaltet ist. X_1 führt Eingangs- und Ausgangsspannung des Moduls nach außen. Sockel IC_1 nimmt den PIC auf und verbindet ihn mit der Peripherie, insbesondere mit der Pfostenleiste X_2, die alle I/O-Pins trägt, und der ICSP-Schnittstelle X_3. D_2, R_1, R_8 und S_2 bilden den Reset-Kreis, MCLR kann über JP_6 mit ihnen oder direkt mit X_2 verbunden werden. JP_7 überbrückt bei Bedarf die Schutzdiode in der Versorgungsleitung des PICs. Der Sockel Q_1 nimmt einen Resonator oder einen Quarz auf, bei letzterem kommen noch passende Kondensatoren in die Sockel C_4 und C_5. Die fünf Leuchtdioden LED_2 bis LED_6 können über die Jumper JP_1 bis JP_5 mit dem PIC verbunden werden.

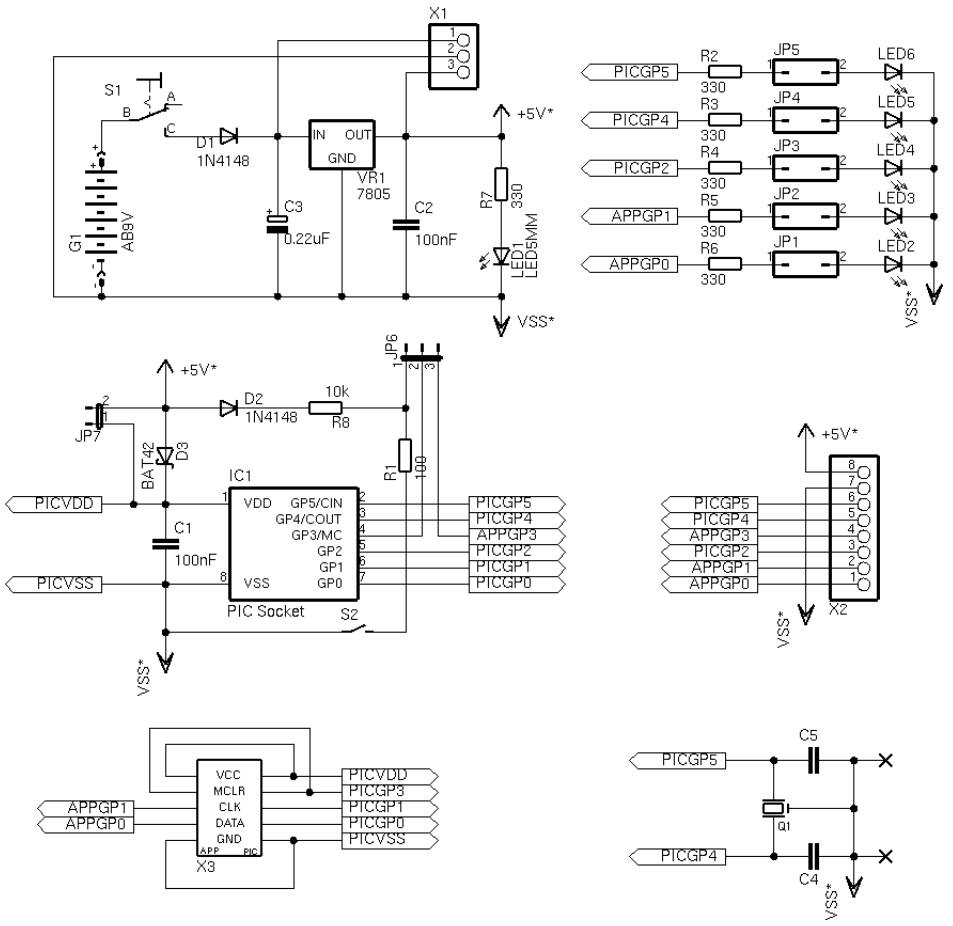


Abbildung A.10: Schaltung der Prototyp-Platine PIC8

Tabelle A.3 zählt alle für den Aufbau der Schaltung nötigen Teile auf. In der Liste fehlt nur noch ein passender Mikrocontroller, um die Platine benutzen zu können.

	Halbleiter
1x	Spannungsregulator 7805 (TO-220)
2x	Diode 1N4148
1x	Schottky-Diode BAT42
5x	LED rot, 5 mm
1x	LED gelb, 5 mm

	Widerstände
1x	100 Ω, 1/8 Watt
6x	330 Ω, 1/8 Watt
1x	10 kΩ, 1/8 Watt

	Kondensatoren
2x	100 nF, Keramik
1x	0,22 µF, 10 V, Elektrolyt

	Pfostenstecker 2.54 mm
5x	1x2 Stifte
1x	1x5 oder 2x5 Stifte
1x	1x3 Stifte
1x	1x3 Stifte abgewinkelt
1x	1x8 Stifte abgewinkelt
3x	1x3 Buchsen, gedreht

	Verschiedenes
1x	Fotoplatine, einseitig
1x	Batterieclip für 9 V Blockbatterie
1x	Prellfreier Taster
1x	Schalter 1xUM
1x	IC-Sockel, 8 Pin DIL

Tabelle A.3: Komponentenliste für Prototyp-Platine PIC8

Beim Bestücken der Platine ist zu beachten, daß die Anschlußkabel der 9 V-Batterie durch die Löcher links von dem Anschlußfeld geführt werden, um eine Zugentlastung zu schaffen.

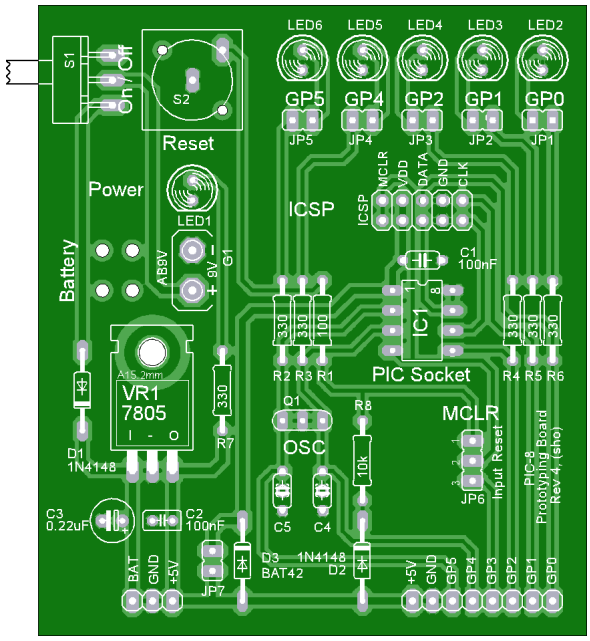


Abbildung A.11: Layout der Prototyp-Platine PIC8

A.5 Prototyp-Platine für PICs mit 18 Pins

Die im vorigen Abschnitt vorgestellte Platine läßt sich einfach so ausbauen, daß sie die größeren PICs mit 18 Pins geeignet ist. Dazu werden weitere Leuchtdioden eingebaut und die zusätzlichen I/O-Pins auf Pfostenstecker geführt, der Rest der Schaltung muß nicht verändert werden. In Abbildung A.12 ist das Ergebnis zu sehen. Die I/O-Pins sind getrennt nach Port A und B auf zwei verschiedene Steckerleisten am oberen und untere Rand der Platine geführt.

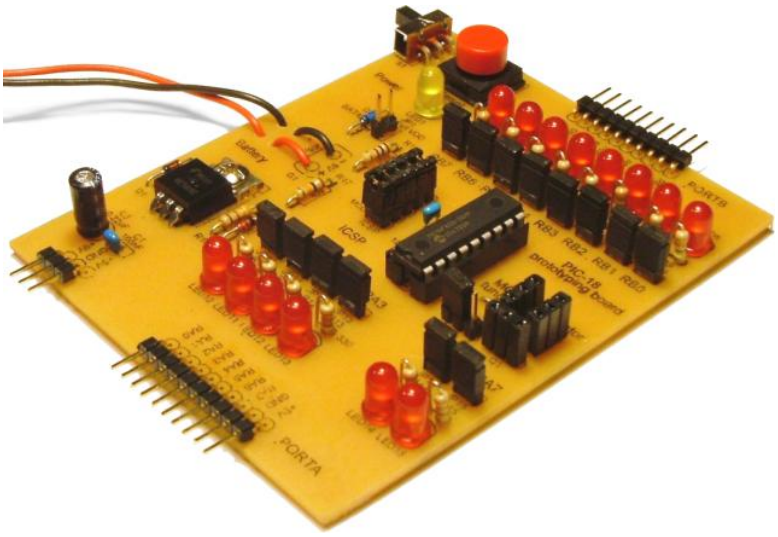


Abbildung A.12: Prototyp-Platine PIC18

	Halbleiter
1x	Spannungsregulator 7805 (TO-220)
2x	Diode 1N4148
1x	Schottky-Diode BAT42
14x	LED rot, 5 mm
1x	LED gelb, 5 mm

	Widerstände
1x	100 Ω, 1/8 Watt
15x	330 Ω, 1/8 Watt
1x	10 kΩ, 1/8 Watt

	Kondensatoren
2x	100 nF, Keramik
1x	0,22 µF, 10 V, Elektrolyt

	Pfostenstecker 2.54 mm
15x	1x2 Stifte
1x	2x5 Stifte
1x	1x3 Stifte
1x	1x3 Stifte abgewinkelt
2x	1x10 Stifte abgewinkelt
3x	1x3 Buchsen, gedreht

	Verschiedenes
1x	Fotoplatine, einseitig
1x	Batterieclip für 9 V Blockbatterie
1x	Prellfreier Taster
1x	Schalter 1xUM
1x	IC-Sockel, 8 Pin DIL

Tabelle A.4: Komponentenliste für Prototyp-Platine PIC18

Die Prototypplatine ist beispielsweise für die PICs 16F84 und seinen Nachfolger 16F628 geeignet. 14 Leuchtdioden können per Jumper mit dem Mikrocontroller verbunden werden, der Taktgenerator ist konfigurierbar und die ICSP-Schnittstelle erlaubt eine Programmierung im laufenden Betrieb. Ein integrierter Spannungswandler und die konfigurierbare Reset-Logik bilden die nötige Minimalbeschaltung, so daß einfache Programme ohne zusätzliche Hardware ausgeführt werden können.

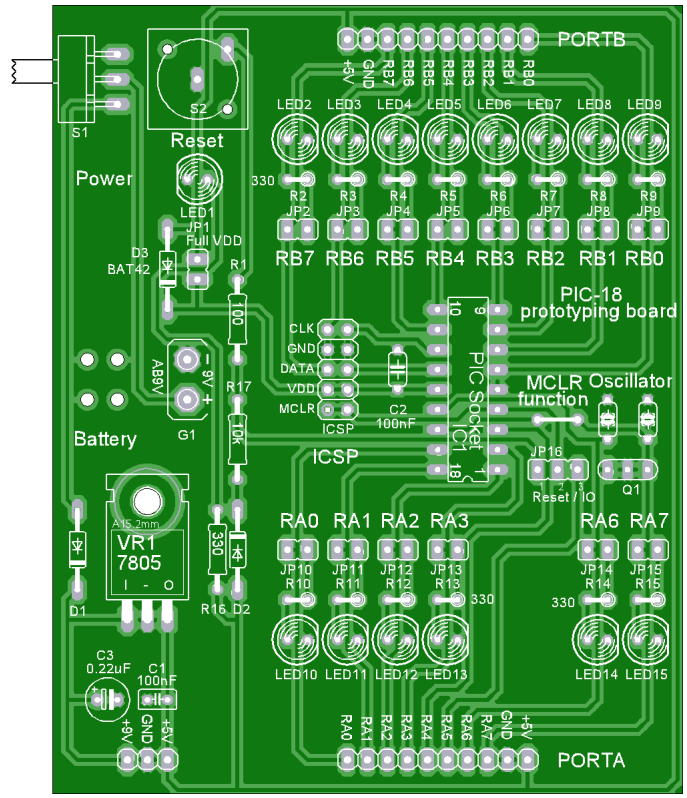


Abbildung A.13: Layout der Prototyp-Platine PIC18

Beim Aufbau ist die übliche Zugentlastung für das Batteriekabel zu beachten.

A.6 Programmiergerät für PICs

Es gibt bereits eine ganze Reihe von kommerziellen Programmiergeräten und freien Bauanleitungen im Internet (vergleiche [Piclinks]). Dennoch gibt es einen guten Grund, ein neues Gerät zu konstruieren. Die gängigen Programmiergeräte eignen sich überwiegend nur schlecht oder gar nicht für die Programmierung in der Anwendungsschaltung. Gerade diese Fähigkeit kann beim Entwickeln und Testen von neuen Steuerprogrammen eine Menge Zeit sparen. Mit ihrer Hilfe kann eine neue Programmversion in wenigen Sekunden übersetzt, gebrannt, gestartet und getestet werden. Das hier beschriebene Programmiergerät zeichnet sich durch die folgenden Fähigkeiten aus:

- Der Brenner ist voll kompatibel zum verbreiteten Tait-Standard ([Tait]).
- In dem integrierten ZIF-Sockel können die gängigsten Standardtypen im DIL-Gehäuse mit 8, 14, 18, 28 und 40 Pins beschrieben und ausgelesen werden. Die ICSP-Schnittstelle stellt alle zum Programmieren benötigten Leitungen direkt zur Verfügung. Über spezielle Adapter wird jeder PIC unterstützt und das Programmieren in der Anwendungsschaltung ermöglicht.
- Das gesamte Design ist auf optimale ICSP-Unterstützung hin entwickelt und erlaubt es, das Programm eines PICs im laufenden Betrieb zu tauschen. Teile der ICSP-Schnittstelle sind auf einen externen Adapter ausgelagert, der genau auf die Zielschaltung abgestimmt werden kann. Die Funktionalität ist transparent und funktioniert daher auch mit Steuerprogrammen, die nicht auf ICSP ausgelegt sind.
- Zur Stromversorgung reicht ein 9 V-Netzteil aus, die Programmierspannung wird intern erzeugt.
- Das Parallelport-Interface ist sehr störsicher und erlaubt Taktraten bis zu 0,5 MHz.
- Die Ausgangstreiber können größere Ströme liefern, lassen sich aber auch hochohmig schalten.

Abbildung A.14: Programmiergerät für PIC Mikrocontroller

Die Schaltung ist in Abbildung A.15 zu sehen. Die Versorgungsspannung von 9-12 V wird durch die Baugruppe um VR_1 auf 5 V stabilisiert, der Regler muß dabei passiv gekühlt werden. Gleichzeitig erzeugt IC_1 ein Rechtecksignal mit einer Frequenz von 7 kHz und treibt damit eine Ladungspumpe an. Diese verdoppelt die Eingangsspannung und leitet sie VR_2 zu, der daraus die Programmierspannung von 13,2 V erzeugt. Diese Spannung ist für alle gängigen PICs passend.

Das Parallelport-Interface ist mit einem RC-Netzwerk und dem Schmitt-Trigger IC_2 ausreichend gegen Störimpulse abgesichert. Die eigentlichen Schaltfunktionen werden von dem Open Collector-Treiber IC_3 gesteuert. Die Leitung D_3 des Parallelports kontrolliert die Programmierspannung. Eine logische 0 öffnet T_1 und legt die 13,2 V an den MCLR-Pin des PICs, der Steuerstrom des Transistors läßt gleichzeitig die rote LED aufleuchten. Steuerleitung D_2 übernimmt die selbe Aufgabe für die Versorgungsspannung von 5 V und öffnet T_2, wobei die gelbe LED aufleuchtet. Für T_1 und T_2 werden Leistungstransistoren eingesetzt, damit der Spannungsverlust durch sie möglichst gering ist.

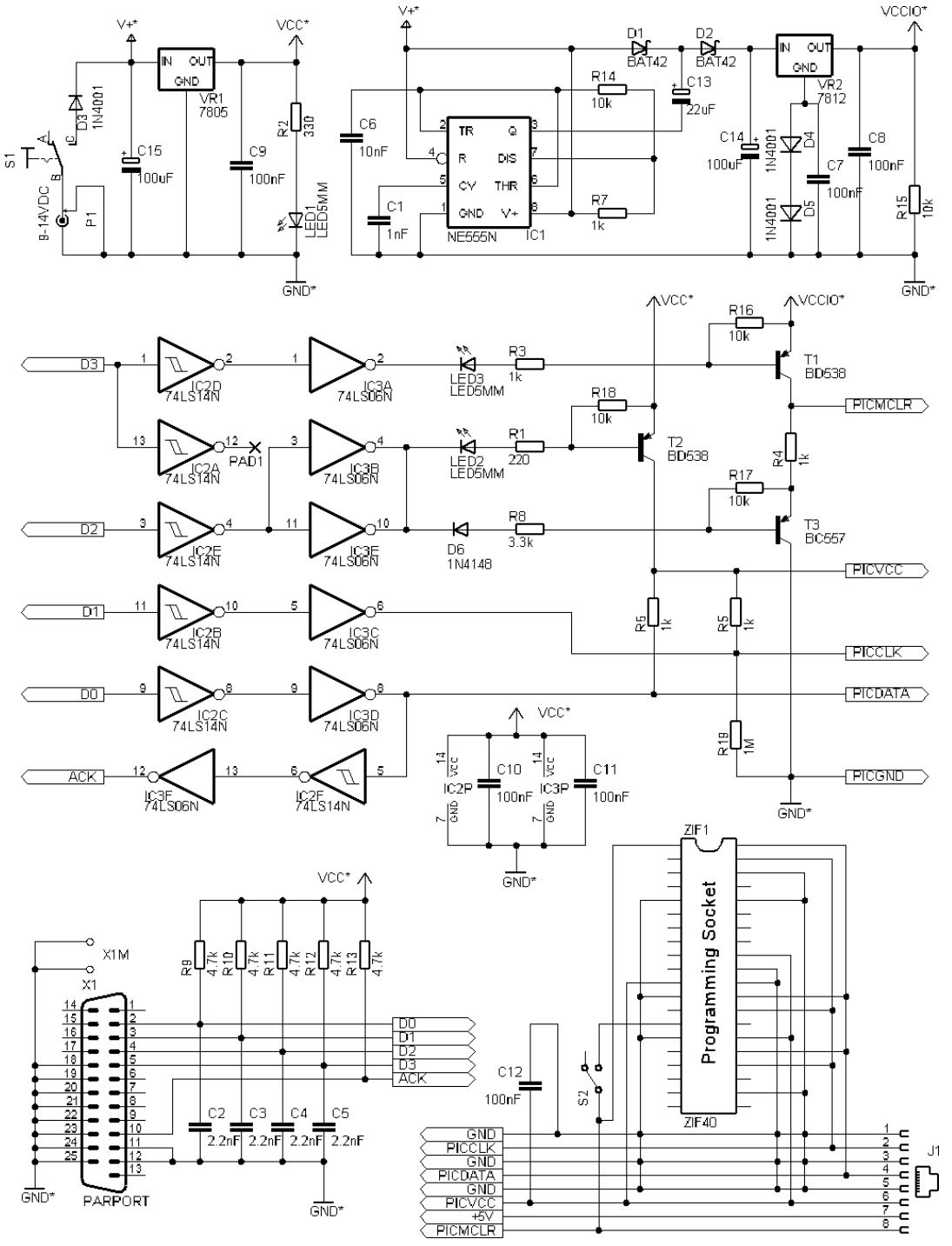


Abbildung A.15: Schaltung des Programmiergerätes

Das Einschalten der Versorgungsspannung öffnet gleichzeitig die Verbindung von MCLR nach Masse. Dadurch kann ein Reset des PICs ausgelöst werden. Ist die Programmierspannung nicht aktiviert, wird MCLR über 1 kΩ auf Masse gezogen, sonst liegt an ihm die Programmierspannung an. Sind Programmierspannung und Versorgungsspannung ausgeschaltet, ist der MCLR-Ausgang hochohmig. Über die Leitung D_1 wird das Taktsignal für den PIC erzeugt, D_0 transportiert die Daten. Wird diese Leitung auf eine logische 1 gelegt, sperrt der Treiberbaustein IC_{3D} und der PIC kann die Datenleitung steuern. Das Datensignal wird an die Leitung ACK des Parallelports zum Rechner zurückgegeben. Es dient zur Kontrolle der gesendeten Daten und zum Empfang von Antworten des PICs. Die Programmierleitungen werden an den internen ZIF-Sockel und den externen ICSP-Adapter weitergegeben. Am ZIF-Sockel entscheidet ein Schalter darüber, welcher Pin mit MCLR verbunden wird. Eine Einstellung ist für die großen PICs mit 28 und 40 Pins, die andere für die kleineren Modelle.

An der Schaltung lassen sich auch die technischen Daten ablesen, auf die sich eine Anwendungsschaltung einstellen muß, damit sie sich einwandfrei programmieren läßt.

- Der Brenner kann je nach Netzteil einen Strom von bis zu 500 mA auf der 5 V-Leitung liefern.
- Bei der Programmierspannung sind es etwa 50 mA, beim Einsatz eines stärkeren Netzteils mit einer Spannung von 12 V steigt der Strom auf maximal 100 mA.
- Die Impedanz des MCLR-Ausgangs beträgt wenige Ohm beim Programmieren, 1 kΩ beim Auslösen eines Resets und mindestens 150 kΩ im hochohmigen Zustand.
- Die Impedanz der Takt- und Datenleitungen beträgt 1 kΩ im High-Zustand, wenige Ohm im Low-Zustand und 1 MΩ im hochohmigen Zustand. Die Treiberstufe kann Ströme von bis zu 30 mA aus der Schaltung aufnehmen.

Für den Aufbau der Schaltung werden die Teile aus Tabelle A.5 benötigt.

	Halbleiter
3x	Diode 1N4001
1x	Diode 1N4148
2x	Schottky-Diode BAT42
1x	LED rot, 5 mm
1x	LED gelb, 5 mm
1x	LED grün, 5 mm
1x	PNP-Transistor BC557
2x	PNP-Transistor BC538 (TO-220)
1x	Spannungsregler 7805 (TO-220)
1x	Spannungsregler 7812 (TO-220)
1x	IC NE555N
1x	IC 74LS06N
1x	IC 74LS14N

	Verschiedenes
2x	Fotoplatine
2x	Schalter 1xUM
1x	Netzteilbuchse 5mm
1x	RJ-45-Buchse für Printmontage
1x	25pol. Sub-D Stecker, Printmontage
1x	40pol. ZIF-Sockel

	Widerstände
1x	220 Ω, 1/8 Watt
1x	330 Ω, 1/8 Watt
5x	1 kΩ, 1/8 Watt
1x	3,3 kΩ, 1/8 Watt
5x	4,7 kΩ, 1/8 Watt
5x	10 kΩ, 1/8 Watt
1x	1 MΩ, 1/8 Watt

	Kondensatoren
1x	1 nF
4x	2,2 nF
1x	10 nF
6x	100 nF, Keramik
4x	22 µF, 35 V, Elektrolyt
4x	100 µF, 35 V, Elektrolyt
4x	220 µF, 35 V, Elektrolyt

	Gehäusebau
3x	LED-Fassung 5mm
1x	Kühlkörper für TO220-Gehäuse
1x	Pultgehäuse Bopla BP810

Tabelle A.5: Komponentenliste für das Programmiergerät

Die Schaltung besteht aus zwei Platinen, eine trägt das Programmiergerät und die andere den ZIF-Sockel für die Frontplatte. Zunächst wird die Hauptplatine bestückt, wobei statt dem Schalter und den drei Leuchtdioden Kabel eingelötet werden. In die Gehäuserückseite werden Löcher für die drei Buchsen gefräst, die möglichst eng sein sollten, so daß die Platine stabilen Halt findet.

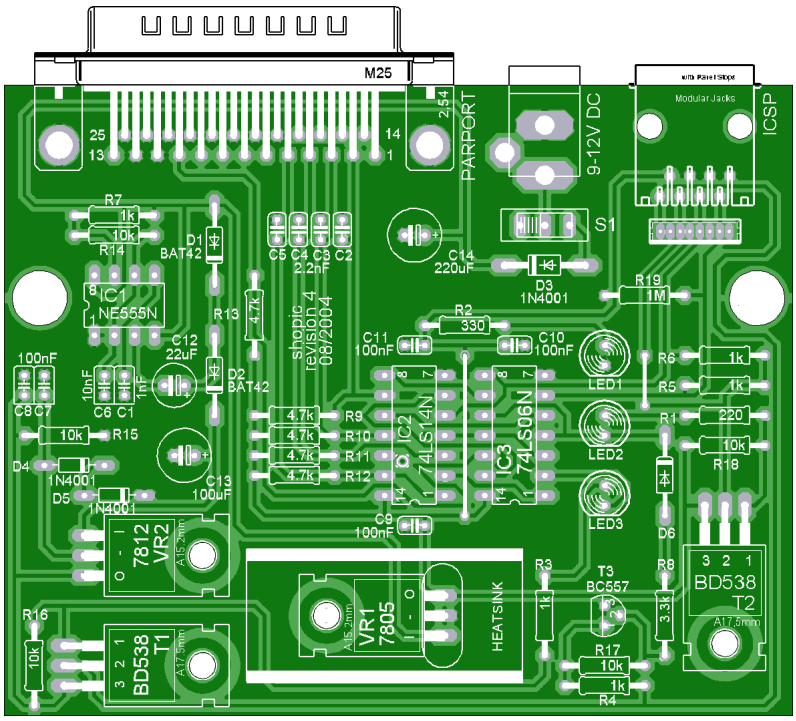


Abbildung A.16: Hauptplatine des Programmiergerätes

Anschließend wird eine Folie mit der Beschriftung gedruckt und auf die Frontplatte des Gehäuses geklebt. Die Folie gibt vor, wo die Löcher für die LEDs, die Schalter und den ZIF-Sockel gefräst werden müssen. Die LEDs werden mit ihren Fassungen eingesetzt und mit der Hauptplatine verbunden, genauso der Hauptschalter. Die Platine mit dem ZIF-Sockel wird zuletzt bestückt, der 100 nF-Kondensator gehört auf die Unterseite, genauso wie das Flachbandkabel, das die beiden Platinen untereinander verbindet und mit Heißkleber gesichert wird.

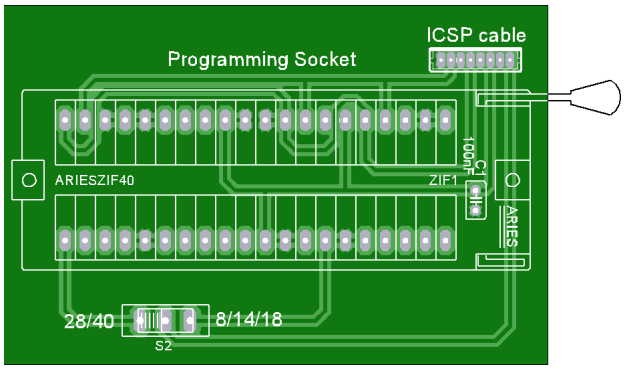


Abbildung A.17: ZIF-Platine des Programmiergerätes

Zum Schluß wird die ZIF-Platine ebenfalls festgeklebt und das Gehäuse verschlossen.

A.7 ICSP-Adapter für das Programmiergerät

Das Programmiergerät führt seine 5 Steuerleitungen über eine Buchse nach außen. Hier kann über ein Verlängerungskabel eine Adapterplatine angeschlossen werden, die für die Anbindung der Zielschaltung verantwortlich ist. Durch diese Modularisierung kann die Adapterplatine mit wenig Arbeit an die Zielschaltung angepaßt werden, das Programmiergerät braucht nicht verändert zu werden. Der einfachste denkbare Adapter verbindet die Steuerleitungen nur mit den entsprechenden Pins des PICs.

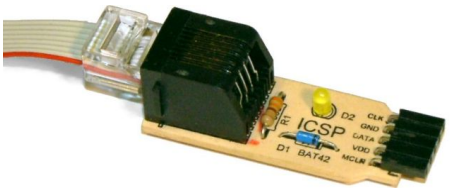


Abbildung A.18: ICSP-Platine, Typ 1

Dieser in Abbildung A.18 dargestellte Adapter wird auf einen Pfostenstecker in der Zielschaltung gesteckt, der die folgende Belegung ausweist.

MCLR	VCC	DATA	GND	CLK

Tabelle A.6: Pinbelegung ICSP-Stecker Typ 1

Der Aufbau der kleinen Platine ist sehr einfach. Eine Schottky-Diode verhindert, daß die Versorgungsspannung der Zielschaltung in das Programmiergerät gelangt. Die LED leuchtet auf, wenn gerade ein Brennvorgang stattfindet.

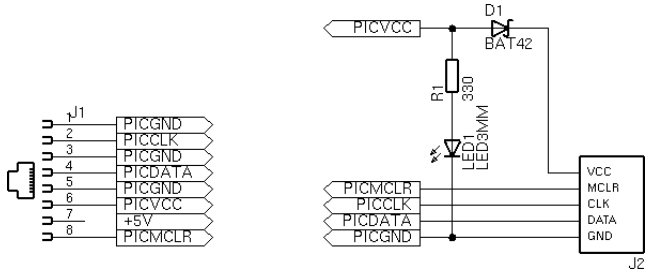


Abbildung A.19: Schaltung des ICSP-Interfaces, Typ 1

Für den Aufbau werden auch nur sehr wenige Teile benötigt. Sie sind in Tabelle A.7 aufgeführt.

	Halbleiter
1x	Schottky-Diode BAT42
1x	LED gelb, 3 mm
	Widerstände
1x	330 Ω, 1/8 Watt

	Pfostenstecker 2.54 mm
1x	1x5 Buchsen, abgewinkelt
	Verschiedenes
1x	RJ45-Buchse für Printmontage
1x	Fotoplatine, einseitig

Tabelle A.7: Komponentenliste für das ICSP-Interface Typ 1

Zusätzlich zu der Platine wird noch ein Kabel benötigt, das zwei RJ45-Stecker über ein Flachbandkabel verbindet. Es muß so beschaffen sein, daß die Pinbelegung der beiden Stecker identisch ist.

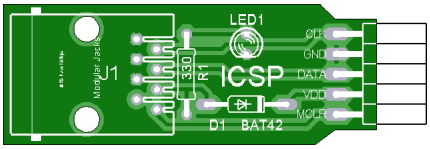


Abbildung A.20: Layout des ICSP-Interfaces, Typ 1

Für viele Schaltungen reicht der einfache Adapter nicht aus, insbesondere wenn Daten- und Taktsignal sich die Pins mit anderen Baugruppen in der Zielschaltung teilen. Die einzig universelle Lösung für solche Fälle ist, einen mechanischen Umschalter für diese Funktionen zu benutzen. Dieser Ansatz führt auch zu einem neuen Programmierstecker mit einer geänderten Pinbelegung.

$MCLR_A$	VCC_A	$DATA_A$	GND	CLK_A
$MCLR_P$	VCC_P	$DATA_P$	GND	CLK_P

Tabelle A.8: Pinbelegung ICSP-Stecker Typ 1

Die Pins in der oberen Reihe sind mit der Anwendungsschaltung verbunden, die untere Reihe hingegen mit dem PIC. Normalerweise sind alle Pins der oberen mit denen der unteren Reihe über Jumper verbunden, der PIC ist dann vollständig an seine Schaltung angeschlossen. Entfernt man diese Brücke, wird auch der PIC von der Schaltung getrennt. Die aufgesteckte Adapterplatine entscheidet dann, ob sie den PIC mit der Zielschaltung oder dem Programmiergerät verbindet.

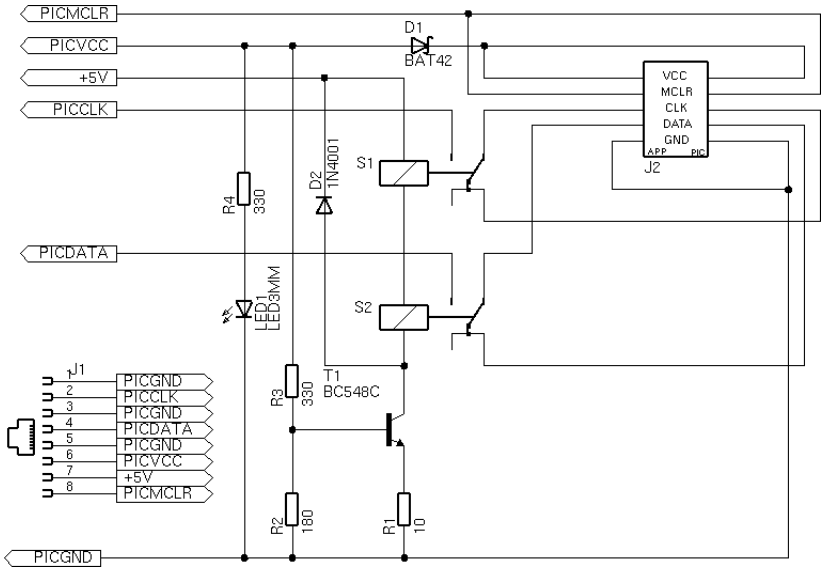


Abbildung A.21: Schaltung des ICSP-Interfaces, Typ 2

In der Schaltung in Abbildung A.21 übernehmen zwei Relais diese Aufgabe. Der Adapter wird vom Programmiergerät gespeist und schaltet beim Aktivieren der Versorgungsspannung die besagten Relais so, daß der PIC von seiner Schaltung getrennt und an das Programmiergerät gekoppelt wird. Gleichzeitig leuchtet die LED auf. Ein dritter Umschalter für MCLR wäre zwar auch denkbar, in der

Regel ist dieser aber nicht nötig. MCLR kann nur als Eingang betrieben werden und läßt sich daher durch einen großen Widerstand in Richtung der Anwendungsschaltung ausreichend isolieren. Auf einen Umschalter für die Versorgungsspannung wurde ebenfalls verzichtet. Soll die Anwendungsschaltung mit einer anderen Spannung als 5 V laufen, wäre dies aber sehr sinnvoll.

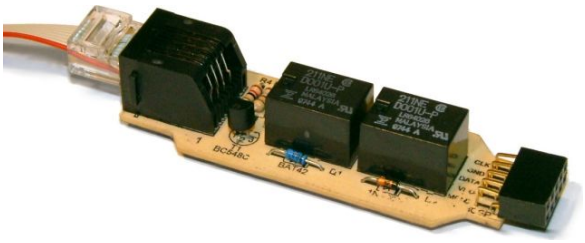


Abbildung A.22: ICSP-Platine, Typ 2

Statt eines Relais könnte auch ein CMOS Analog Switch der neusten Generation eingesetzt werden, wenn sein Innenwiderstand niedrig genug ist und seine Durchlaßeigenschaften symmetrisch sind.

	Halbleiter
1x	Schottky-Diode BAT42
1x	Diode 1N4148
1x	LED gelb, 3 mm

	Verschiedenes
1x	RJ45-Buchse für Printmontage
1x	Fotoplatine, einseitig
2x	Relais 1xUM, geeignet für 2 V

	Pfostenstecker 2.54 mm
1x	2x5 Buchsen, abgewinkelt

	Widerstände
1x	10 Ω, 1/8 Watt
1x	180 Ω, 1/8 Watt
2x	330 Ω, 1/8 Watt
1x	NPN-Transistor BC548C

Tabelle A.9: Komponentenliste für ICSP-Platine Typ 2

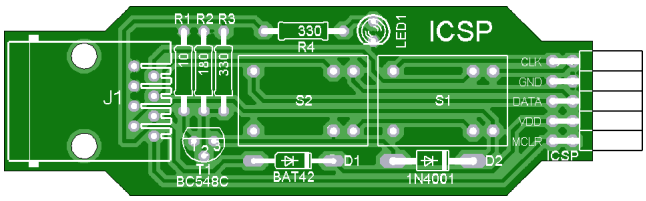


Abbildung A.23: Layout des ICSP-Interface (Typ 2)

A.8 Serielles Debug-Interface für PICs

Die Debug-Bibliothek aus Anhang B.6 benutzt einen Ausgangspin des PICs, um Daten im RS232-Protokoll an einen PC zu senden. Auf diese Weise kann der PC Statusinformationen des Programms aufzeichnen und ausgeben. Damit der PIC mit der seriellen Schnittstelle des Rechners verbunden werden kann, müssen die Signale durch einen Pegelwandler auf die von RS232 vorgeschriebenen Spannungspegel gebracht werden. Ein IC vom Typ MAX232 in Standardbeschaltung kann diese Aufgabe einfach übernehmen.

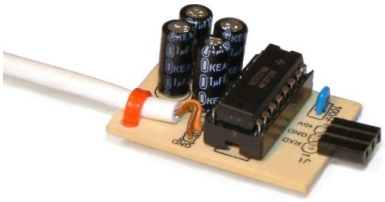


Abbildung A.24: Serielles Debugging-Interface

In Abbildung A.25 ist die Grundschaltung zu sehen. Der IC wird über eine dreipolige Anschlußbuchse mit Strom und Daten versorgt. Mit Hilfe der 4 Elkos und einer internen Ladungspumpe werden Spannungen von nahezu +10 V und -10 V erzeugt und die serielle Schnittstelle des PCs damit angesteuert. Der Stromverbrauch beträgt einige Milliampere und hängt auch von der seriellen Schnittstelle ab.

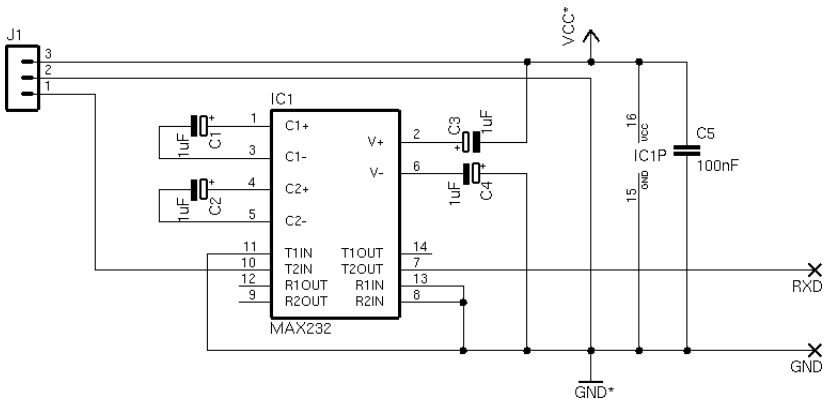


Abbildung A.25: Schaltung das seriellen Debuginterfaces

Das Layout ist so gewählt, daß die Schaltung an beide Prototyp-Schaltungen angesteckt werden kann. Drei der äußersten Pins liefern eine Datenleitung die Versorgungsspannung in der richtigen Reihenfolge.

	Halbleiter
1x	IC MAX232

	Pfostenstecker 2.54 mm
1x	1x3 Buchsen, abgewinkelt

	Kondensatoren
1x	100 nF, Keramik
4x	1 µF, 16 V, Elektrolyt

	Verschiedenes
1x	Fotoplatine, einseitig
1x	Kabel mit 9-poliger Sub-D Buchse

Tabelle A.10: Komponentenliste für das serielle Debug-Interface

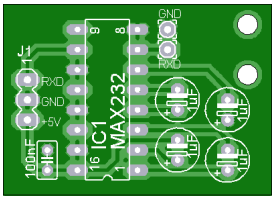


Abbildung A.26: Layout des Debuginterfaces

A.9 Ultraschallreflektoren

Kapitel 4 hat gezeigt, daß die Kapseln zum Senden und Empfangen von Ultraschall mit kegelförmigen Reflektoren ausgestattet werden müssen. Nur dann können sie in der Ebene richtungsunabhängig arbeiten. Die Reflektoren müssen den gerichteten Schall aus der Sendekapsel gleichmäßig verteilen bzw. aus jeder Richtung einfangen und in den Empfänger umleiten können. Die optimale Geometrie des Reflektors läßt sich berechnen. Bei 40 kHz ist ein auf der Spitze stehender Kegel mit 48,5 mm Bodendurchmesser, einer Höhe von 14 mm und einem Öffnungswinkel von 120° gut geeignet. Dieser Abschnitt beschreibt, wie ein Reflektor mit diesen Maßen hergestellt werden kann. Als Material kommt Glasfaserverstärkter Kunststoff (GFK) auf der Basis von Polyesterharz zum Einsatz. Dieses Material ist leicht, formstabil, bruchfest und kann relativ einfach in jede gewünschte Form gebracht werden.

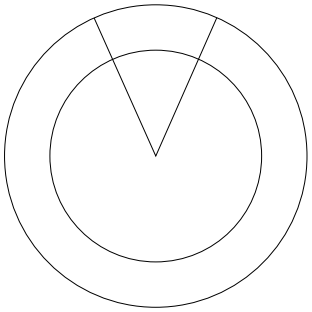


Abbildung A.27: Vorlage für die Negativform bei der Reflektorherstellung

Als erstes wird eine Negativform aus Gips hergestellt, die später beim Herstellen der einzelnen Reflektoren benötigt wird. Dazu wird eine Kreisscheibe wie in Abbildung A.27 aus Overheadfolie ausgeschnitten. Der Durchmesser des inneren Kreises beträgt 56 mm, der äußere Ring ist deutlich größer. Dann wird die Kreisscheibe radial entlang einer der beiden Linien eingeschnitten und zu einem Kegel zusammengerollt. Dabei muß die Schnittkante auf der zweiten radialen Linie liegen. Der Kegel wird dann mit Tesafilm fixiert und in eine Schüssel gestellt, die dünn mit Trennspray eingesprüht ist. Dabei muß es sich um eine konvexe Schüssel mit ebenem Boden handeln.

Abbildung A.28: Herstellung der Negativform aus Gips

Leichter Druck auf die Kegelspitze sorgt dafür, daß die Form dicht an dem Boden anliegt und der Kegel nicht deformiert ist. Dann wird Spachtelmasse oder Gips möglichst homogen und blasenfrei angerührt und in der Schüssel verteilt, bis eine ebene Oberfläche entstanden ist. Die entstandene Form muß gründlich aushärten, dann kann sie vorsichtig durch Klopfen aus der Schüssel gelöst werden. Zum Schluß wird der Plastikkegel entfernt.

Die Herstellung des Reflektors beginnt mit einem Papierkegel, der mit ähnlichen Maßen wie zuvor der Plastikkegel hergestellt wird. Die Vorlage ist links in Abbildung A.29 zu sehen. Der innere Ring hat wieder einen Durchmesser von 56 mm, der äußere ist nur wenige Millimeter größer.

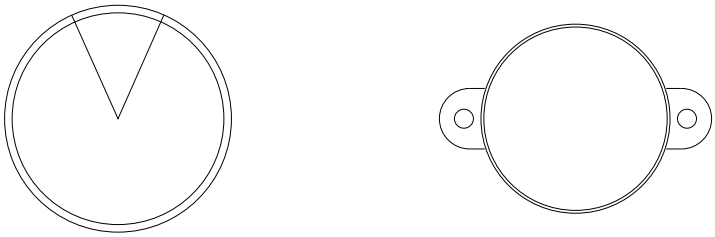


Abbildung A.29: Vorlage für die Positivformen bei der Reflektorherstellung

Das Papier wird zu einem Kegel zusammengerollt und mit Kleber fixiert. Die Linien müssen außen liegen, sie werden nachher als Markierungen beim Sägen gebraucht. Die Negativform aus Gips wird mit reichlich Trennspray eingesprüht und der Papierkegel reingestellt. Nach dem bekannten Schema wird ein Kegel aus einer Glasfasermatte ausgeschnitten und in den Papierkegel gestellt. Das Polyesterharz wird entsprechend der Anleitung mit Härter vermischt und gleichmäßig auf der Glasfasermatte verteilt, bis sie gründlich durchgetränkt ist.

Abbildung A.30: Herstellung eines Reflektors

Sobald das Kunstharz fest genug ist, wird der Reflektor aus der Gipsform entnommen und zum Trocknen aufgestellt, bis keine Lösungsmittel mehr entweichen und die Oberfläche sich nicht mehr klebrig anfühlt. Mäßige Wärme kann helfen, diesen Prozeß zu beschleunigen. Anschließend kann der überstehende Rand außerhalb des 56 mm-Ringes abgesägt und die Kante glattgeschliffen werden. Für die Montage muß der Kegel auf einer Grundplatte befestigt werden. Sie wird nach dem Muster auf der rechten Seite der Abbildung A.29 aus Plexiglas ausgesägt und glattgeschliffen, so daß der Kegel genau in den inneren Ring paßt. Kegel und Trägerplatte werden mit Kunststoffkleber aneinander befestigt und das entstandene Gebilde lackiert.

Nach einer ausreichenden Trockenzeit ist der Reflektor fertig. Er kann mit Schrauben der Größe M4 mit 35 mm Länge an seinem späteren Einsatzort befestigt werden. Die Höhe des Reflektors über der Ultraschallkapsel kann dabei leicht über die Position der Muttern eingestellt werden.

Abbildung A.31: Fertiger Ultraschallreflektor

Alle für den Prozeß nötigen Druckvorlagen befinden sich maßstabsgetreu auf der beiliegenden CD.

A.10 Steuereinheit des Ortungssystems

Die Leuchtfeuer des Ortungssystems werden von einer zentralen Steuereinheit koordiniert und mit Strom versorgt. Die Steuereinheit erkennt, wie viele Leuchtfeuer an ihr angeschlossen sind und kann an jedes Befehle senden. Dadurch steuert sie, welches Leuchtfeuer wann Ultraschall aussendet. Der Benutzer kann zwischen mehreren vorgegebenen Sendemustern per Knopfdruck wechseln. Daneben übernimmt die Steuereinheit das Senden der Funkimpulse, so daß in den Leuchtfeuern keine Funksender enthalten sein müssen. Die genaue Funktionsweise ist in 7.1 beschrieben.

Abbildung A.32: Steuereinheit des Ortungssystems

Das Gerät besitzt an der Rückseite zwei Anschlüsse, der eine ist für den Anschluß der Leuchtfeuer, der andere ist für ein Steckernetzteil mit 9-20 V vorgesehen, wobei höhere Spannungen eine größere Reichweite erlauben. Mit einem Hauptschalter an der Seite wird die Steuereinheit eingeschaltet, die beiden LEDs und das Display zeigen den Systemstatus an. Mit Hilfe der beiden roten Taster ist es möglich, zwischen den Sendemustern umzuschalten.

Der Kern der Schaltung besteht aus einem Mikrocontroller mit 4 MHz-Quarz und ICSP-Schnittstelle, darum gruppieren sich mehrere Peripheriebausteine. Das LC-Display ist über 7 Leitungen mit Port B des PICs verbunden. Über diese Verbindung werden Steuerbefehle und Daten für die Anzeige gesendet. Das Potentiometer R_{14} dient zur Kontrasteinstellung. Zwei der Leitungen von Port B teilt sich das Display mit den beiden Tastern SW_2 und SW_3. Sie stören die Kommunikation mit dem Display durch den relativ großen Pulldown-Widerstand nicht, solange die Ausgangstreiber im PIC oder Display aktiv sind. Konfiguriert der PIC die entsprechenden Pins aber als Eingänge mit aktivierter Weak Pullup-Funktion, kann der Zustand der Taster ausgelesen werden.

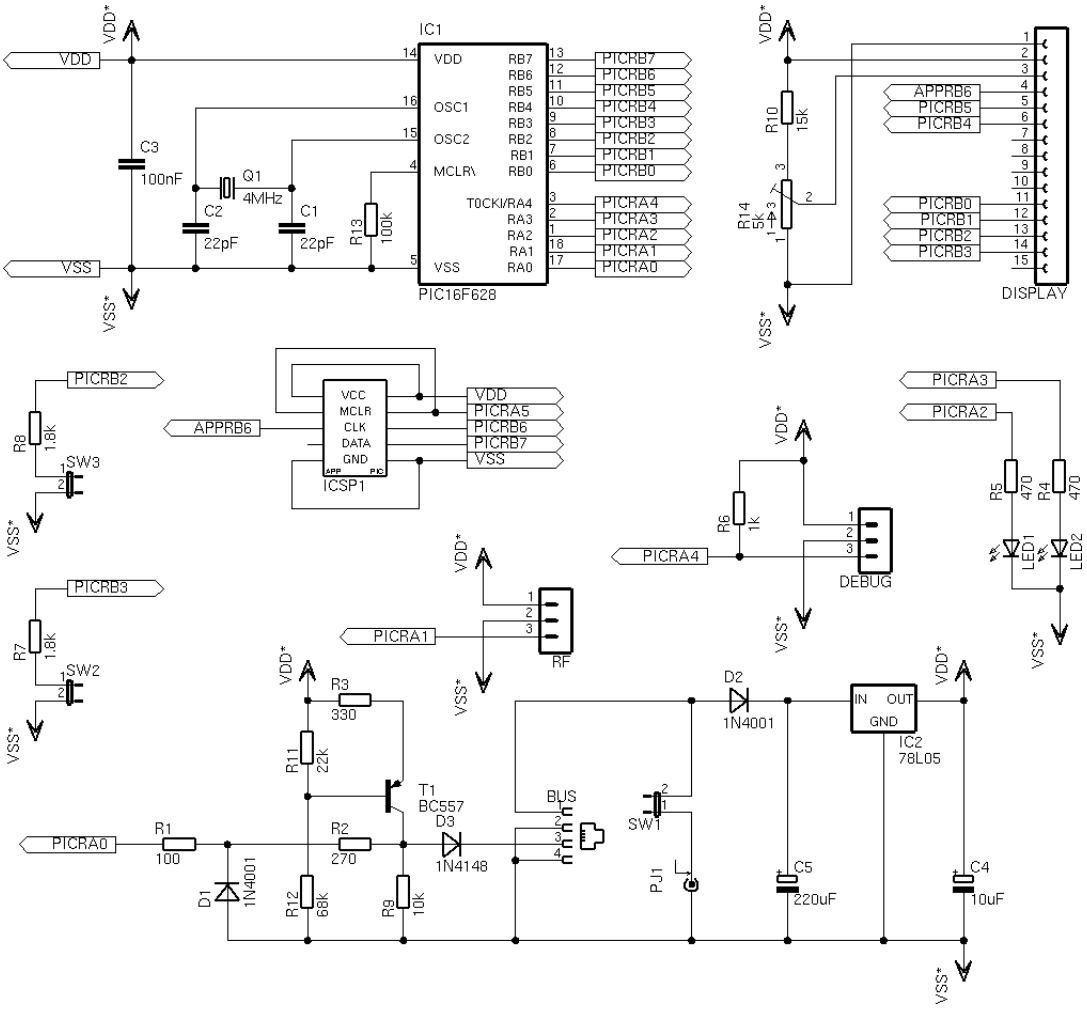


Abbildung A.33: Schaltung der Steuereinheit

T_1 bildet zusammen mit R_3, R_{11} und R_{12} eine Konstantstromquelle, die einen Strom von 2 mA liefert und über D_3 an die Leuchtfeuer weitergibt. Mit dem durch D_1, R_1 und R_2 abgesicherten Pin RA_0 kann der PIC die Spannung auf dem Leuchtfeuerbus messen oder beeinflussen, indem der Pin aus Ausgang konfiguriert wird. Auf diese Weise wird die Kommunikation mit den Leuchtfeuern abgewickelt. Die restlichen Baugruppen bilden die Stromversorgung, die beiden Anzeige-LEDs, die Debug-Schnittstelle und die Verbindung zum Funksendemodul. Letzteres muß im Gehäuse möglichst weit entfernt von Metallteilen angebracht und mit einer Antenne versehen werden, anderenfalls ist das abgestrahlte Signal zu schwach. Für die Plazierung der Bedienelemente gibt es auf der beiliegenden CD eine Bohrschablone. Nach der Programmierung des PICs mit der Firmware `controller.asm` muß die ICSP-Schnittstelle mit Jumpern überbrückt werden, dann ist das Gerät einsatzbereit.

	Widerstände
1x	100 Ω, 1/8 Watt
1x	270 Ω, 1/8 Watt
1x	330 Ω, 1/8 Watt
2x	470 Ω, 1/8 Watt
1x	1 kΩ, 1/8 Watt
2x	1,8 kΩ, 1/8 Watt
1x	10 kΩ, 1/8 Watt
1x	15 kΩ, 1/8 Watt
1x	22 kΩ, 1/8 Watt
1x	68 kΩ, 1/8 Watt
1x	10 kΩ, 1/8 Watt
1x	5 kΩ Poti, stehend

	Kondensatoren
2x	22 pF
1x	100 nF
1x	10 µF, 6,3 V, Elektrolyt
1x	220 µF, 35 V, Elektrolyt

	Pfostenstecker 2.54 mm
1x	3 Stifte
1x	5x2 Stifte

	Halbleiter
2x	Diode 1N4001
1x	Diode 1N4148
1x	Mikrocontroller PIC16F628, DIL
1x	Spannungsregler 78L05
1x	LED 5mm, rot
1x	LED 5mm, grün
1x	PNP-Transistor BC557

	Gehäusebauelemente
1x	Schalter
2x	Taster
2x	LED-Fassung 5mm
1x	Montagerahmen für Display
1x	Gehäuse Teko TK 11S, schwarz
	Verbindungskabel zur Platine

	Verschiedenes
1x	Fotoplatine, einseitig
1x	Quarz 4 MHz, HC49U
1x	Buchse AMP 520249-2
1x	Netzteilbuchse
1x	LC-Display 2x16 Zeichen (HD44780)
1x	Sendemodul 433 MHz, 5 V

Tabelle A.11: Komponentenliste für die Steuereinheit

Die Platine wird bestückt und im Gehäuse verschraubt. Kabel bilden die Verbindungen zur Peripherie.

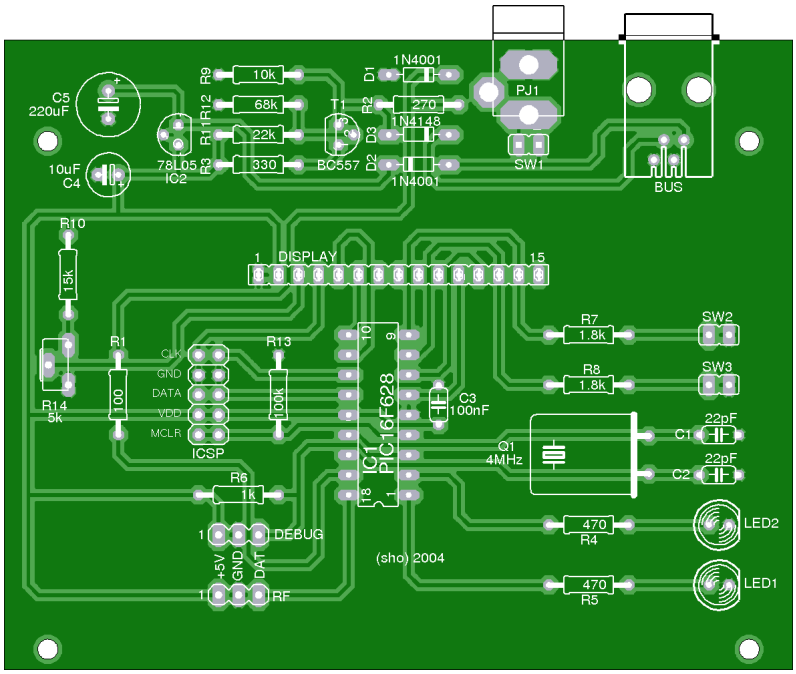


Abbildung A.34: Layout der Steuereinheit

A.11 Stationäres Leuchtfeuer

Die Leuchtfeuer enthalten jeweils einen Ultraschallsender. Mehrere davon werden in der Umgebung des Roboters räumlich verteilt und liefern die Daten für die Koordinatenberechnung. Die Leuchtfeuer werden in einer Kette untereinander verbunden, am Anfang der Kette steht die Steuereinheit und am Ende ein Terminator. Eine in dem Kabel enthaltene Kommunikationsleitung überträgt Befehle zur Aktivierung von der Steuereinheit zu den Leuchtfeuern.

Abbildung A.35: Leuchtfeuer für das Ortungssystem

Die Schaltung ist verhältnismäßig einfach, ihr Aufbau ist in Abbildung A.36 dargestellt.

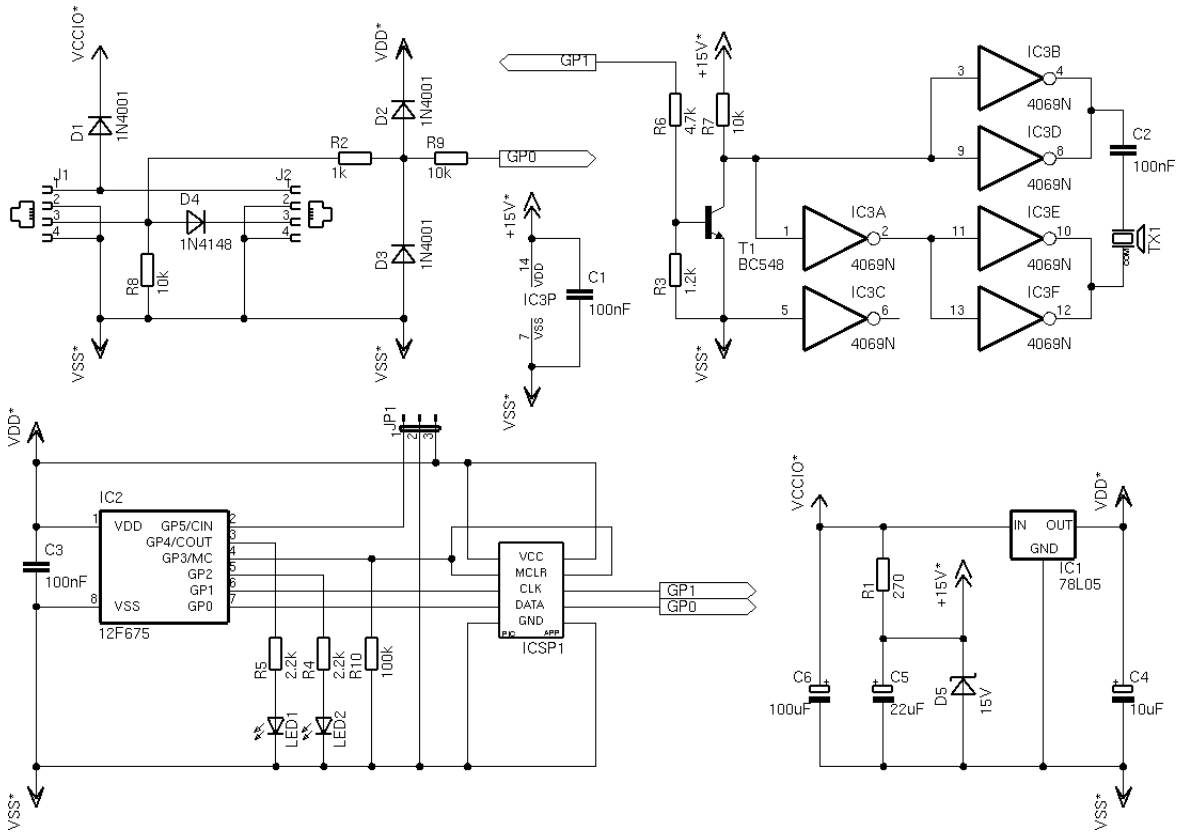


Abbildung A.36: Schaltung des Leuchtfeuers

Kern der Schaltung ist der Mikrocontroller mit Debug-Schnittstelle und ICSP-Interface. Dieser kann an GP0 die Spannung an der Datenleitung des Businterfaces messen, die Bauteile D_2, D_3, R_2 und R_9 schützen den PIC vor Überspannungen. Die Stromversorgung besteht aus zwei Teilen, ein 78L05 erzeugt die 5 V für die Logikbausteine und über eine Zenerdiode werden bis zu 15 V für den Ultraschallsender zur Verfügung gestellt, der im Wesentlichen aus T_1 und IC_3 besteht. Zwei Leuchtdioden stehen für die Anzeige des Systemstaus zur Verfügung.

	Widerstände
1x	270 Ω, 1/8 Watt
1x	1 kΩ, 1/8 Watt
1x	1,2 kΩ, 1/8 Watt
2x	2,2 kΩ, 1/8 Watt
1x	4,7 kΩ, 1/8 Watt
3x	10 kΩ, 1/8 Watt
1x	100 kΩ, 1/8 Watt

	Pfostenstecker 2.54 mm
1x	3 Stifte
1x	5x2 Stifte

	Kondensatoren
3x	100 nF
1x	10 µF, 6,3 V, Elektrolyt
1x	22 µF, 16 V, Elektrolyt
1x	100 µF, 35 V, Elektrolyt

	Halbleiter
3x	Diode 1N4001
1x	Diode 1N4148
1x	Zenerdiode 15 V
1x	Mikrocontroller PIC16F628, DIL
1x	Spannungsregler 78L05
1x	CMOS-IC 4069
1x	LED 3mm, rot, Low Current
1x	LED 3mm, grün, Low Current
1x	NPN-Transistor BC548

	Verschiedenes
1x	Fotoplatine, einseitig
2x	Buchse AMP 520249-2
1x	Ultraschall-Sendekapsel 40 kHz
1x	Gehäuse Strapubox 85x50
1x	Ultraschallreflektor

Tabelle A.12: Komponentenliste für ein Leuchtfeuer

Die Schaltung wird wie in Abbildung A.37 bestückt. Die Platine ist durch die beiden Buchsen zu hoch für das Gehäuse, daher müssen im Boden Löcher für jede einzelne Lötstelle und sonstige Vertiefung der Platine gebohrt werden. Auf diese Weise liegt die Platine tiefer, zusätzlich muß noch der Deckel oberhalb der beiden Buchsen eingekerbt werden, damit die Schaltung in das Gehäuse paßt.

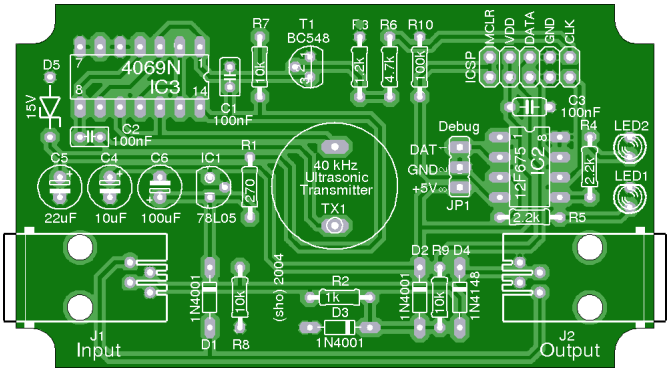


Abbildung A.37: Layout des Leuchtfeuers

Druckvorlagen auf der beiliegenden CD helfen dabei, die Löcher für die Ultraschallkapsel und die Befestigung des Reflektors zu bohren, sowie zwei Legoplatten der Größe 1x4 auf die Unterseite zu kleben. Durch die Legoplatten ist es möglich, das Leuchtfeuer einfach zu befestigen. Der Reflektor

wird so montiert, daß seine Spitze über der Mitte der Kapsel hängt. Die Höhe muß so gewählt werden, daß der Abstand von der Spitze zur Mitte der Kapsel 14 mm beträgt. Die Leuchtdioden werden im Gehäuse befestigt und mit Kabeln an die Platine angeschlossen. Beim Übertragen der Software muß auch der Kalibrierwert OSCCAL des PICs justiert werden, ein Testprogramm auf der beiliegenden CD existiert eigens für diese Aufgabe. Nach dem Brennen der Firmware `beacon.asm` wird die ICSP-Schnittstelle mit Jumpern geschlossen, dann ist das Leuchtfeuer einsatzbereit.

A.12 Verkabelung und Terminatoren

Die Steuereinheit und die Leuchtfeuer werden in einer Kette miteinander verbunden. Einzelne Kabelstücke verbinden die Output-Buchse einer Station mit der Input-Buchse der nächsten. Am Anfang der Kette befindet sich die Steuereinheit, in die letzte Output-Buchse wird ein Terminator gesteckt.

Abbildung A.38: Terminator für die Verkabelung der Leuchtfeuer

Die einzelnen Kabelstücke bestehen aus vieradrigem Telefonkabel mit RJ11-Steckern an beiden Enden. Das Kabel ist nicht gekreuzt, es werden also immer gleiche Kontakte von zwei Steckern miteinander verbunden. Leitung 1 transportiert die Versorgungsspannung und 3 das Kommunikationssignal, die beiden anderen Leitungen sind mit Masse belegt. Im Terminator werden alle Leitungen miteinander verbunden und dadurch auf Masse gelegt. Die Datenleitung wird direkt angeschlossen, bei der Leitung mit der Versorgungsspannung liegt noch ein Widerstand von 10 kΩ dazwischen.

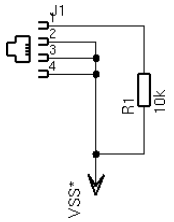


Abbildung A.39: Schaltung des Terminators

Der Widerstand wird direkt an das Kabelende gelötet und dann in einem Stück Schrumpfschlauch eingeschlossen. Die Verkabelung ist verpolungssicher. Bei falsch angeschlossenen Steckern funktioniert das System zwar nicht, es können aber auch keine Schäden an der Elektronik entstehen.

A.13 Sender für den RCX

Der Benutzer des Ortungssystems kann sich entscheiden, statt fester Leuchtfeuer einen Sender auf den RCX zu montieren. Dafür werden dann feststehende Empfänger in der Umgebung des Roboters aufgestellt. Für dieses Szenario existiert ein mobiler Sender, der vom RCX angesteuert werden kann

und auf Kommando Funk- und Ultraschallsignale senden kann. Die Übertragung der Kommandos erfolgt über die Stromversorgung durch Änderung der Pulsweitenmodulation.

Abbildung A.40: RCX-basierender Sender für das Ortungssystem

Die Schaltung ist weitgehend mit der eines Leuchtfeuers identisch, es kommen lediglich ein Funksender an GP2 und das RCX-Interface hinzu. Der PIC kann das PWM-Signal über GP0 abtasten, und die Bauteile im Umfeld von T_1 nutzen die Pulsweitenmodulation zum Antrieb einer Ladungspumpe aus, die eine Spannung von bis zu 15 V für den Ultraschallsender erzeugt.

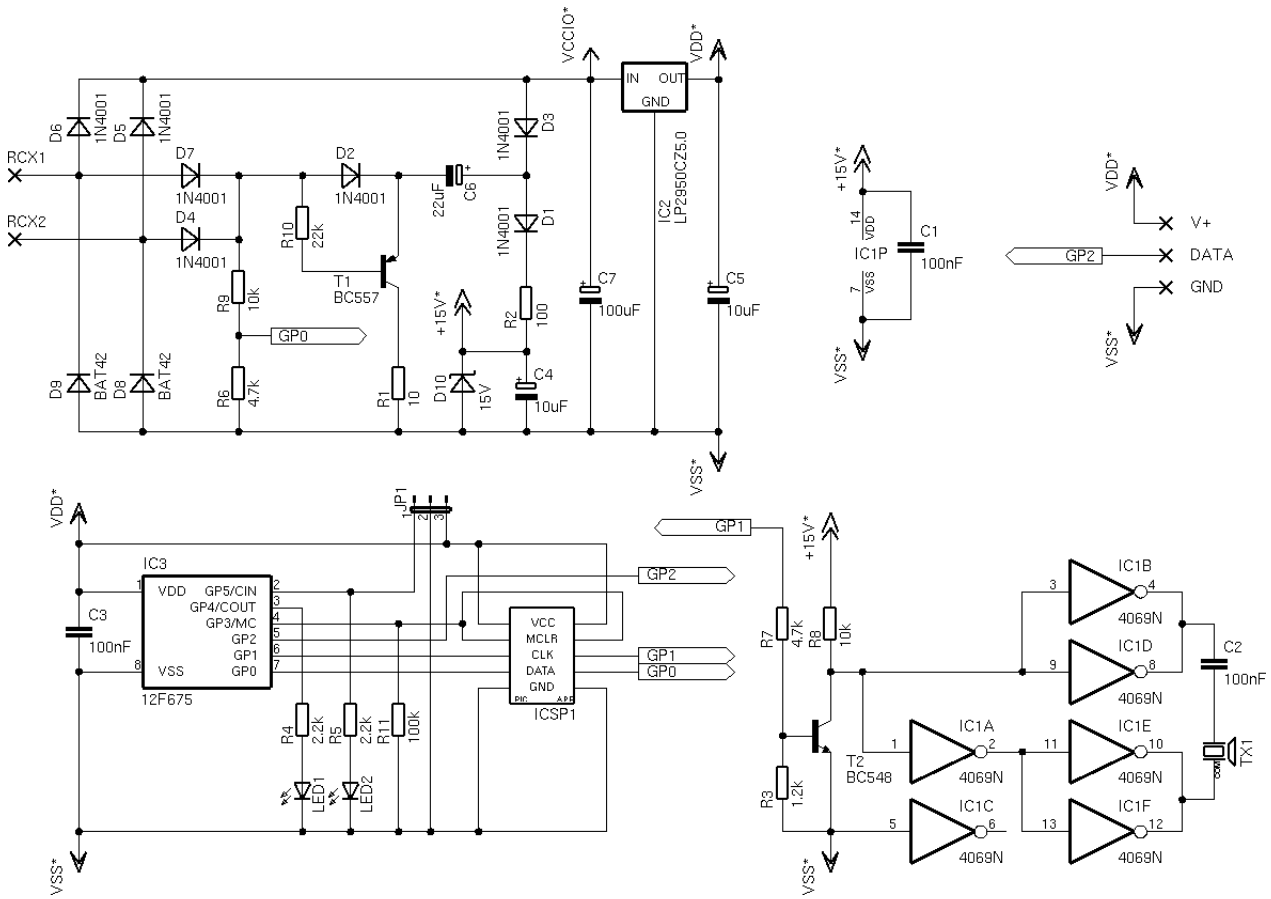


Abbildung A.41: Schaltung des Senders

	Widerstände
1x	10 Ω, 1/8 Watt
1x	100 Ω, 1/8 Watt
1x	1,2 kΩ, 1/8 Watt
2x	2,2 kΩ, 1/8 Watt
2x	4,7 kΩ, 1/8 Watt
2x	10 kΩ, 1/8 Watt
1x	22 kΩ, 1/8 Watt
1x	100 kΩ, 1/8 Watt

	Kondensatoren
3x	100 nF
2x	10 µF, 16 V, Elektrolyt
1x	22 µF, 16 V, Elektrolyt
1x	100 µF, 16 V, Elektrolyt

	Pfostenstecker 2.54 mm
1x	3 Stifte
1x	5x2 Stifte

	Halbleiter
7x	Diode 1N4001
2x	Schottky-Diode BAT42
1x	Zener-Diode 15 V
1x	CMOS-IC 4069
1x	Spannungsregler LP2950CZ5.0
1x	Mikrocontroller PIC16F628, DIL
1x	LED 3mm, rot, Low Current
1x	LED 3mm, grün, Low Current
1x	NPN-Transistor BC548
1x	PNP-Transistor BC557

	Verschiedenes
1x	Fotoplatine, einseitig
1x	Ultraschall-Sendekapsel 40 kHz
1x	Gehäuse Strapubox 85x50
1x	Halbes Mindstorms-Kabel
1x	Sendemodul 433 MHz, 5 V
1x	Ultraschallreflektor

Tabelle A.13: Komponentenliste für einen Sender

In das Gehäuse des Senders werden mit Hilfe der Bohrschablone die Löcher für die Sendekapsel und die Befestigung des Reflektors gebohrt und die Legoplatten auf der Unterseite angeklebt. Weitere Löcher für die Antenne und die beiden LEDs können nach Augenmaß ergänzt werden. Der Reflektor wird angebracht und justiert, wie es bereits beim Leuchtfeuer beschrieben wurde. Die Antenne besteht aus einem 17 cm langen Metallstab mit Gewinde an einem Ende. Sie wird mit zwei Muttern am Gehäusedeckel befestigt und mit einem kurzen Drahtstück mit dem Sendemodul verbunden.

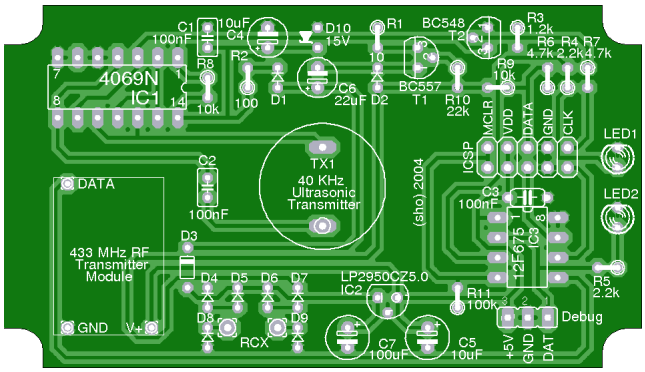


Abbildung A.42: Layout des mobilen Senders

Beide LEDs werden im Gehäuse verklebt und über Kabel mit der Platine verbunden. Nach dem Brennen der Firmware `rcxsender.asm`, wobei auch der OSCCAL-Wert justiert werden sollte, muß die ICSP-Schnittstelle mit Jumpern überbrückt werden. Kleine Streifen aus Schaumstoff können die Platine bei Bedarf so fixieren, daß sie sich im Gehäuse nicht bewegen kann. Das Mindstorms-Kabel wird über eine Aussparung nach außen geführt, die so schmal sein sollte, daß das Kabel darin fest eingeklemmt wird.

A.14 Empfänger für den RCX

Das Gegenstück zu den Leuchtfeuern und dem mobilen Sender bildet der Empfänger. Er fängt die Funk- und Ultraschallsignale auf und überträgt seine Meßdaten an einen RCX, von dem er auch mit Strom versorgt wird. Die Schaltung ist in Abbildung A.44 dargestellt.

Abbildung A.43: RCX-basierender Empfänger für das Ortungssystem

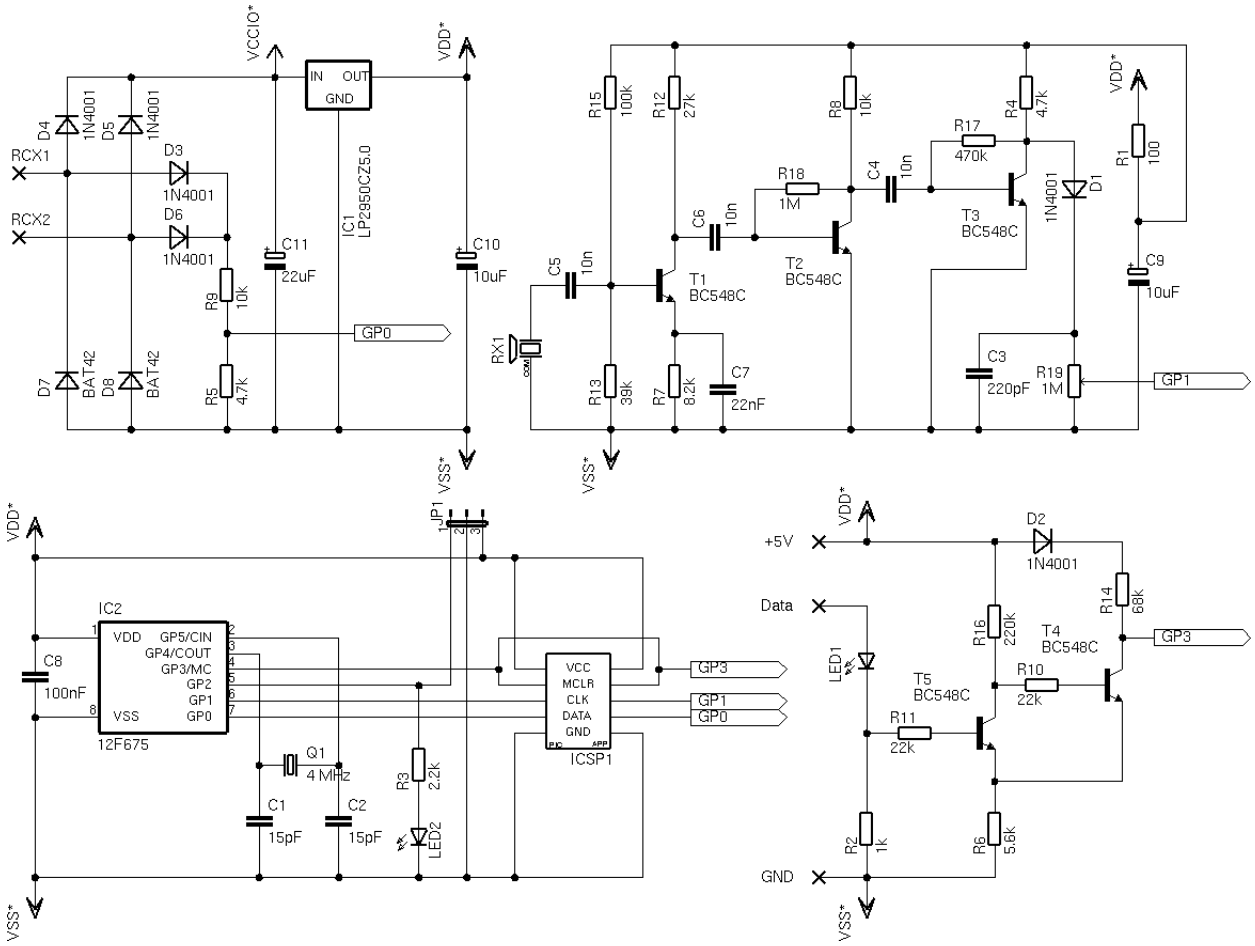


Abbildung A.44: Schaltung des Empfängers

Mittelpunkt der Schaltung ist der Mikrocontroller mit Quarz, Debug-Schnittstelle und ICSP-Interface. Das Sensorinterface liefert die Stromversorgung für die gesamte Schaltung, außerdem kann der PIC

an GP0 den Ausleserhythmus des RCX abtasten und den Meßwert beeinflussen. Der aus T_1 bis T_3 bestehende Verstärker nimmt das Signal von der Ultraschallkapsel auf und leitet es dem Detektor aus D_1, C_3 und R_{19} zu. Über das Poti kann die Empfindlichkeit eingestellt werden. Die Energieversorgung des Verstärkers wird über R_1 und C_9 von Schwankungen der Hauptversorgung abgekoppelt. T_4 und T_5 bilden einen Schmitt-Trigger, der die Signale vom Funkempfänger auf gültige Logikpegel bringt.

	Widerstände
1x	100 Ω, 1/8 Watt
1x	1 kΩ, 1/8 Watt
1x	2,2 kΩ, 1/8 Watt
2x	4,7 kΩ, 1/8 Watt
2x	5,6 kΩ, 1/8 Watt
2x	8,2 kΩ, 1/8 Watt
2x	10 kΩ, 1/8 Watt
2x	22 kΩ, 1/8 Watt
1x	27 kΩ, 1/8 Watt
1x	39 kΩ, 1/8 Watt
1x	68 kΩ, 1/8 Watt
1x	100 kΩ, 1/8 Watt
1x	220 kΩ, 1/8 Watt
1x	470 kΩ, 1/8 Watt
1x	1 MΩ, 1/8 Watt

	Kondensatoren
2x	15 pF
1x	220 pF
3x	10 nF
1x	22 nF
1x	100 nF
2x	10 µF, 6,3 V, Elektrolyt
1x	22 µF, 16 V, Elektrolyt

	Halbleiter
6x	Diode 1N4001
2x	Schottky-Diode BAT42
1x	Spannungsregler LP2950CZ5.0
1x	Mikrocontroller PIC16F628, DIL
1x	LED 3mm, rot, Low Current
1x	LED 3mm, grün, Low Current
5x	NPN-Transistor BC548C

	Pfostenstecker 2.54 mm
1x	3 Stifte
1x	5x2 Stifte

	Verschiedenes
1x	Poti 1 MΩ, stehend
1x	Fotoplatine, einseitig
1x	Quarz 4 MHz, HC49U
1x	Halbes Mindstorms-Kabel
1x	Ultraschall-Empfangskapsel 40 kHz
1x	Empfangsmodul 433 MHz, 5 V
1x	Gehäuse Strapubox 85x50
1x	Ultraschallreflektor
	Schaumstoff

Tabelle A.14: Komponentenliste für einen Empfänger

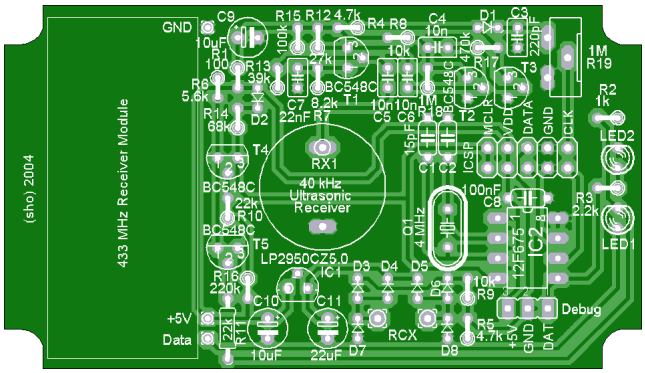


Abbildung A.45: Layout des Empfängers

Zunächst wird das Gehäuse des Empfängers wie das des Senders vorbereitet (bis auf die Antenne). Die Empfindlichkeit kann an zwei Stellen beeinflußt werden. Der Widerstand R_{17} bestimmt den

Verstärkungsfaktor für den Ultraschall. Setzt man hier weniger als die 470 kΩ ein, sinken die Reichweite und die Störanfälligkeit der Schaltung gleichermaßen. Das Potentiometer R_{19} dient dazu, die Schwelle des Detektors einzustellen. Zum Kalibrieren wird ein Sender in Betrieb genommen und so eingestellt, daß er regelmäßig bis zu 5 Impulse pro Sekunde sendet. Am einen Anschlag des Potis leuchtet die rote LED permanent, am anderen Anschlag nie. Das Optimum liegt genau zwischen diesen Extremen, die LED sollte bei jedem Impuls kräftig aufblitzen und danach komplett dunkel bleiben.

Zum Abschluß wird die Firmware `rcxreceiver.asm` in den PIC gebrannt und die ICSP-Schnittstelle mit Jumpern überbrückt. Ein Stück Schaumstoff wird so zurechtgeschnitten, daß es genau auf die Platine paßt und das Gehäuse möglichst gut füllt. Ein zweites dünnes Stück wird unter die Platine gelegt. Durch diese Verpackung ist der Empfänger einigermaßen gegen Erschütterungen gesichert. Die Wurfantenne des Empfängers wird oben auf dem Schaumstoff befestigt. Es können einige Versuche nötig sein, bis eine gut funktionierende Ausrichtung der Antenne gefunden ist.

A.15 Vibrationsdämpfer

Der Schaumstoff im Empfänger isoliert die Schaltung und insbesondere die Ultraschallkapsel relativ gut gegen Erschütterungen. In einigen Situationen kann es aber sein, daß der Schutz nicht ausreicht. Vibrationen, die zum Beispiel durch schnell drehende Mechanikteile des Roboters entstehen, können den Ultraschalldetektor empfindlich stören. In solchen Fällen kann ein spezieller Vibrationsdämpfer zwischen Roboter und Empfänger gebaut werden.

Abbildung A.46: Vibrationsdämpfer

Der Dämpfer besteht aus zwei Legoplatten, zwischen denen ein Stück Neopren mit reichlich Pattex befestigt wurde. Diese Methode ist zwar einfach, sie erweist sich aber als sehr effektiv. Eine spezielle Aufhängung und Dämpfung der Ultraschallkapsel im Empfängergehäuse hat beispielsweise deutlich schlechtere Ergebnisse geliefert.

A.16 Timer für UV-Belichter

Bei der Herstellung von Platinen, wie sie in Anhang D beschrieben wird, spielt die Belichtungsdauer eine entscheidende Rolle. Das Leiterbahnlayout wird mit Hilfe von UV-Licht von einer Vorlagenfolie auf die Platine übertragen. Das Resultat ist bei einer falschen Belichtungsdauer unter- oder überentwickelt, als Folge ist die Platine unbrauchbar. Eine Zeitschaltuhr kann dafür sorgen, daß die nötige Zeit genau eingehalten wird, auch wenn der Belichter unbeaufsichtigt ist. Abbildung A.47 zeigt einen solchen Timer. Über die Zifferntastatur kann die gewünschte Dauer eingegeben werden, mit der Taste * wird die eingestellte Zeit gelöscht und die Taste # startet und unterbricht den Vorgang. Das Gerät führt die entsprechende Belichtung autonom durch und schaltet die Leuchtstoffröhren pünktlich wieder aus. Ein LC-Display hilft bei der Eingabe und zeigt laufend den Status wie die noch verbleibende Restzeit

ein. Der Mikrocontroller speichert eine einmal eingegebene Belichtungszeit in seinem EEPROM und schlägt sie beim nächsten Mal wieder als Startwert vor.

Abbildung A.47: Zeitschaltuhr zum Steuern der Belichtungsdauer

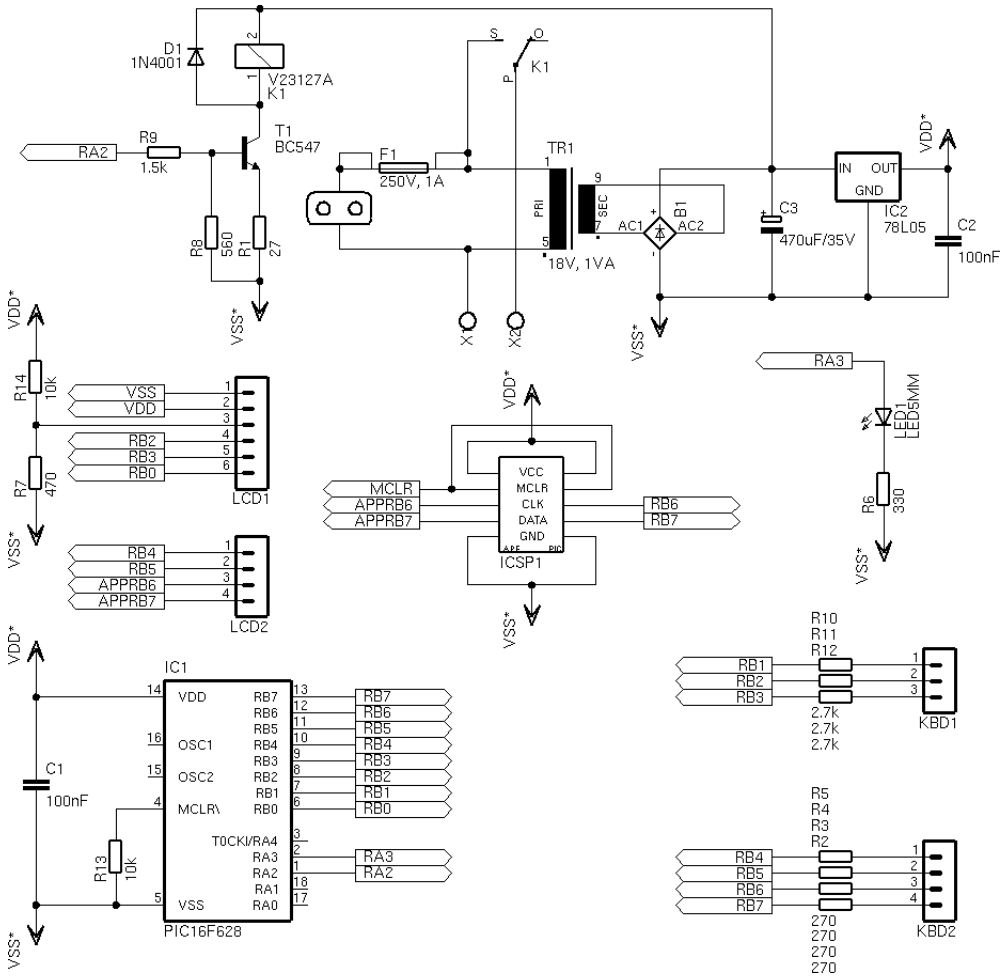


Abbildung A.48: Schaltung der Belichtungstimers

Die Schaltung ist relativ einfach aufgebaut. Ein Transformator liefert eine Spannung von 18 V für das Relais, daraus erzeugt ein 78L05 die 5 V für die Logik. Der PIC arbeitet mit dem internen Taktgenerator und steuert das Display, die Tastatur, eine LED und das Relais zum Schalten der Leuchtstoffröhren an. Details zur Programmierung von Display und Tastatur finden sich bei [Bredendiek].

	Widerstände
1x	27 Ω, 1/8 Watt
4x	270 Ω, 1/8 Watt
1x	330 Ω, 1/8 Watt
1x	470 Ω, 1/8 Watt
1x	560 Ω, 1/8 Watt
1x	1,5 kΩ, 1/8 Watt
3x	2,7 kΩ, 1/8 Watt
2x	10 kΩ, 1/8 Watt

	Halbleiter
1x	Diode 1N4001
1x	Mikrocontroller PIC16F628, DIL
1x	Spannungsregler 78L05
1x	LED 5mm, rot
1x	NPN-Transistor BC548
1x	Brückengleichrichter

	Kondensatoren
2x	100 nF
1x	470 µF, 35 V, Elektrolyt

	Pfostenstecker 2.54 mm
1x	5x2 Stifte

	Verschiedenes
1x	Fotoplatine, einseitig
1x	Buchse für Eurostecker, Printmontage
1x	Relais 250 V / 1 A, Spulenspannung 18 V
1x	Sicherungshalter (2 Clips)
1x	Sicherung 250 V, 1 A
1x	Transformator 18 V, 1 VA, Printmontage
2x	Anschlußklemme Wago 236
1x	LC-Display 2x16 Zeichen (HD44780)
1x	Zifferntastatur
1x	Gehäuse Teko TK22

Tabelle A.15: Komponentenliste für den Belichtungstimer

Die Leuchtstoffröhren werden an die Wago-Klemmen angeschlossen. Vor der Inbetriebnahme mit der Firmware `uvtimer.asm` muß die ICSP-Schnittstelle noch mit Jumpern überbrückt werden.

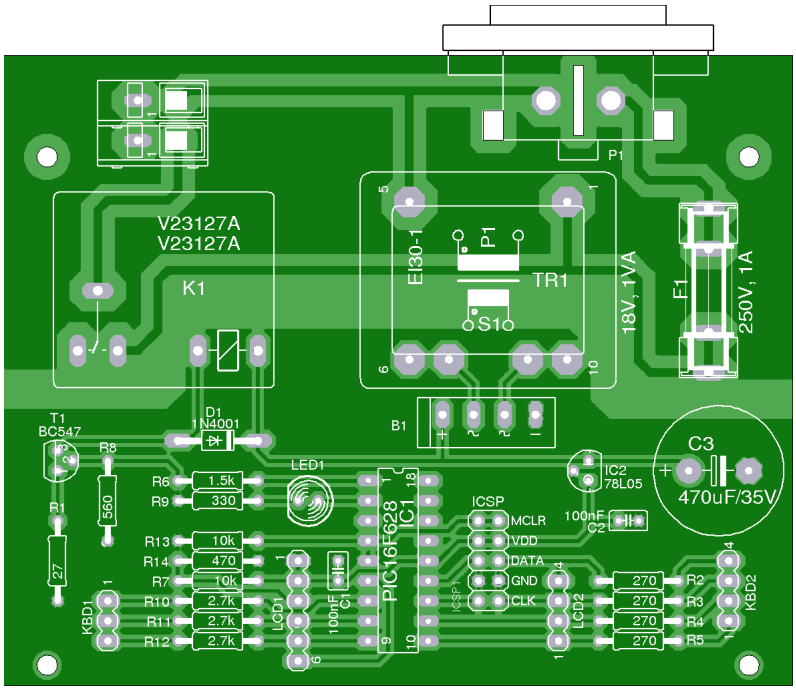


Abbildung A.49: Layout des Belichtungstimers

A.17 Joystick mit Infrarotsender

Zum Abschluß dieses Kapitels soll noch eine Schaltung erwähnt werden, die zwar im Rahmen dieser Arbeit entstanden aber nicht in das Resultat eingeflossen ist. Es handelt sich dabei um einen Joystick als Mindstorms-Fernbedienung, der in Abbildung A.50 zu sehen ist.

Abbildung A.50: Joystick als Mindstorms-Fernbedienung

Das analoge Modell war als Steuerknüppel für einen PC gedacht. Er enthielt eine Platine, die bei gedrückten Buttons Dauerfeuer simuliert hat. Diese ist durch eine Schaltung ersetzt worden, welche die Stellung des Knüppels und den Status der Buttons messen und per Infrarot an einen RCX senden kann. Als Protokoll kommt der Integrity Layer von brickOS zum Einsatz. Die eigentliche Schaltung ist in Abbildung A.51 dargestellt. Eine 9 V-Blockbatterie liefert die Energie für die Schaltung, sie wird von einem 78L05 auf 5 V reduziert und versorgt den Mikrocontroller. Der Takt wird durch einen Keramikresonator von 4 MHz vorgegeben. Die Pins RA0 und RA1 dienen zur Abfrage der Position des Joysticks, RA4 und RB1 sind mit den Buttons verbunden.

Die Bewegung des Knüppels in X- und Y-Richtung wird mechanisch an zwei Potentiometer weitergegeben. Die Kondensatoren C_1 und C_2 werden über je eines der Potentiometer aufgeladen. Je höher der Widerstand, desto langsamer steigt die Spannung über dem Kondensator an. Diesen Effekt nutzt der PIC zur Messung der Achsenstellung. Zur Abfrage der X-Achse wird Pin RA0 als Ausgang konfiguriert und auf Masse gelegt, was den Kondensator schlagartig entleert. Anschließend konfiguriert der PIC den Pin als analogen Eingang. Die Spannung über dem Kondensator steigt jetzt langsam an, bis sie eine gegebene Schwelle erreicht. Die dabei vergangene Zeit ist ein Maß für die Achsenstellung. Der selbe Vorgang wird anschließend für die Y-Achse wiederholt. Die mit den Buttons verbundenen Eingänge können wesentlich einfacher digital ausgelesen werden.

Der PIC stellt den Status des Joysticks in regelmäßigen Abständen fest. Die Ergebnisse werden in ein Datenpaket verpackt und über die beiden IR-Sendedioden an den RCX übermittelt. Dabei erzeugt die Software auch das nötige Trägersignal von 38 kHz. Der Aufbau eines Paketes ist einfach, es enthält in den Nutzdaten nur den Buchstaben J gefolgt von zwei Datenbytes D_0 und D_1. Das oberste Bit von D_0 ist gesetzt, wenn Button 1 gedrückt ist, der Rest beschreibt die X-Koordinate. Bit 6 ist gesetzt, wenn der Knüppel sich links von der Mitte befindet, die untersten drei Bits geben die Auslenkung in die jeweilige Richtung als Zahl zwischen 0 und 7 an. D_1 beschreibt auf die selbe Weise den Status von Button 2 und die Y-Auslenkung. Bit 6 ist gesetzt, wenn der Knüppel sich oberhalb der Mitte befindet.

Es wird etwa alle 150 ms ein Datenpaket gesendet. Die rote LED leuchtet während dieser Zeit, und zwar vom Auslesen des Status bis zum Senden des letzten Datenbits. Der Joystick kann über einen Hauptschalter eingeschaltet werden, als Button 1 werden der Knopf an der Spitze des Knüppels sowie der vordere der beiden großen Knöpfe interpretiert.

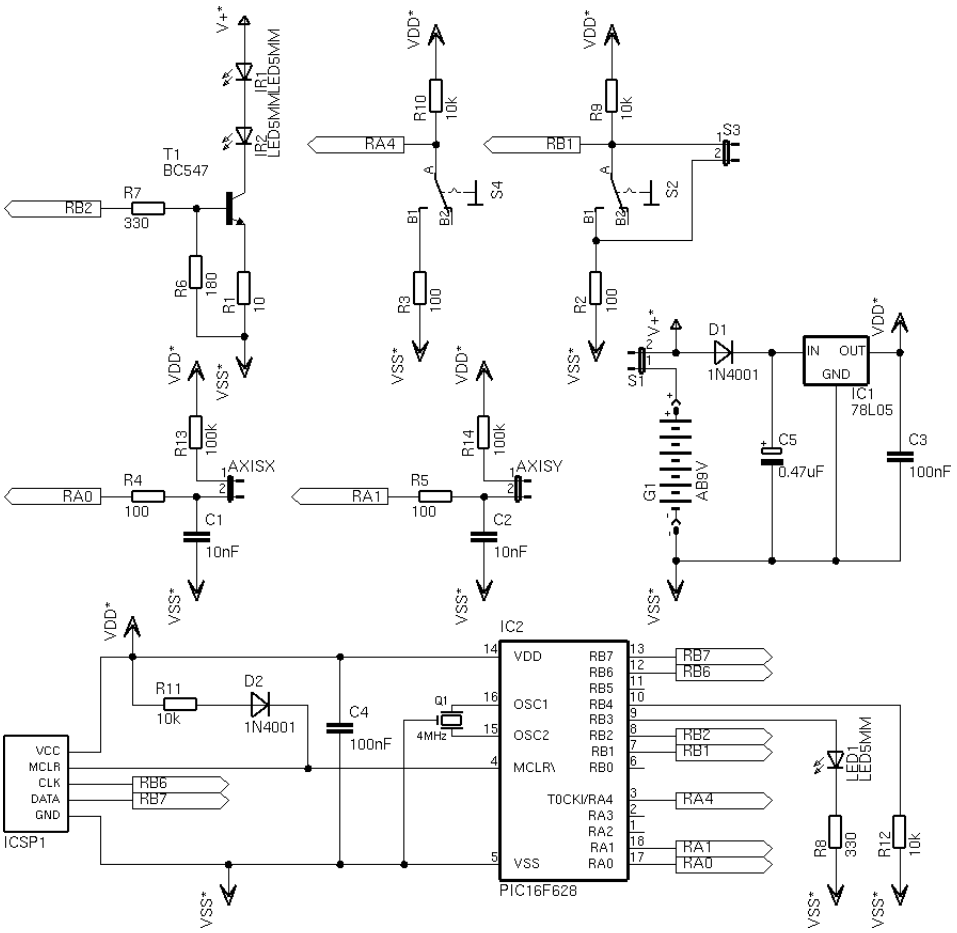


Abbildung A.51: Schaltung der Joysticks

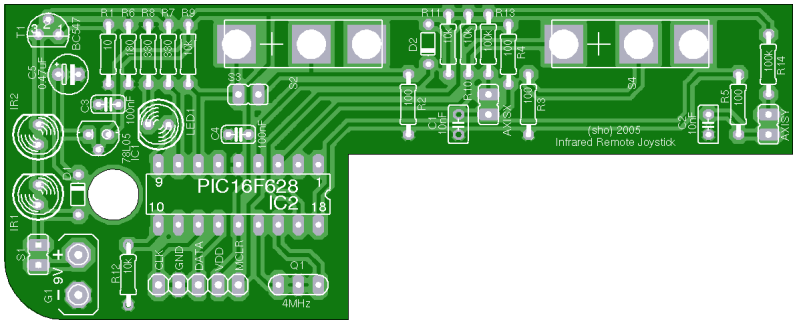


Abbildung A.52: Layout des Joysticks

Für die Umrechnung der Koordinaten muß der PIC den Wertebereich für beide Achsen und die Position der Mittelstellung kennen. Diese werden durch eine Kalibrierung gemessen und im EEPROM gespeichert, so daß nur bei Bedarf neu kalibriert werden muß. Dazu werden beide Buttons gedrückt

gehalten und der Joystick eingeschaltet. Die rote LED leuchtet jetzt ohne Unterbrechungen. Der Knüppel wird möglichst weit außen im Kreis gedreht und dann in die Mittelstellung gebracht. Jetzt werden beide Buttons losgelassen, der Joystick ist kalibriert und beginnt normal mit dem Senden.

	Widerstände
1x	10 Ω, 1/8 Watt
4x	100 Ω, 1/8 Watt
1x	180 Ω, 1/8 Watt
2x	330 Ω, 1/8 Watt
4x	10 kΩ, 1/8 Watt
2x	100 kΩ, 1/8 Watt

	Kondensatoren
2x	10 nF
2x	100 nF
1x	0,47 µF, 6,3 V, Elektrolyt

	Pfostenstecker 2.54 mm
1x	5x1 Stifte, halbe Höhe

	Halbleiter
2x	Diode 1N4001
1x	Spannungsregler 78L05
1x	Mikrocontroller PIC16F628, DIL
1x	LED 5mm, rot
2x	IR-Sendediode TSAL-6200
1x	PNP-Transistor BC548

	Verschiedenes
1x	Fotoplatine, einseitig
1x	Batterieclip 9 V
1x	Keramikresonator 4 MHz
1x	Gravis Analog-Joystick
1x	Kippschalter

Tabelle A.16: Komponentenliste für Joystickumbau

Die Platine kann bis auf die beiden Mikroschalter bestückt werden. Im Gehäuse müssen größere Teile weggefräst und einige Bohrungen angebracht werden, bis die Platine und die Batterie problemlos hineinpassen. Die Halterung der beiden Mikroschalter muß intakt bleiben, um die Platine fixieren zu können. Mehrere Kabel stellen später die nötigen Verbindungen her.

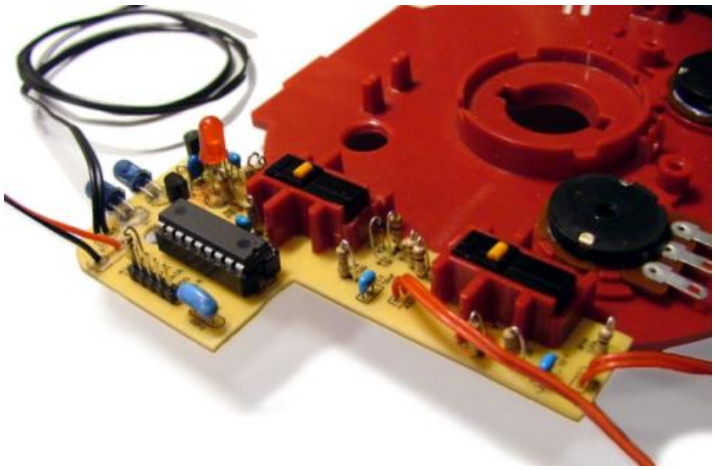


Abbildung A.53: Einbau der Platine in den Joystick

Wenn die Teile mechanisch zusammenpassen, wird die Platine in die endgültige Position gebracht und die Mikroschalter werden eingelötet. Das selbe passiert mit den Kabeln zu den Schaltern, der LED und den beiden Achsenpotis. Die Kabel können teilweise mit Heißkleber befestigt und gesichert werden, was vor allem das spätere Schließen des Gehäuses deutlich vereinfacht. Zu diesem Zeitpunkt ist auch ein gründlicher Funktionstest sinnvoll, das Steuerprogramm ist `joystick.asm` und die übertragenen Werte lassen sich zum Beispiel mit `lnpdump` aus Anhang B.2 darstellen.

Abbildung A.54 zeigt den kompletten Joystickumbau. Die Schaltung arbeitet mit einer Batterie mehrere Stunden lang und kann einen Roboter wie das Modell Roadster fernsteuern, dessen Bauanleitung sich unter [Legokiel] befindet. Ein geeignetes Steuerprogramm ist zum Beispiel `roadster.c`.

Abbildung A.54: Fertig zusammengebauter Joystick

Anhang B

Erstellte Programme

Im Rahmen dieser Arbeit wurde auch einiges an Software entwickelt. In erster Linie sind dies alle Programme, die für das Ortungssystem gebraucht werden. Daneben ist aber auch eine Reihe von Entwicklungswerkzeugen entstanden, die im Folgenden kurz vorgestellt und erklärt werden sollen.

B.1 Programmierung des Mindstorms-Towers

Die Tatsache, daß der RCX über Infrarotsignale mit seiner Umgebung kommunizieren kann, ermöglicht eine Vielzahl von interessanten Anwendungen. Beispielsweise kann eine Gruppe von Robotern als Team agieren und miteinander kommunizieren. Ein Roboter kann Telemetriedaten oder Meßwerte über seine Umgebung an einen PC senden, während er diese erkundet. Der PC ist auch in der Lage, Steuerbefehle an einen RCX senden. Bei einigen Robotern wird so die Datenübertragung zu einem zentralen Bestandteil der Funktionalität.

BrickOS bietet für die Programmierung des RCX eine komplette Schnittstelle zum Transceiver, die das Senden und Empfangen von Daten in allen gängigen Protokollen abdeckt. Auf der PC-Seite ist die Situation völlig anders. Die Programme zum Übertragen von Kernel und Anwendungen steuern den Tower direkt an, es gibt aber keine Programmierschnittstelle für eigene Anwendungen. Diese Lücke soll mit einem externem Programm, dem Lego Network Protocol Daemon (lnpd) geschlossen werden. Sein Einsatz ist aber mit zahlreichen Problemen verbunden, und einen adäquaten Ersatz gab es bisher nicht. Dieser Abschnitt erklärt die Gründe für die Probleme und stellt eine neue, in C geschriebene Bibliothek für Linux, Solaris und Cygwin vor. Sie erlaubt es, sowohl den seriellen als auch den USB-Tower direkt anzusteuern.

B.1.1 Erkennen und Vermeiden von Kollisionen

Alle in einem Raum befindlichen Transceiver senden auf der selben Trägerfrequenz im selben Medium. Damit besteht die Gefahr von Kollisionen: senden zwei Stationen zur selben Zeit, überlagern sich ihre Signale und sind für keinen Empfänger mehr zu trennen. Eine wichtige Voraussetzung bei der Ansteuerung eines Transceivers ist also, Kollisionen möglichst sicher zu verhindern. Die Zugriffskontrolle läßt sich durch zwei einfache Regeln realisieren, wie sie beispielsweise bei Ethernet zu finden sind (dort bilden sie die Grundlage des CSMA/CD-Verfahrens).

- Keine Station darf mit dem Senden beginnen, während eine andere noch sendet (Vermeidung).
- Sollten zwei Stationen gleichzeitig mit einer Übertragung beginnen, müssen beide dies erkennen und den Sendevorgang abbrechen. Ein neuer Versuch darf erst nach einer zufälligen Wartezeit unternommen werden (Erkennen von Kollisionen).

BrickOS setzt diese Regeln direkt in seinem Programmcode zur Steuerung des Transceivers um.

- Wenn der RCX ein Byte empfängt, wird ein Timer auf seinen Anfangswert gesetzt und zählt dann auf 0 herunter. Alle Sendebefehle bleiben blockiert, bis der Timer 0 erreicht hat. Der Startwert ist dabei so gewählt, daß in der Zeit mehrere Bytes übertragen werden könnten. Auf diese Art wird mit hoher Wahrscheinlichkeit verhindert, daß ein bereits laufender Sendevorgang gestört werden kann.
- Zum Erkennen von Kollisionen während des Sendens muß der RCX die von ihm gesendeten Daten verifizieren. Dafür läßt sich die Tatsache ausnutzen, das jedes gesendete Infrarotsignal im Raum reflektiert wird und zum Transceiver zurückkehrt. BrickOS muß nur nach dem Senden eines Bytes den Empfänger abfragen und den empfangenen mit dem gesendeten Wert vergleichen. Stimmen sie nicht überein, oder meldet der Empfänger einen Fehler, ist die Ursache wahrscheinlich eine Kollision. BrickOS bricht in so einem Fall den Sendevorgang sofort ab und sperrt den Transceiver für eine zufällige Zeit. Ein praktischer Nebeneffekt hierbei ist, daß der RCX von ihm gesendete Pakete im Datenstrom des Empfängers erkennen und entfernen kann.

Die Algorithmen zur Vermeidung und Erkennung von Kollisionen funktionieren nur, wenn sie die volle Kontrolle über das Timing des Transceivers haben. Für jedes empfangene Byte muß die Zeit bekannt sein, wann es den Transceiver erreicht hat, beziehungsweise wann es gesendet wurde. Ist diese Zuordnung nicht möglich, funktionieren die Algorithmen nicht mehr sicher. Beim RCX ist diese Anforderung erfüllt, da brickOS die Hardware des Senders auf unterster Ebene steuern kann.

B.1.2 Übertragbarkeit der Algorithmen auf den PC

Ob sich die beschriebenen Algorithmen zur Kollisionsvermeidung auch auf einen PC realisieren lassen, hängt von den Eigenschaften der seriellen Schnittstelle und des Betriebssystem ab. In älteren PCs wurden Schnittstellenbausteine vom Typ 16450 eingesetzt, deren Verhalten bei der Datenübertragung weitgehend mit dem des RCX übereinstimmt. Der Baustein leitet Daten in beide Richtungen ohne Verzögerung weiter und löst bei jedem Byte einen Interrupt aus. Die Kollisionsvermeidung funktioniert auf dieser Grundlage problemlos, ganz im Gegensatz zum Nachfolgebaustein 16550.

Der große Nachteil des 16450 ist, daß er sich nur schlecht für hohe Geschwindigkeiten eignet. Die hohe Interruptfrequenz und das Fehlen von Puffern stellen zu hohe Anforderungen an die Latenzzeit des Betriebssystems. Jedes empfangene Byte muß ausgelesen werden, bevor ein weiteres eintrifft, sonst wird es von diesem überschrieben. Der Nachfolger 16550 verfügt daher über zwei Puffer mit einer Größe von je 16 Bytes. Zu sendende Daten werden als Block in einen der Puffer geschrieben von dem Controller der Reihe nach gesendet. Empfangene Daten werden in dem anderen Puffer gesammelt, bis dieser einen programmierbaren Füllstand erreicht hat oder einige Zeit lang keine neuen Daten eingetroffen sind. Erst dann wird ein Interrupt ausgelöst (vergleiche [Tischer]). Diese Optimierungen verbessern das Verhalten des Bausteins im normalen Anwendungsfall deutlich. Dies geschieht allerdings auf Kosten der Echtzeitfähigkeit, die aber gerade für die Kollisionsvermeidung gebraucht wird.

Beispielsweise kann eine Anwendung nicht feststellen, ob gerade eine Übertragung stattfindet. Eine laufende Übertragung könnte schon dabei sein, den Empfangspuffer zu füllen, ohne daß die Anwendung dies mitbekommt. Genauso unmöglich ist es, die Integrität der gesendeten Daten zu überprüfen. Die Anwendung weiß nicht, wann ein zu sendendes Byte den Transceiver verläßt, wann es vom Empfangspuffer an die Anwendung zurückgegeben wird wie viele Daten von anderen Stationen dazwischen geraten können. Im Falle einer Kollision können ganze Bytes verfälscht oder ausgelassen werden, so daß es gänzlich ausgeschlossen ist, gesendete Daten und ihr Echo eindeutig einander zuzuordnen.

Die Konsequenz aus beiden Problemen ist, daß es auf einem PC mit einer seriellen Schnittstelle vom Typ 16550 völlig unmöglich ist, Kollisionen auf der Ebene des Physical Layers zu erkennen. Es kann

auch nicht sichergestellt werden, daß das Echo eines gesendeten Paketes erkannt und unterdrückt wird. Von den gängigen Betriebssystemen sind nur Windows und Linux in der Lage, eine serielle Schnittstelle in den 16450-Kompatibilitätsmodus zu schalten. Ein portables Programm muß sich daher an den Schwächen des 16550 orientieren und die Probleme der Kollisionsvermeidung und Integritätssicherung auf höhere Protokollebenen verlagern.

B.1.3 LegOS Networking Protocol Daemon

Der lnpd wurde 1999 etwa zeitgleich mit der Versionen 0.2.2 des brickOS-Vorgängers legOS veröffentlicht und sollte es dem Benutzer ermöglichen, den Tower aus eigenen Anwendungen heraus zu steuern. Der Daemon wird auf einem beliebigen Rechner installiert und steuert einen lokal angeschlossenen seriellen Tower. Ein Anwendungsprogramm benutzt die dazugehörige Bibliothek liblnp, um eine TCP/IP-Verbindung zum Daemon aufzubauen und über diesen Pakete zu senden und zu empfangen.

Der lnpd wurde für ältere serielle Schnittstellen vom Typ 16450 entwickelt und benutzt die selben Algorithmen wie brickOS zur Kollisionsvermeidung. Dies führt bei neueren Rechnern zu einer sehr instabilen Datenübertragung, insbesondere werden Pakete um so wahrscheinlicher verworfen, je größer sie sind. Auf vielen Systemen liegt die kritische Grenze bereits bei 8 Bytes. Beim Senden werden Kollisionen häufig zu Unrecht erkannt, sobald Bandbreite von anderen Stationen belegt wird.

Die Paketverluste beim Empfänger lassen sich reduzieren, indem die entsprechenden Timeouts deutlich erhöht werden. Dies verschlechtert jedoch die Situation des Senders noch weiter, und die übrigen Probleme bleiben davon ebenfalls unberührt. Der lnpd ist auch nicht mehr kompatibel zu neueren Versionen von BrickOS, da ab Version 0.2.5.2 das serielle Übertragungsprotokoll mit ungerader Parität arbeitet, während bei älteren Versionen keine Parität benutzt wurde. Unterstützung für den USB-Tower bietet der lnpd auch nicht, so daß die Entwicklung einer Alternative gerechtfertigt ist.

B.1.4 Die lnphost-Bibliothek

Wie die vorigen Abschnitte gezeigt haben, können die Algorithmen zur Vermeidung von Kollisionen sowie zum unterdrücken des Echos gesendeter Pakete in einem portablen Programm nicht funktionieren. Daher ist es sinnvoll, diese Aufgaben der übergeordneten Anwendung zu übertragen. Ihr Pakethandler kann Echo-Pakete verwerfen oder als Bestätigung für gesendete Pakete interpretieren. Außerdem hat die Anwendung Kenntnisse über den globalen Ablauf der Kommunikation und kann die Sendezeitpunkte so wählen, daß Kollisionen möglichst unwahrscheinlich sind. Dieser Ansatz bildet die Grundlage der Bibliothek `lnphost`, mit deren Hilfe Programme einen an den Rechner angeschlossenen Tower steuern können. Darüber hinaus zeichnet sie sich durch eine Reihe von Funktionen aus.

- Der serielle Tower wird unter allen Posix-konformen Plattformen unterstützt. Ob der USB-Tower angesteuert werden kann, hängt von der Unterstützung des Betriebssystems ab. Momentan funktioniert er nur unter Linux und Cygwin.
- In einem Programm können beliebig viele Tower unabhängig voneinander angesteuert werden.
- Das Lego-Protokoll, Integrity Layer und Addressing Layer von brickOS werden voll unterstützt und können auch gleichzeitig eingesetzt werden. Daneben können auch ungefilterte Rohdaten gesendet und empfangen werden. Zum Senden können entsprechende Routinen aufgerufen werden, das Empfangen von Paketen funktioniert über frei definierbare Callback-Funktionen.
- Die Pakethandler im brickOS Addressing Layer können für jede beliebige Adresse installiert werden und sind nicht an eine statische Netzmaske oder Hostadresse gebunden.
- Alle Versionen von legOS und brickOS werden unterstützt. Es muß nur konfiguriert werden, ob das Paritätsbit benutzt werden soll.

- Im Gegensatz zum lnpd bietet `lnphost` selber keine Netzwerkfunktionalität. Es ist aber einfach möglich, einen Daemon zu programmieren, der `lnphost` zur Ansteuerung des Towers verwendet. Diese Trennung der Aufgaben ist sinnvoll, weil sie es ermöglicht, auf die meistens unnötige Netzwerkfunktionalität zu verzichten.

Die Bibliothek installiert beim Aktivieren eines Towers zwei Threads. Einer ist beim seriellen Tower für die Keepalive-Funktion zuständig. Er sendet nach 4 Sekunden Inaktivität des Senders ein Byte mit dem Wert 255, um den Tower empfangsbereit zu halten. Der andere Thread sucht in dem empfangenen Datenstrom nach gültigen Paketen. Dafür benutzt er eine Heuristik, die Pakete aus allen unterstützten Layern erkennen kann. Das benutzte Verfahren ist darauf ausgelegt, möglichst viele gültige Pakete zu erkennen und erreicht dabei eine sehr hohe Zuverlässigkeit. Die erkannten Pakete werden an die entsprechenden Callback-Funktionen der Anwendung weitergereicht. Zum Senden von Paketen werden die Daten mit Header und Checksumme versehen und direkt an den Tower weitergegeben.

Arbeitsweise der Heuristik

Alle empfangenen Daten werden von `lnphost` zunächst in einem Puffer gesammelt, bis darin ein gültiges Paket zusammengesetzt wurde. Wann ein vollständiges Paket enthalten ist, wird nach dem folgenden Algorithmus entschieden.

- Der Empfänger fragt laufend den Tower nach neuen Bytes ab. Wurde eines empfangen, wird es in den Puffer eingefügt. Anderenfalls ist es zu einem Timeout gekommen, die letzte Übertragung ist dann definitiv beendet.
- In beiden Fällen startet anschließend der Paketdetektor. Er ruft Hilfsfunktionen auf, die überprüfen, ob am Anfang des Puffers ein gültiges Paket in einem der verschiedenen Formate liegt. Die Hilfsfunktionen können darauf mit *Ja*, *Nein* und *Vielleicht* antworten. Als Entscheidungshilfe bekommen die Hilfsfunktionen noch die Information, ob noch weitere Daten zu erwarten sind oder ob es zu einem Timeout gekommen ist.
- Antwortet eine der Funktionen mit *Ja*, wird das Paket ausgewertet und aus dem Puffer entfernt.
- Antworten alle Funktionen mit *Nein*, wird das erste Byte auf dem Puffer entfernt und die Suche mit den Restdaten wiederholt.
- In allen anderen Fällen wartet der Empfänger auf weitere Daten oder ein Timeout. Wenn es zu einem Timeout kommt, müssen die Hilfsfunktionen eine eindeutige Entscheidung treffen.

Es ist relativ einfach, ein korrektes brickOS-Paket zu erkennen. Dazu müssen lediglich überprüft werden, ob der Header korrekt ist, die Nutzlast vorhanden sind und die Checksumme stimmt. Bei dem Lego-Paketformat ist die Überprüfung schwieriger, weil die Länge des Paketes im allgemeinen unbekannt ist. Hier läßt sich die Tatsache ausnutzen, daß ein Paket aus Paaren von einem Byte und seinem Komplement besteht, vor denen jedoch ein einzelnes Byte mit dem Wert 0x55 gesendet wird. Die Heuristik erwartet bei einem Lego-Paket also, daß es mit einem gültigen Header beginnt. Danach werden gültige Bytepaare angefügt, bis es zu einem Timeout kommt oder ein ungültiges Bytepaar entdeckt wird. Das letzte Bytepaar wird dann als Checksumme interpretiert und überprüft. Das einzelne Byte 0x55 am Anfang des folgenden Paketes wird also immer als Grenze zwischen zwei Paketen interpretiert. Die anderen Protokolle benutzen keine Komplemente und werden damit auch nicht als Verlängerung eines Lego-Paketes interpretiert. Trotzdem kann es in seltenen Fällen im Mischbetrieb mehrerer Protokolle dazu kommen, daß Lego-Pakete nicht richtig erkannt werden. Im Rahmen der normalen Request-Reply-Kommunikation, wie sie für das Lego-Protokoll vorgesehen ist, funktioniert die Heuristik aber zuverlässig.

Überblick über das API

Die Bibliothek `lnphost` stellt die folgenden Routinen zur Verfügung. Die I/O-Funktionen geben eine Null zurück, wenn sie fehlerfrei ausgeführt werden konnten.

```
int lnp_open(tower, device, flags);
void lnp_close(tower);
int lnp_set_range(tower, range);

void lnp_raw_set_handler(tower, handler);
void lnp_lego_set_handler(tower, handler);
void lnp_integrity_set_handler(tower, handler);
void lnp_addressing_set_handler(tower, address, handler);
void lnp_addressing_set_multi(tower, mask, value, handler);

void lnp_block(tower);
void lnp_unblock(tower);

int lnp_raw_send(tower, data, unsigned length);
int lnp_lego_send(tower, data, length);
int lnp_integrity_send(tower, data, length);
int lnp_addressing_send(tower, data, length, dest, src);

unsigned char lnp_addressing_host(mask, address);
unsigned char lnp_addressing_port(mask, address);
unsigned char lnp_addressing_addr(mask, host, port);

void lnp_hexdump(prefix, data, length);
```

Die einzelnen Funktionen sollen in den nächsten Abschnitten ausführlich vorgestellt werden.

Initialisierung des Towers

Die Funktion `lnp_open` öffnet das Device, an dem der Tower angeschlossen ist, und konfiguriert ihn entsprechend. Gelingt dies, werden auch die Threads für den Keepalive-Mechanismus und den Datenempfang aktiviert. Die Datenstruktur `tower` speichert alle Einstellungen des Towers und dient als eine Art Handle, die allen anderen Funktionen übergeben werden muß.

```
int lnp_open(struct lnptower *tower, char *device, unsigned flags);
```

Der Parameter `device` gibt den Namen des Devices an, die Dokumentation listet eine Reihe von Beispielen für die verschiedenen Plattformen auf. Wird statt eines Namens `NULL` übergeben, wertet die Funktion stattdessen die Umgebungsvariable `RCXTTY` aus und weicht, falls diese auch nicht definiert sein sollte, auf einen plattformspezifischen Standardwert aus (`LNP_DEFAULTDEVICE`). Die Bitmaske `flags` kann aus den folgenden Konstanten bestehen:

- `LNP_FAST` erhöht die Geschwindigkeit von 2400 auf 4800 Baud (momentan nur serieller Tower)
- `LNP_NOPARITY` schaltet auf ein Protokoll ohne Paritätsbit um (legOS-Kompatibilitätsmodus)
- `LNP_NOKEEPALIVE` deaktiviert den Keepalive-Mechanismus des Towers (Standard bei USB)

Die Routine `lnp_close` schließt die Verbindung zum Tower und beendet die beiden Threads.

```
void lnp_close(struct lnptower *tower);
```

Mit `lnp_set_range` kann die Reichweite des USB-Towers in drei Schritten verändert werden. Gültige Werte für `range` sind `LNP_LOWRANGE`, `LNP_MIDRANGE` und `LNP_HIGHRANGE`.

```
int lnp_set_range(struct lnptower *tower, int range);
```

Diese Funktion ist noch nicht implementiert, da das Linux-Modul die Funktion noch nicht unterstützt. Beim seriellen Tower kann die Reichweite außerdem gar nicht per Software geändert werden. Daher wird immer ein von 0 verschiedener Wert zurückgegeben.

Handler für den Paketempfang

Die Bibliothek `lnphost` unterstützt insgesamt 4 Arten von Pakethandlern für Rohdaten, Legopakete sowie Integrity und Addressing Layer von brickOS. Ein Handler wird aufgerufen, wenn ein für ihn passendes Datenpaket empfangen wurde (auch, wenn es vom selben Tower aus gesendet wurde). Ihm wird dabei eine Kopie des Paketinhaltes übergeben.

```
typedef void (*lnp_raw_handler_t) (unsigned char *data, unsigned length,
                  int isvalid);
typedef void (*lnp_lego_handler_t) (unsigned char *data, unsigned length);
typedef void (*lnp_integrity_handler_t)(unsigned char *data,
                  unsigned char length);
typedef void (*lnp_addressing_handler_t) (unsigned char *data,
                  unsigned char length, unsigned char source, unsigned char dest);
```

Die Handler dürfen alle Funktionen des Systems nutzen, müssen nur beachten, daß sie nebenläufig zum Hauptprogramm ausgeführt werden. Außerdem ist der Empfangsmechanismus während ihrer Ausführung blockiert, so daß der Handler sich beenden muß, bevor der Empfangspuffer des Betriebssystems überläuft. Die folgenden Routinen können benutzt werden, um eigene Handler zu installieren. Dabei kann immer `LNP_NOHANDLER` anstelle des Namens einer Routine übergeben werden, um den betreffenden Handler zu deaktivieren.

```
void lnp_raw_set_handler(struct lnptower *tower, lnp_raw_handler_t handler);
void lnp_lego_set_handler(struct lnptower *tower, lnp_raw_handler_t handler);
void lnp_integrity_set_handler(struct lnptower *tower,
                           lnp_integrity_handler_t handler);
void lnp_addressing_set_handler(struct lnptower *tower, unsigned char address,
                                lnp_addressing_handler_t handler);
```

Der Handler für die Rohdaten bekommt alle empfangenen Daten ungefiltert übergeben, einschließlich der Bytes vom Keepalive-Mechanismus. Der Handler kann für einzelne Bytes oder ganze Blöcke aufgerufen werden. Insbesondere werden gültige Pakete an diesen Handler übergeben, bevor die Nutzlast an einen der protokollspezifischen Handler weitergereicht wird. In diesem Fall ist der Parameter `isvalid` auf einen Wert ungleich 0 gesetzt, um anzuzeigen, daß die Daten ein gültiges Paket darstellen.

BrickOS und lnpd verwenden im Addressing Layer Adressen mit einer Breite von 8 Bit, die sie in eine Hostadresse und eine Portnummer aufteilen. Der Benutzer kann nur die Portnummer wählen, der Host ist implizit gegeben. Damit ist es unmöglich, Handler für zwei verschiedene Hostnummern zu installieren. Aus diesem Grund verwendet `lnphost` die kompletten Adressen und bietet Umrechnungsfunktionen für Host und Portnummer an, die weiter unten vorgestellt werden.

Wenn ein Programm einen Handler für eine ganze Reihe von Adressen setzen soll, bietet sich die Funktion `lnp_addressing_set_multi`an. Der übergebene Handler wird auf allen Adressen installiert, die mit `mask` und-verknüpft `value` ergeben.

```
lnp_addressing_set_multi(struct lnptower *tower, unsigned char mask,
                unsigned char value, lnp_addressing_handler_t handler);
```

Mit den folgenden beiden Funktionen kann der Callback-Mechanismus zeitweise gesperrt werden.

```
void lnp_block(struct lnptower *tower);
void lnp_unblock(struct lnptower *tower);
```

Der Empfänger arbeitet unabhängig davon weiter und stellt Pakete zusammen. Es wird nur kein Handler aufgerufen, wenn ein Paket komplett ist. Das Blockieren löscht aber direkt keine Daten, es leert insbesondere nicht den Empfangspuffer.

Senden von Daten

Die hier vorgestellten Funktionen kapseln die übergebenen Daten in Pakete des jeweiligen Formates ein und verschicken sie unmittelbar danach über den Tower. Die einzige Ausnahme bildet `lnp_raw_send`. Diese Routine sendet die übergebenen Daten unverändert.

```
int lnp_raw_send(struct lnptower *tower, unsigned char *data, unsigned length);
int lnp_lego_send(struct lnptower *tower, unsigned char *data,
                            unsigned char length);
int lnp_integrity_send(struct lnptower *tower, unsigned char *data,
                            unsigned char length);
int lnp_addressing_send(struct lnptower *tower, unsigned char *data,
                            unsigned char length,
                            unsigned char dest, unsigned char src);
```

Adreßberechnungen

Im Addressing Layer von brickOS wird jedes Paket mit einer Quell- und einer Zieladresse mit einer Breite von 8 Bit versehen. Die obersten Bits stellen die Hostadresse dar, während die unteren Bits eine Portnummer bilden. Eine Netzmaske beschreibt, wie die Aufteilung erfolgen soll. Ein gesetztes Bit in der Netzmaske bedeutet, daß das selbe Bit einer Adresse zur Hostnummer gehört. Die Aufteilung sollte bei allen Teilnehmern identisch sein.

Normalerweise benutzt brickOS die Netzmaske 0xf0 und weist dem RCX die Hostadresse 0 und dem Tower die Adresse 8 zu. Jeder Host kann über 16 Ports verfügen, von denen lediglich Port 0 für den Programmdownload reserviert ist. Der Tower kann in diesem Schema also die Adressen 0x80 bis 0x8f benutzen. Viele dieser Parameter sind veränderbar, teilweise aber nur in der Kernelkonfiguration. Diese beiden Routinen können eine Adresse basierend auf der Netzwerkmaske in Host und Port zerlegen.

```
unsigned char lnp_addressing_host(unsigned char mask, unsigned char address);
unsigned char lnp_addressing_port(unsigned char mask, unsigned char address);
```

Die umgekehrte Berechnung wird von der folgenden Funktion durchgeführt.

```
unsigned char lnp_addressing_addr(unsigned char mask,
                  unsigned char host, unsigned char port);
```

Die Routinen von `lnphost` erlauben beliebigen Netzmasken, es bietet sich aber an, bei dem Standardformat zu bleiben und die höchsten n Bits für die Hostadresse zu benutzen.

Verschiedene Hilfsroutinen

Für Testzwecke ist es mit der Funktion `lnp_hexdump` möglich, Daten als Hexdump auszugeben. Jeder Zeile kann ein Präfix vorangestellt werden, NULL unterdrückt dieses Verhalten.

```
void lnp_hexdump(char *prefix, void *data, unsigned length);
```

B.2 Multiprotokoll-Paketmonitor für Mindstorms

Eine Beispielapplikation der Bibliothek `lnphost`, die gleichzeitig ein nützliches Werkzeug zur Fehlersuche darstellt, ist der Paketmonitor `lnpdump`. Dieses Programm überwacht alle Infrarot-Übertragungen, die der Tower empfangen kann, und gibt die Daten im Rohformat oder interpretiert aus. Die Art der Ausgabe kann über eine Reihe von Parametern eingestellt werden.

```
Usage:
  lnpdump [options]

Dump all packets received from the Lego Mindstorms tower

Options:
  -a, --all                 show packets even if they are invalid
  -d, --device device       set tower device name
  -f, --fast                use 4800 instead of 2400 baud
  -h, --help                show this summary
  -m, --mask n              LNP addressing layer mask, default 0xf0
  -p, --noparity            legOS compatibility mode
  -r, --raw                 don't interpret packages (implies --all)

If no tower device name is specified, it is taken from the environment
variable RCXTTY or the default setting /dev/ttyS0.

Copyright 2004 Stephan Höhrmann, published under the GPL.
```

Das Makefile von `lnpdump` läßt sich sehr einfach für eigene Programme anpassen.

B.3 Disassembler für PIC-Programme

Ein Disassembler ist bei der Programmentwicklung ein wichtiges Werkzeug, um die Ausführungszeit des Programmcodes zu bestimmen. Das Programm kann mit dem normalen Assembler übersetzt werden, wobei alle Makros expandiert werden und der Assembler automatischen Code einfügen kann. Die Hex-Datei enthält dann das endgültige Speicherabbild und kann von einem Disassembler wieder in eine lesbare Darstellung gebracht werden. In seiner Ausgabe kommen nur noch die elementaren Assemblerbefehle des PICs vor (vergleiche Tabelle B.1 für den 14-Bit Kern). Die Ausführungszeit jedes einzelnen Befehls ist statisch bekannt, so daß sich die Ausführungszeit für ganze Blöcke durch einfaches Auszählen ermitteln läßt.

Der hier vorgestellte Disassembler erzeugt im Gegensatz zu den meisten ähnlichen Programmen Code, der durch die Formatierung und den Einsatz symbolischer Namen gut lesbar ist. Außerdem kann das Ergebnis neu assembliert werden und ergibt dann eine Hex-Datei, die komplett identisch mit der

Ausgangsdatei ist. Eine Benennung von Variablen und Prozedurnamen ist nicht ohne weiteres möglich, da eine Datenflußanalyse nötig wäre, um die jeweils aktive Bank zu ermitteln.

Befehl		Beschreibung	Ticks	Opcode	Flags
Datentransport					
movlw	k	Konstante in W laden	1	`11 00?? kkkk kkkk`	-
movwf	f	W in f schreiben	1	`00 0000 1fff ffff`	-
movf	f,d	Wert aus f laden/testen	1	`00 1000 dfff ffff`	Z
Arithmetik					
clrw		W auf 0 setzen	1	`00 0001 0??? ????`	Z
clrf	f	f auf 0 setzen	1	`00 0001 1fff ffff`	Z
incf	f,d	f inkrementieren	1	`00 1010 dfff ffff`	Z
decf	f,d	f dekrementieren	1	`00 0011 dfff ffff`	Z
addlw	k	Konstante zu W addieren	1	`11 111? kkkk kkkk`	C,DC,Z
addwf	f,d	W und f addieren	1	`00 0111 dfff ffff`	C,DC,Z
sublw	k	W von Wert subtrahieren	1	`11 110? kkkk kkkk`	C,DC,Z
subwf	f,d	W von f subtrahieren	1	`00 0010 dfff ffff`	C,DC,Z
comf	f,d	Komplement von f bilden	1	`00 1001 dfff ffff`	Z
swapf	f,d	Halbbytes von f tauschen	1	`00 1110 dfff ffff`	-
Logische Operationen					
andlw	k	W und Konstante berechnen	1	`11 1001 kkkk kkkk`	Z
iorlw	k	W oder Konstante berechnen	1	`11 1000 kkkk kkkk`	Z
xorlw	k	W xor Konstante berechnen	1	`11 1010 kkkk kkkk`	Z
andwf	f,d	W und f berechnen	1	`00 0101 dfff ffff`	Z
iorwf	f,d	W oder f berechnen	1	`00 0100 dfff ffff`	Z
xorwf	f,d	W xor f berechnen	1	`00 0110 dfff ffff`	Z
Bitoperationen					
bcf	f,b	Bit in f löschen	1	`01 00bb bfff ffff`	-
bsf	f,b	Bit in f setzen	1	`01 01bb bfff ffff`	-
rlf	f,d	f nach links rotieren	1	`00 1101 dfff ffff`	C
rrf	f,d	f nach rechts rotieren	1	`00 1100 dfff ffff`	C
Kontrollfluß					
btfsc	f,b	Sprung, wenn Bit in f 0	1(2)	`01 10bb bfff ffff`	-
btfss	f,b	Sprung, wenn Bit in f 1	1(2)	`01 11bb bfff ffff`	-
decfsz	f,d	f=f-1, Sprung wenn Null	1(2)	`00 1011 dfff ffff`	-
incfsz	f,d	f=f+1, Sprung wenn Null	1(2)	`00 1111 dfff ffff`	-
goto	addr	Unbedingter Sprung	2	`10 1aaa aaaa aaaa`	-
call	addr	Prozedur aufrufen	2	`10 0aaa aaaa aaaa`	-
return		Rückkehr von Prozedur	2	`00 0000 0000 1000`	-
retlw	k	Rücksprung mit Wert in W	2	`11 01?? kkkk kkkk`	-
retfie		Rückkehr von Interrupt	2	`00 0000 0000 1001`	-
Verschiedenes					
nop		Keine Operation	1	`00 0000 0??0 0000`	-
sleep		In Standby wechseln	1	`00 0000 0110 0011`	-
clrwdt		Reset Watchdog-Timer	1	`00 0000 0110 0100`	-

Tabelle B.1: Befehlssatz der 14-Bit PIC Mikrocontroller

Diese elementaren Befehle werden in den meisten Assemblern noch um Makros ergänzt, die entweder einfach zu merkende Namen für komplexe Befehle einführen oder Standardsequenzen von Befehlen

unter einem Namen zusammenzufassen. Tabelle B.2 führt die häufigsten Makros auf.

Befehl		Beschreibung	Entspricht
Bedingte Sprungbefehle			
b	addr	Unbedingter Sprung	goto
bc	addr	Sprung wenn Carry gesetzt ist	btfsc/goto
bdc	addr	Sprung wenn Digital Carry gesetzt ist	btfsc/goto
bz	addr	Sprung wenn Zero-Flag gesetzt ist	btfsc/goto
bnc	addr	Sprung wenn Carry nicht gesetzt ist	btfss/goto
bndc	addr	Sprung wenn Digital Carry nicht gesetzt ist	btfss/goto
bnz	addr	Sprung wenn Zero-Flag nicht gesetzt ist	btfss/goto
Bedingtes Überspringen des Folgebefehls			
skpc		Überspringen, wenn Carry gesetzt ist	btfss
skpdc		Überspringen, wenn Digital Carry gesetzt ist	btfss
skpz		Überspringen, wenn Zero gesetzt ist	btfss
skpnc		Überspringen, wenn Carry nicht gesetzt ist	btfsc
skpndc		Überspringen, wenn Digital Carry nicht gesetzt ist	btfsc
skpnz		Überspringen, wenn Zero nicht gesetzt ist	btfsc
Setzen von Flags			
clrc		Carry-Flag auf 0 setzen	bcf
clrdc		Digital Carry-Flag auf 0 setzen	bcf
clrz		Zero-Flag auf 0 setzen	bcf
setc		Carry-Flag auf 1 setzen	bsf
setdc		Digital Carry-Flag auf 1 setzen	bsf
setz		Zero-Flag auf 1 setzen	bsf
Datentransport und Arithmetik			
movfw	f	Den Inhalt von f nach w kopieren	movf
tstf	f	Prüfen, ob der Inhalt von f 0 ist	movf
negf	f	Den Inhalt von f bitweise negieren	comf/incf

Tabelle B.2: Makros erweitern den Befehlssatz der PICs

Der Disassembler `picdasm` benötigt als Parameter nur den Namen einer Eingabedatei sowie eventuell Informationen über deren Format. Zusätzlich ist es aber sinnvoll, den konkreten Zielprozessor anzugeben, da nur so der Einsatz von symbolischen Namen beispielsweise für die Fuses möglich wird. Standardmäßig geht `picdasm` von einem generischen PIC mit 14-Bit Kern aus.

```
Usage:
  picdasm [options] file.hex

Disassembles binary code for a PIC microcontroller.

Options:
  -f, --format format          hex file format
  -h, --help                   show this summary
  -p, --processor pic          target processor type (or 'list')

Supported hex file formats are inhx8, inhx8m, inhx8s, inhx16 and inhx32.

Copyright 2003 Stephan Höhrmann, published under the GPL.
```

B.4 Konvertierung von Dateien im Hex-Format

Es existieren sehr viele Varianten des Intel-Hex-Formates, und nicht jeder Assembler, Programmer oder Debugger unterstützt jedes Format. Daher kann ein Programm nötig sein, daß zwischen den verschiedenen Formatvarianten konvertieren kann. Das Programm `inhx` kann diese Aufgabe übernehmen und bietet darüber hinaus noch einige Optionen nur Bearbeitung von Hex-Dateien.

```
Usage:
    inhx [options] command [parameter(s)]

Utility to view and convert files in Intel hex file format

Commands:
  dump file.hex ...                dump the contents of hex file(s)
  convert src.hex dest.hex         convert hex file
  join in1.hex in2.hex out.hex     join hex file contents
  cut <from> <to> in.hex out.hex   extract memory block from file

Options:
  -b, --big-endian          set byte ordering for conversions
  -d, --dos                 create MS-DOS compatible line endings
  -h, --help                show this summary
  -i, --input format        input file format [default inhx8m]
  -o, --output=format       output file format [default inhx8m]

Supported hex file formats are inhx8, inhx8m, inhx8s, inhx16 and inhx32.

Copyright 2003 Stephan Höhrmann, published under the GPL.
```

B.5 Programmiersoftware für PIC Mikrocontroller

Es gibt viele Steuerprogramme für Programmiergeräte, die unterschiedliche Geräte unterstützen, unter anderem auch den nach dem Tait-Standard funktionierenden shopic aus Anhang A.6. Dennoch gibt es einen guten Grund, eine Steuersoftware zu entwickeln. Der shopic entfaltet zusammen mit einem Programm, das seine Fähigkeiten voll ausnutzt, optimale ICSP-Eigenschaften (vergleiche 3.5). Damit ist es möglich, einen PIC in der Schaltung und sogar im laufenden Betrieb umzuprogrammieren, was eine Menge an Zeit während der Entwicklungsphase und beim Testen eines Programms sparen hilft.

Das Programm `fpp` zeichnet sich durch eine Reihe an Funktionen aus:

- Die Standardfunktionen (Lesen, Schreiben, Verifizieren und Löschen) funktionieren einfach und sind deutlich schneller beendet als bei den meisten vergleichbaren Programmen.
- Der zu programmierende PIC kann automatisch erkannt werden. Wichtige Parameter wie das Timing beim Programmieren werden automatisch angepaßt.
- Es ist möglich, eine Reihe von Informationen über den angeschlossenen PIC oder eine Hex-Datei anzugeben. Dazu gehören die Configuration Fuses, die Speicherbelegung, die Seriennummer und technische Daten des PICs.
- Die werksseitig voreingestellten Kalibrierwerte OSCCAL und BGCAL können ausgelesen und verändert werden. Beim Programmieren bleiben die aktuellen Werte erhalten.

- Unter bestimmten Bedingungen ist es mit `fpp` möglich, einen einmal beschreibbaren PIC mit einem zweiten Programm zu überschreiben. Das Programm hilft bei der Diagnose, ob ein Überschreiben möglich ist, und führt den Vorgang gegebenenfalls durch.
- Die Configuration Fuses können beim Programmieren geändert werden, ohne die Hex-Datei verändern zu müssen.
- Ein PIC kann nachträglich mit Leseschutz für Code und Daten versehen werden. Der Schutz kann auch entfernt werden, allerdings gehen dabei alle Daten verloren.
- Die eingebaute Diagnosefunktion kann bei der Fehlersuche in der Hardware des Programmiergerätes helfen. Alle Grundfunktionen können einzeln getestet werden, und `fpp` ist in der Lage, Testsignalmuster auf den Daten- und Taktleitungen zu erzeugen.
- Die Liste der unterstützten Programmiergeräte und PICs kann von `fpp` ausgegeben werden. Momentan werden praktisch alle Programmiergeräte für den Parallelport unterstützt.

Alle Funktionen sind über ein einfaches Kommandozeileninterface erreichbar. Dadurch kann `fpp` auch besonders einfach in Makefiles oder vergleichbaren Automatismen eingesetzt werden.

```
Usage:
  fpp [options] command [parameter(s)]

User interface to control a parallel port PIC programmer

Commands:
  read filename.hex             read contents of pic, save to file
  write filename.hex            write contents of file into the pic
  verify filename.hex           verify the pic contents against a file
  dump [filename.hex]           dump contents of pic of hex file
  info [filename.hex]           show information about pic or hex file
  overwrite filename.hex        overwrite existing pic contents
  erase                         completely erase the pic
  protect [cp|cpd|cp,cpd]       enable protection for code and/or data
  osccal [value]                read or change osccal setting
  bgcal [value]                 read or change bgcal setting
  debug                         debug the programmer hardware

Options:
  -c, --complete                do not ignore reserved locations
  -d, --device dev              set parport device with programmer
  -e, --noeeprom                do not read/write/erase EEPROM
  -f, --format hexfmt           hex file format
  -h, --help                    show this summary
  -i, --icsp tup,tdn            set ICSP startup/shutdown delays
  -l, --lvp                     use low voltage programming
  -p, --pic (name|list)         set pic type manually
  -s, --set (fuse=val|list)     override configuration fuse setting
  -t, --type (name|list)        set programmer type
  -w, --wait                    pause to insert/remove pic

Copyright 2003 Stephan Höhrmann, published under the GPL.
```

B.6 Serielles Debug-Interface für PICs

Während der Entwicklungsphase müssen PIC-Programme laufend getestet und ihre Eigenschaften verifiziert werden. Dazu ist es erforderlich, Informationen über den internen Zustand, Ergebnisse von Berechnungen und den Inhalt von Variablen nach außen senden zu können. Der Empfänger ist im Idealfall ein PC, der die Daten anzeigen, aufbereiten und automatisierten Tests unterziehen kann. Ein Simulator kann diese Aufgabe nur begrenzt übernehmen, da Fehler in der Kommunikation mit der umgebenden Elektronik auf diese Weise nicht erkannt werden. Außerdem muß für eine Simulation immer die Peripherie und eventuell sogar die Umwelt nachbilden, so daß die schnell außerordentlich komplex werden kann. Es ist daher einfacher, das Programm in der realen Schaltung laufen zu lassen und es mit der Möglichkeit zur Textausgabe auszustatten. Dieser Abschnitt beschreibt eine entsprechende Lösung, die auf der Idee einer seriellen Datenverbindung aufbaut.

- Ein beliebiger Pin des PICs wird als digitaler Ausgang konfiguriert und dient zur Laufzeit ausschließlich der seriellen Kommunikation mit dem PC. Ein UART ist nicht erforderlich.
- In das zu testende Programm wird eine kleine Bibliothek eingebunden, die alle nötigen Kommunikationsroutinen enthält. Die hier vorgestellte Version belegt etwa 30 Instruktionen und 2 Bytes RAM, ist für eine Taktfrequenz von 4 MHz ausgelegt und arbeitet mit 115200 Baud.
- Das Programm kann Makros aufrufen, um Daten an den PC senden. Dabei kann es sich um Zahlen mit 8 oder mehr Bits sowie um Textmeldungen handeln, wobei die Zahlen auf verschiedene Arten formatiert werden können. Die Makros expandieren unabhängig von den Daten zu Codeblöcken von je 4 Instruktionen.
- Der serielle Ausgabepin wird über eine Adapterplatine direkt an einen PC angeschlossen. Auf diesem läuft ein Monitorprogramm, das alle empfangenen Daten aufbereitet und anzeigt.

Die Adapterplatine ist im Anhang A.8 ausführlich beschrieben, während der folgende Text sich mit der Software für den PIC und den PC beschäftigt und das verwendete Protokoll dokumentiert.

B.6.1 API für den Mikrocontroller

Die Bibliothek `sdebug` kann in jedes PIC-Programm eingebunden werden. Voraussetzung ist nur, daß die Taktfrequenz möglichst genau 4 MHz beträgt und noch genügend Speicher frei ist. Zum Einbinden wird als erstes die folgende Zeile in das Programm eingebaut.

```
include "sdebug.inc"
```

Dieser Befehl deklariert zunächst nur Konstanten und Makros, es wird weder Code erzeugt noch Speicher belegt. Dies passiert erst bei der Initialisierung durch den Benutzer. Das API von `sdebug` besteht aus den folgenden Makros:

```
usesdbg  var1,var2,port,pin
sdinit   port,pin
sdinfo   "Text"
sdsendw  FORMAT
sdsendf  FILE,FORMAT
sdstop
```

Um die Bibliothek auch nutzen zu können, muß der Benutzer zunächst zwei Variablen deklarieren und einen Pin als digitalen Ausgang konfigurieren. Diese Ressourcen dürfen vom Hauptprogramm selber nicht benutzt werden.

Anschließend wird an einer beliebigen Stelle des Codes das Makro `usesdbg` eingefügt. Der Assembler expandiert diese Stelle zu dem Rumpf einer Prozedur, die einzelne Bytes im RS232-Protokoll verschickt und von den anderen Makros gebraucht wird. Dem Makro müssen die Namen der beiden Variablen sowie das Register und die Bitnummer für die Steuerung des I/O-Pins übergeben werden.

```
usesdbg  var1,var2,port,pin
```

Das Makro `sdinit` setzt den Pin dann auf den Ruhesignalpegel. Die serielle Schnittstelle im PC benötigt noch einige Millisekunden Wartezeit, dann ist die Verbindung bereit zur Datenübertragung.

```
sdinit   port,pin
```

An dieser Stelle ist die Initialisierung abgeschlossen und die Makros zur Datenübertragung können beliebig aufgerufen werden. Mit `sdinfo` können statische Textmeldungen übertragen werden. Der Text zwischen den Anführungszeichen wird vom Monitorprogramm ausgegeben.

```
sdinfo   "Text"
```

Um Speicherplatz zu sparen, wird nicht der gesamte Text übertragen sondern nur die Programmadresse, an der das Makro auftaucht. Das Monitorprogramm kann anhand der vom Assembler erzeugten Listing-Datei den Text ermitteln und ausgeben.

Die beiden Makros `sdsendw` und `sdsendf` dienen dazu, den Inhalt des W-Registers oder einer Speicherzelle auszugeben.

```
sdsendw  FORMAT
sdsendf  FILE,FORMAT
```

Beide erfordern die Angabe einer Bitmaske mit Formatanweisungen für das Monitorprogramm. Die grundlegenden Formatierungen faßt die folgende Liste zusammen.

- `SDBIN` - Binärformat
- `SDHEX` - Hexadezimalformat
- `SDDEC` - Dezimalformat (vorzeichenlose Zahl)
- `SDINT` - Dezimalformat (mit Vorzeichen)
- `SDCHR` - Zeichen im der Codepage des Monitorprogramms

Die einzelnen Werte können auch kombiniert werden, um mehrere Formate gleichzeitig ausgeben zu können, die Konstante `SDALL` enthält alle Formate. Eine Sonderrolle nimmt das Flag `SDSEQ` ein. Mit ihm gekennzeichnete Bytes werden vom Monitorprogramm zu einer größeren Zahl zusammengesetzt (big endian), bis ein Byte ohne `SDSEQ` empfangen wird. Die Flags des letzten Bytes bestimmen dann das Ausgabeformat für den Gesamtwert.

Das Makro `sdstop` signalisiert dem Monitorprogramm, daß es sich beenden soll. Dieses Signal kann beim automatisierten Testen auch dazu benutzt werden, ein neues Programm in den PIC zu brennen und den nächsten Test zu starten.

```
sdstop
```

Alle Makros können das W-Register und die Flags verändern. Bei ihrem Aufruf muß sichergestellt sein, daß der von `usesdbg` erzeugte Code und die beiden Variablen ohne Umschalten der Bänke oder vergleichbare Voraussetzungen möglich ist. Interrupts dürfen ebenfalls nicht aktiv sein.

B.6.2 Beschreibung des verwendeten Protokolls

Das von `sdebug` verwendete Protokoll ist sehr schlicht aufgebaut. Auf der seriellen Ebene werden 8 Datenbits, 1 Stoppbit und keine Parität bei 115200 Baud verwendet. Jedes Makro zur Datenübertragung sendet ein Paket mit einer Größe von 2 Bytes.

Für die Übertragung einer Textnachricht mit `sdinfo` wird der aktuelle Wert des Programmzählers mit 0x8000 oder-verknüpft und in zwei Bytes zerlegt. Die unteren 8 Bits werden zuerst gesendet, dann folgen die oberen 8 Bits. Das Monitorprogramm erkennt an dem gesetzten MSB des zweiten Bytes, um was für eine Nachricht es sich handelt. Der eigentliche Text kann aus der zum Programm gehörenden Listings-Datei extrahiert und ausgegeben werden.

Bei der Ausgabe eines Zahlenwertes mit `sdsendw` und `sdsendf` wird zuerst das Datenbyte und anschließend ein Byte mit den Formatierungen übertragen.

- Bit 0: `SDBIN` - Binärformat
- Bit 1: `SDHEX` - Hexadezimalformat
- Bit 2: `SDDEC` - Dezimalformat (vorzeichenlose Zahl)
- Bit 3: `SDINT` - Dezimalformat (mit Vorzeichen)
- Bit 4: `SDCHR` - Zeichen im der Codepage des Monitorprogramms
- Bit 5: Nicht benutzt, sollte immer 0 sein
- Bit 6: `SDSEQ` - Verkettung mehrerer Bytes zu einem Wert
- Bit 7: Ist immer 0

Das Makro `sdstop` sendet zwei Bytes mit dem Wert 0xff. Das Monitorprogramm kann diesen Wert sicher von einer Adresse unterscheiden, da es nicht möglich ist, an der Adresse 0x7fff ein `sdinfo`-Makro einzufügen.

B.6.3 Monitorprogramm für den PC

Das zu `sdebug` gehörende Monitorprogramm für den PC heißt `sdmonitor` und sollte unter jedem Posix-konformen Betriebssystem funktionieren. Es gibt für jedes empfangene Paket eine Zeile aus, die mit einer Zeitangabe beginnt und anschließend die Daten in der gewünschten Darstellungsform enthält. Die Zeitmessung startet mit dem ersten empfangenen Paket.

```
Usage:
  sdmonitor [options]

Monitor to display debugging information from PIC microcontroller boards
as generated by the sdebug macro package.

Options:
  -b, --baud=N                baud rate, default 115200
  -h, --help                  show this summary
  -t, --tty=device            serial port [default /dev/ttyS0]
  -l, --listing=file          listing file generated by assembler

Copyright 2003 Stephan Höhrmann, published under the GPL.
```

B.7 Numerische Fehlersimulation der Ortung

In Kapitel 4 wurde eine Reihe von Verfahren zur Koordinatenberechnung vorgestellt. Sie basieren alle darauf, daß aus der Geometrie der Leuchtfeuer und den gemessenen Zeiten die Koordinaten des Roboters ermittelt werden. Dabei führt jeder Fehler in den Meßwerten zu einem Fehler in der Position. Wie groß der resultierende Fehler jeweils ist, kann mit dem Programm `coordsim` simuliert werden.

Es geht von einem quadratischen Feld mit standardmäßig 2 m Seitenlänge aus, in dem eine Reihe von Meßpunkten entlang eines regelmäßigen Gitters gelegt werden. In jedem Meßpunkt werden die gemessenen Zeiten berechnet und mit Fehlern versehen. Aus den Ergebnissen werden Koordinaten berechnet und in eine Grafik eingezeichnet. Im Idealfall entsteht dabei ein Gitter aus sehr kleinen Punkten. Wächst der Fehler an, breiten sich die kleinen Punkte zu großflächigen Punktwolken aus.

Das Programm kann sowohl absolute Meßfehler als auch Fehler simulieren, die mit der Entfernung wachsen. Zu den vorgegebenen Grenzen (absoluter Fehler a in Sekunden, relativer Fehler r) wird eine Anzahl von äquidistanten Werten aus dem Intervall

$$t_{err} \quad \in \quad [t - a - rt, \ldots, t + a + rt] \tag{B.1}$$

für jede der relevanten Meßzeiten t eingesetzt. Als Ergebnis liefert das Programm eine Xfig-Datei, die mit dem gleichnamigen Programm angezeigt und in andere Formate konvertiert werden kann.

```
Usage:
  coordsim [options]

Perform a numerical error simulation of the algorithms that calculate
positions for the ultrasonic positioning system.

Options:
  -N, --scenarioN           use scenario N (1..7)
  -a, --abs value           simulate absolute errors of +/- value [s]
  -c, --clip                clip resulting image
  -d, --decorate            include decorations (area, scale)
  -e, --errsteps value      number of steps in error interval
  -g, --grid value          points in position grid per direction
  -h, --help                show this summary
  -o, --output filename     name of the output xfig file, default stdout
  -r, --rel value           simulate relative errors of +/- value
  -s, --size value          set the size of the square area [m]
  -v, --verbose             show all simulation parameters

This program creates a single xfig file containing the results.

Copyright 2004 Stephan Höhrmann, published under the GPL.
```

B.8 Serielle Datenübertragung vom Sensor zum RCX

Der Sensor des Ortungssystems überträgt seine Meßdaten als seriellen Datenstrom in den RCX. Das analoge Sensorinterface wird benutzt, um einzelne Bits entsprechend codiert zu senden. Eine Erweiterung im Sensorhandler von brickOS setzt daraus wieder ganze Bytes zusammen und übergibt sie an

eine Callback-Funktion der Anwendung. Abschnitt 6.8 beschreibt das Interface und die Kommunikation, hier soll es um die Implementierung des Sensorhandlers und das Benutzerinterface gehen.

Die Grundlage für den Sensorcode bildet das aktuelle brickOS 0.9.0, ergänzt um einige Patches.

- **portability.patch** beseitigt einige Fehler auf anderen Plattformen als Linux und Cygwin.
- **kernelsize.patch** sorgt dafür, daß Anwendungen auch dann compiliert werden können, wenn der Kernel größer als 12 KB wird. Umgekehrt wird es dann auch möglich, den Kernel zu verkleinern, um Platz für eine große Anwendung zu schaffen. Beides ist mit den Standardeinstellungen nicht möglich, weil brickOS von einer festen Kernelgröße von 12 KB ausgeht.
- **serialsensor.patch** enthält den Code für die Ansteuerung der Sensoren.
- **positioning.patch** ergänzt den Sensorcode um das API zum Zugriff auf den Positionssensor.

Alle Patches müssen auf den Quellcode im brickOS-Verzeichnis angewendet werden. Der Hauptschalter für den neuen Sensorcode ist in der Datei `config.h` zu finden und lautet `CONF_DSENSOR_SERIAL`. Wird dieser Schlüssel definiert, aktivieren sich die Veränderungen im Code. Als erstes verändert sich das Timing beim Auslesen aktiver Sensoren, genauer gesagt wird die Pause zwischen dem Abschalten der Stromversorgung und dem Auslesen des Sensorwertes für jeden Sensor um 8 µs erhöht. Die Variable `ds_serial` speichert, welche Sensoren als serielle Quelle konfiguriert sind. Bei diesen wird nach dem Erfassen eines neuen Meßwertes der folgende Handler aufgerufen.

```
#define DS_SERIAL_THRESHOLD (ds_unscale(850))

static unsigned char ds_serial_data[4];
static unsigned char ds_serial_bitcount[4] = { 0, 0, 0, 0 };
static unsigned char ds_serial_sequence[4] = { 0, 0, 0, 0 };
volatile ds_serial_handler_t ds_serial_handler[4] = { DS_SERIAL_NOHANDLER,
            DS_SERIAL_NOHANDLER, DS_SERIAL_NOHANDLER, DS_SERIAL_NOHANDLER };

void ds_serial_bithandler()
{
  unsigned channel = ds_channel;
  unsigned raw = (*((&AD_A)+channel));

  if (ds_serial_bitcount[channel] == 0) {
    if (raw>=DS_SERIAL_THRESHOLD) ds_serial_bitcount[channel] = 8;
      else ds_serial_sequence[channel]=0;
  } else {
    ds_serial_data[channel] = (ds_serial_data[channel]<<1)
      | (raw>=DS_SERIAL_THRESHOLD?1:0);
    ds_serial_bitcount[channel]--;
    if (ds_serial_bitcount[channel]==0) {
      if (ds_serial_handler[channel]!=DS_SERIAL_NOHANDLER)
       ds_serial_handler[channel](((unsigned char)(2-channel))&3,
                                  ds_serial_data[channel],
                                  ds_serial_sequence[channel]);
      ds_serial_sequence[channel]=1;
    }
  }
}
```

Solange der Handler noch keine Daten bekommen hat, wartet er auf ein Startbit mit dem Wert 1. Wenn es eintrifft, faßt der Handler die nächsten 8 Bit zu einem Byte zusammen, wobei die Variable `ds_serial_bitcount` die noch ausstehende Anzahl speichert. Die einzelne Bits werden in dem Schiebepuffer `ds_serial_data` gesammelt, bis das vollständige Byte einer Callback-Funktion der Anwendung übergeben wird. Neben den genannten Daten verwaltet der Handler für jeden Sensor noch das Flag `ds_serial_sequence`. Es ist gesetzt, wenn das aktuelle Byte unmittelbar auf das vorhergehende folgt, ohne daß eine Pause dazwischen ist. Dieses Flag kann den höheren Protokollschichten dabei helfen, Pakete aus mehreren Bytes zusammenzusetzen. Das erste Byte eines Paketes hat das Sequence-Bit nicht gesetzt, alle weiteren schon.

Dem Benutzer steht eine einfache Schnittstelle zu diesen Funktionen zur Verfügung.

```
typedef void (*ds_serial_handler_t) (unsigned char channel,
                                     unsigned char data,
                                     unsigned char sequence);

#define DS_SERIAL_NOHANDLER NULL

void ds_serial_on(volatile unsigned *sensor);
void ds_serial_off(volatile unsigned *sensor);
void ds_serial_set_handler(volatile unsigned *sensor,
                           ds_serial_handler_t handler);
```

Die ersten beiden Funktionen dienen dazu, einen vorher aktivierten Sensoreingang als serielle Datenquelle oder als normalen Sensor zu konfigurieren. Mit der dritten Funktion wird eine Callback-Funktion installiert, wobei auch `DS_SERIAL_NOHANDLER` übergeben werden kann, um die Callback-Funktion wieder zu deaktivieren. Dem Handler wird die Nummer des Eingangs übergeben, von dem die Daten stammen (0=Sensor 1, 1=Sensor 2, 2=Sensor 3). Dank dieser Angabe kann die Anwendung den selben Handler für mehrere Sensoren benutzen und darin die Quellen anhand der Nummer unterscheiden. Bei den weiteren Parametern handelt es sich um den Wert des empfangenen Bytes und das Sequence-Flag. Die Benutzung der Routinen ist einfach, wie das folgende Beispiel zeigt.

```
void handler(unsigned char channel, unsigned char data,
             unsigned char insequence)
{
  ..
}

int main(void) {
  ds_active(&SENSOR_1);
  ds_serial_on(&SENSOR_1);
  ds_set_handler(&SENSOR_1,handler);
  ...
  ds_serial_off(&SENSOR_1);
  ds_passive(&SENSOR_1);
  return(0);
}
```

In jedem Fall muß aber berücksichtigt werden, daß die Callback-Funktion aus einem Interrupt heraus aufgerufen wird. Sie muß sich möglichst schnell beenden und darf möglichst keine Systemfunktionen aufrufen. Durch den Code zur Abfrage der seriellen Sensoren sinken die freie Rechenleistung und die Auslesefrequenz der Sensoren ein wenig, der Verlust ist vergleichbar mit demjenigen, der bei der Aktivierung eines Rotationssensors auftritt.

B.9 BrickOS-API des Ortungssystems

Das API zum Ortungssystem stellt alle Funktionen bereit, die für die Ansteuerung von Sender und Empfänger gebraucht werden. Es baut auf dem Code zur Ansteuerung serieller Sensoren auf und kann über die Einstellung `CONF_POSITION` in `config.h` aktiviert werden. Dadurch wird hauptsächlich ein Handler in den Kernel eingefügt, der die Datenpakete von den Sensoren zusammensetzen kann. In der Header-Datei sind noch einige Funktionen deklariert, die im Folgenden vorgestellt werden.

B.9.1 Ansteuerung des Senders

Der Sender kann an jeden der Motorausgänge angeschlossen werden. Er wird über Veränderungen der Pulsweitenmodulation gesteuert, die über drei verschiedenen Makrosgruppen abgewickelt wird.

```
void possend_a_enable(void);
void possend_b_enable(void);
void possend_c_enable(void);
```

Mit diesen Funktionen wird der Sensor an dem entsprechenden Anschluß in den Bereitschaftsmodus versetzt. Die Kondensatoren laden sich auf und der Mikrocontroller kalibriert sich auf die Spannungen vom RCX. Nach etwa 250 ms ist der Sender dann voll einsatzbereit.

```
#define POSSEND_BURSTTIME 40

void possend_a_burst(void);
void possend_b_burst(void);
void possend_c_burst(void);
```

Wenn der Sender die Initialisierung abgeschlossen hat, kann über die obigen drei Funktionen ein Impuls gesendet werden. Dabei wird auch die fest einprogrammierte Identifikation des Senders per Funk übertragen. Die Ausführung des Befehls dauert `POSSEND_BURSTTIME` Millisekunden, und irgendwo in diesem Intervall wird der eigentliche Impuls gesendet. Es ist prinzipiell nicht möglich, den Sendezeitpunkt präzise zu bestimmen oder zu ermitteln.

```
void possend_a_disable(void);
void possend_b_disable(void);
void possend_c_disable(void);
```

Nach der Benutzung kann der Sender wieder abgeschaltet werden, wenn keine Impulse mehr gesendet werden sollen. Es dauert aber mindestens 250 ms, bis die Kondensatoren des Senders leer sind und er sich wirklich abschaltet. Das folgende Programmfragment sendet ungefähr alle 200 ms einen Impuls.

```
#include <position.h>

possend_a_enable();
msleep(250);
while (...) {
  possend_a_burst();
  msleep(200-POSSEND_BURSTTIME);
}
possend_a_disable();
```

B.9.2 Ansteuerung des Empfängers

Der Empfänger des Ortungssystems wird an einen beliebigen Sensoreingang des RCX angeschlossen, genauso ist auch der gleichzeitige Betrieb mehrerer Empfänger möglich. Bevor diese tatsächlich benutzt werden können, müssen sie zunächst konfiguriert werden.

```
void posrecv_enable(volatile unsigned *sensor);
```

Der Funktion wird ein Zeiger auf den betreffenden Sensor übergeben (z.B. `&SENSOR_1`). Sie schaltet den Eingang in den aktiven Modus, konfiguriert ihn als serielle Datenquelle und verbindet ihn mit dem Handler zum Zusammensetzen der Pakete. Danach muß nur noch eine Callback-Funktion installiert werden, die alle Meßdaten entgegennimmt.

```
typedef void (*posrecv_handler_t) (unsigned char *packet);

#define POSRECV_PACKETSIZE 10
#define POSRECV_NOHANDLER NULL
void posrecv_set_handler(volatile unsigned *sensor, posrecv_handler_t handler);
```

Wenn ein neues Paket mit Meßdaten eintrifft, ruft brickOS diese Callback-Funktion auf und übergibt ihr einen Zeiger auf das Paket mit den Meßdaten. Darin sind die folgenden Werte enthalten.

- **abstime** ist die absolute Laufzeit des Impulses vom Sender zum Empfänger, gemessen in Mikrosekunden mit 24 Bit. Dieser Wert ist nur gültig, wenn das Flag RFVALID gesetzt ist.
- **reltime** enthält die Zeit in Mikrosekunden, die zwischen dem Eintreffen des letzten und des aktuellen Ultraschallsignals vergangen ist. Auch dieser Wert ist mit 24 Bit gespeichert und nur gültig, wenn das Flags USVALID gesetzt ist.
- **echotime** gibt an, wie viele Millisekunden das Echo des aktuellen Ultraschallimpulses im Raum noch nachweisbar war, nachdem der Detektor ausgelöst hat. Das Flag ECVALID ist gesetzt, wenn dieser Wert gültig ist.
- **senderid** enthält die Identifikation des Senders, von dem die aktuellen Daten stammen. Die untersten beiden Bits enthalten die Nummer der Station von 0 bis 3, das Bit darüber ist gesetzt, wenn es sich bei der Quelle um ein Leuchtfeuer (nicht einen mobilen Sender) handelt. Die Identifikation wird per Funk übermittelt, sie gilt also nur bei gesetztem Flag RFVALID.
- **flags** ist die Bitmaske mit den verschiedenen Gültigkeitsflags. Dies sind im einzelnen RFVALID (Bit 0), USVALID (Bit 1) und ECVALID (Bit 2).

Die Callback-Funktion wird direkt aus dem Interrupt-Handler für die Sensoren aufgerufen. Sie muß sich daher möglichst schnell wieder beenden und darf auch fast keine Systemfunktionen aufrufen. Idealerweise kopiert die Funktion nur die Meßdaten und setzt ein entsprechendes Flag. So können andere Teile des Programms die Auswertung übernehmen, die dem normalen Scheduler unterliegen. Die Konstante `POSRECV_PACKETSIZE` enthält aus diesem Grund die Größe des Paketes in Bytes. Ein einmal installierter Handler kann wieder deaktiviert werden, indem `POSRECV_NOHANDLER` statt einer existierenden Funktion übergeben wird.

```
void posrecv_disable(volatile unsigned *sensor);
```

Die Konfiguration des Empfängers kann mit der Funktion `posrecv_disable` wieder rückgängig gemacht werden, wenn dieser nicht mehr gebraucht wird. Der Eingang wird wieder passiv geschaltet, außerdem werden die verschiedenen Handler von dem Kanal entfernt.

Eine Reihe von Funktionen kann benutzt werden, um auf die einzelnen Felder des Paketes zuzugreifen.

```
unsigned long posrecv_abstime(unsigned char *packet);
unsigned long posrecv_reltime(unsigned char *packet);
unsigned int  posrecv_echotime(unsigned char *packet);
unsigned char posrecv_senderid(unsigned char *packet);
unsigned char posrecv_rfvalid(unsigned char *packet);
unsigned char posrecv_usvalid(unsigned char *packet);
unsigned char posrecv_ecvalid(unsigned char *packet);
unsigned char posrecv_frombeacon(unsigned char *packet);
```

Die meisten dieser Funktionen geben direkt die gleichnamigen Einträge oder Flags zurück, die einzige Ausnahme bildet die Identifikation des Senders. `posrecv_senderid` liefert die Nummer des Senders ohne das Bit zur Identifikation eines Leuchtfeuers, der mobile Sender mit der Nummer 0 und das Leuchtfeuer 0 sind mit dieser Funktion nicht zu unterscheiden. Dafür dient dann `posrecv_frombeacon`. Genau wie die Funktionen zur Abfrage der Flags gibt diese Funktion einen Wert ungleich 0 zurück, wenn das Leuchtfeuer-Bit gesetzt ist.

B.10 Algorithmus zum Ziehen von Quadratwurzeln

Unter brickOS gibt es keine Funktion zum Ziehen von Wurzeln, insbesondere sind die ganzen Routinen aus `math.h` nicht nutzbar. Für die Koordinatenberechnung wird aber eine solche Funktion gebraucht, die möglichst mit Ganzzahlarithmetik arbeitet und die Lösung in konstanter Zeit liefert. Näherungsverfahren sind bei dem abzudeckenden Wertebereich meist nicht schnell genug. Ein sehr elegantes Verfahren ähnelt der schriftlichen Division und nutzt die Tatsache aus, daß jede positive Zahl gleich dem Quadrat ihrer Wurzel ist. Es basiert auf einer Idee von R. Starrost.

```
unsigned sqrt32(unsigned long x)
{
  unsigned long rest,temp,value;
  int pairs=16;

  if (x == 0) return(0);
  while ((x & 0xC0000000) == 0) { x <<= 2; pairs--; }
  value = 1;
  rest = (x >> 30) - 1;
  while (pairs > 1) {
    x = (x << 2) & 0xffffffff;
    rest = (rest << 2) | (x >> 30);
    temp = (value << 2) | 1;
    value <<= 1;
    if (temp <= rest) {
      rest-=temp;
      value |= 1;
    }
    pairs--;
  }
  if (rest > value) value++;
  return(value);
}
```

Anhang C

Meßwerte zu den Abbildungen

In vielen Kapiteln dieser Arbeit werden die Ergebnisse von Messungen nur als Diagramm dargestellt. Dieses Kapitel liefert die Zahlengrundlage der Diagramme nach und beschreibt die Durchführung der Messung, soweit dies zur Interpretation der Werte erforderlich ist.

C.1 Drehmoment des Lego-Motors 71427 (Abbildung 2.5)

In der Studienarbeit von [Janz] werden viele mechanische Eigenschaften des Lego-Motors 71427 experimentell ermittelt. Die hier aufgeführten Zahlen geben einen Teil der Ergebnisse wieder. Bei der Messung werden verschiedene Gewichte über eine Rolle mit 85,4 mm Durchmesser in die Höhe gezogen, wobei die Rolle direkt mit dem Motor und einem Rotationssensor verbunden ist. Aus dem Gewicht, dem Durchmesser und der Rotationsgeschwindigkeit läßt sich das Drehmoment berechnen. Die Messung wird für verschiedene Einstellungen der PWM unter brickOS wiederholt.

PWM	**Gewicht [g]**	**Drehzahl [min^{-1}]**	**Drehmoment [mNm]**
255	0	346,82	0,0
255	25	300,00	7,2
255	50	257,51	14,3
255	75	215,83	21,5
255	100	173,91	28,6
255	125	129,87	35,8
255	150	84,51	43,0
191	0	340,91	0,0
191	25	277,78	7,2
191	50	222,22	14,3
191	75	164,38	21,5
191	100	106,19	28,6
191	125	46,15	35,8
128	0	329,67	0,0
128	25	237,15	7,2
128	50	151,90	14,3
128	75	60,91	21,5
64	0	303,03	0,0
64	25	134,23	7,2

Tabelle C.1: Drehmoment und Drehzahl des Lego-Motors 71427

C.2 Klemmenspannung am Aktorausgang (Abbildung 2.6)

Bei dieser Messung wird der RCX mit frischen Batterien bestückt und unter der Firmware gestartet. Die Motorausgänge A und C werden im Dauerbetrieb aktiviert, wobei Ausgang A mit einer einstellbaren Stromsenke auf der Basis eines Leistungstransistors belastet wird. Für verschiedene Einstellungen der Stromsenke werden der entnommene Strom und die Klemmenspannung am Ausgang gemessen.

Strom [mA]	**Spannung [V]**
0	7,77
58	7,27
108	6,92
216	6,17
324	5,42
432	4,64
445	4,31
461	2,98
486	0,82
488	0,56

Tabelle C.2: Verhalten eines Aktorausgangs bei Belastung

C.3 Sensorwerte im passiven Modus (Abbildung 2.9)

Diese Messung soll die Zusammenhänge zwischen dem Sensorwert eines passiven Sensors und seinen elektrischen Daten aufzeigen. Dazu werden verschiedene Potentiometer an einen Sensoreingang angeschlossen und die anliegende Spannung sowie der durch das Potentiometer fließende Strom gemessen. Aus beiden Größen läßt sich der Widerstand des Potentiometers berechnen. Parallel zeigt ein unter brickOS laufendes Programm den Rohwert des Sensors auf dem Display an.

Sensorwert	**Spannung [V]**	**Strom [mA]**	**Widerstand [kΩ]**
0	0,00	0,500	0,00
16	0,07	0,493	0,14
32	0,14	0,485	0,29
64	0,30	0,469	0,64
128	0,61	0,438	1,39
192	0,92	0,408	2,25
256	1,24	0,376	3,30
320	1,56	0,344	4,53
384	1,87	0,313	5,97
448	2,18	0,282	7,73
512	2,49	0,251	9,92
640	3,12	0,188	16,60
768	3,75	0,125	30,00
896	4,38	0,062	70,65
1000	4,89	0,012	407,50
1023	5,01	0,000	∞

Tabelle C.3: Spannung, Strom und Sensorwerte im passiven Modus

C.4 Klemmenspannung bei aktiven Sensoren (Abbildung 2.13)

Bei dieser Messung handelt es sich um eine der wichtigsten dieser Arbeit. Ihre Ergebnisse zeigen, wie viel Strom einem aktiven Mindstorms-Sensor maximal zur Verfügung steht. Sie liefert damit einen der wichtigsten Parameter für das Design von Sensoren.

Abbildung C.1: Belastungsmessung an einem Sensoreingang

Einer der Sensoranschlüsse des RCX wird in den aktiven Modus geschaltet und mit einer einstellbaren Stromsenke auf Basis eines Leistungstransistors belastet. Dabei werden ständig der entnommene Strom und die Klemmenspannung am RCX gemessen. Der Aufbau ist in Abbildung C.1 zu sehen. Die Messung wird einmal unter brickOS, einmal nur mit der Firmware wiederholt.

Lego Firmware		brickOS	
Strom [mA]	**Spannung [V]**	**Strom [mA]**	**Spannung [V]**
0,08	7,98	0,00	7,96
0,86	7,94	0,10	7,95
2,11	7,87	0,88	7,90
3,05	7,82	1,45	7,86
4,99	7,72	2,40	7,80
6,07	7,66	3,98	7,70
8,60	7,53	5,49	7,60
10,04	7,45	7,04	7,50
10,80	7,40	8,23	7,42
11,77	7,30	8,98	7,35
11,97	7,25	9,68	7,20
12,06	7,20	10,05	6,50
12,32	6,69	10,48	5,26
13,22	4,20	11,07	3,40
14,08	1,82	11,70	1,50

Tabelle C.4: Klemmenspannung bei aktiven Sensoren

C.5 Ultraschall abhängig von der Treiberspannung (Abbildung 4.4)

Diese und die folgende Messung sollen dokumentieren, wie die Signalstärke einer Ultraschallübertragung von der Spannung am Sender und der Entfernung abhängen. Bei der ersten Messung wurden eine Sende- und eine Empfangskapsel im Abstand von 40 cm aufgestellt und aufeinander ausgerichtet. Die Sendekapsel wird von einer Rechteckspannung mit passender Frequenz und veränderlicher Amplitude angesteuert. Ein Oszilloskop am Empfänger dient dazu, die Signalstärke zu messen.

Sender [Vpp]	**Empfänger [mVpp]**
0,15	3
0,20	4
0,40	8
1,00	19
2,00	38
4,00	82
10,00	200
20,00	380

Tabelle C.5: Abhängigkeit der Signalstärke von der Treiberspannung

Die Meßwerte zeigen sehr deutlich, daß die Signalstärke proportional zur Sendespannung ist.

C.6 Ultraschall abhängig von der Entfernung (Abbildung 4.5)

Der zweite wichtige Faktor für die Signalstärke ist, wie diese sich mit der Entfernung verändert. Eine feststehende Sendekapsel strahlt ein Signal konstanter Stärke ab, während die Empfängerkapsel entfernt wird. Die Kapseln werden vor jeder Messung so aufeinander ausgerichtet, daß das Signal möglichst stark ist, um Auswirkungen der Richtungsabhängigkeit zu verhindern.

Distanz [cm]	**Empfänger [mVpp]**
10	1800
20	900
30	600
50	370
80	250
100	200
150	140
180	110
200	100
400	50

Tabelle C.6: Abhängigkeit der Signalstärke von der Entfernung

Die Meßwerte belegen, daß die Signalstärke proportional zum Kehrwert der Entfernung fällt. Zusammenfassend bestätigen die beiden Messungen den erwarteten Zusammenhang zwischen Senderspannung U_S, Empfängerspannung U_E und Entfernung d.

$$U_E \sim \frac{U_S}{d}$$

C.7 Fehlersimulation für Koordinatenberechnung

Die Abbildungen 4.10 bis 4.16 wurden mit dem Programm `coordsim` erstellt. Dabei wurden die folgenden Parameter verwendet.

- `-a 10e-6`: Absoluter Fehler 10 µs (entspricht 3,3 mm)
- `-r 0.0025`: Relativer Fehler 0,25 %
- `-cd`: Grafik zurechtschneiden, Rahmen und Maßstab einzeichnen

Bei Szenario 5 mußte mit `-g10` das Gitter so gelegt werden, daß die Mittelachsen nicht mit ausgewertet werden, und bei einigen Szenarien wurde mit `-e` die Anzahl der Fehlerpunkte erhöht, um eine höhere Auflösung der Fehlerflächen zu erreichen.

C.8 Fehler der Entfernungsmessung (Abbildung 9.1 und 9.6)

Ein Roboter kann nur mit hoher Genauigkeit positioniert werden, wenn das Ortungssystem Entfernungen mit entsprechender Präzision erlaubt. Eine Kontrollmessung soll belegen, wie genau dies möglich ist und mit welchen Fehlern der Benutzer rechnen muß. Dazu wird ein Roboter mit einem Empfänger ausgestattet und vor einem Leuchtfeuer aufgestellt. Von dort aus entfernt er sich in sehr kleinen Schritten und mißt laufend die Entfernung.

Abbildung C.2: Aufbau zur Analyse der Genauigkeit bei der Entfernungsmessung

Damit die Daten mit einem entsprechend präzisen Referenzwert verglichen werden können, mißt ein Rotationssensor am Antriebsmotor die zurückgelegte Strecke. Alle Meßwerte werden über Infrarot an einen Computer in der Nähe übertragen und dort ausgewertet. In einem ersten Schritt werden ungültige Meßwerte gelöscht, dann faßt ein Filter jeweils N aufeinanderfolgende Werte zu einem zusammen, um das Rauschen und eventuelle Ausreißer zu beseitigen. Dabei werden die Werte sortiert, die M größten und kleinsten Zahlen verworfen und aus den übrigen ein Mittelwert gebildet.

Tatsächliche und gemessene Entfernung in Zentimetern											
Real	**Gem**	**Real**	**Gem**	**Real**	**Gem**	**Real**	**Gem**	**Real**	**Gem**	**Real**	**Gem**
8,5	9,5	163,0	162,8	310,7	307,3	458,1	450,5	604,9	588,4	750,0	733,9
12,2	13,1	166,3	166,0	314,0	310,5	461,5	453,5	608,2	591,5	753,3	737,0
15,7	16,6	169,7	168,8	317,4	313,9	464,9	456,7	611,4	594,5	756,5	740,5
19,2	19,6	173,0	172,3	320,7	317,2	468,2	459,7	614,7	597,6	759,7	743,8
22,6	22,9	176,4	176,3	324,1	320,6	471,4	462,6	618,1	600,7	762,9	747,5
26,1	26,0	179,9	180,6	327,4	324,1	474,8	465,7	621,6	603,9	767,7	752,2
29,5	29,2	183,3	183,8	330,8	327,5	478,1	468,7	624,8	606,7	771,0	755,2
32,9	32,5	186,6	187,4	334,2	330,8	481,3	471,7	628,0	609,9	774,3	758,4
36,3	36,6	190,1	190,4	337,6	334,0	485,0	474,9	631,3	613,4	777,8	761,9
39,9	40,2	193,3	193,6	341,0	337,5	488,2	477,9	634,6	616,8	781,1	765,1
43,3	43,7	196,6	196,8	344,5	340,9	491,5	480,9	638,0	620,2	784,3	768,4
46,8	47,1	199,9	199,8	347,7	344,3	494,9	483,9	641,4	623,6	787,4	771,5
50,2	50,4	203,2	203,0	351,2	347,8	498,1	486,9	644,5	626,5	790,8	775,0
53,6	53,7	206,6	206,4	354,7	351,1	501,3	489,9	647,9	629,8	794,0	778,0
57,2	57,1	210,0	209,8	358,0	354,0	504,6	492,9	651,1	633,1	797,3	781,4
60,7	60,9	213,3	213,1	361,6	357,2	507,9	496,0	654,4	636,4	800,8	785,0
64,3	64,9	216,9	216,5	364,9	360,8	511,3	499,1	657,7	639,8	804,1	788,2
67,7	68,1	220,2	219,8	368,3	364,2	514,8	502,4	661,1	643,0	807,4	791,9
71,0	71,4	223,6	223,1	371,7	367,6	518,2	505,5	664,4	646,3	810,7	795,2
74,5	74,7	227,1	226,5	374,9	370,9	521,5	508,6	667,7	649,6	813,9	798,1
78,0	77,8	230,5	229,7	378,2	374,2	524,8	511,8	671,1	653,1	817,1	801,5
81,3	81,2	233,8	233,1	381,5	377,5	528,1	514,9	674,4	656,5	820,4	805,1
84,7	84,5	237,2	235,7	384,7	380,6	531,6	518,2	677,7	659,8	823,6	808,1
88,3	89,0	240,4	238,9	388,2	384,2	535,0	521,5	681,0	663,0	826,9	811,6
92,4	93,2	243,7	242,1	391,5	387,4	538,4	524,7	684,3	666,7	830,3	814,8
98,1	98,5	247,1	245,5	394,8	390,7	541,7	527,9	687,6	669,8	833,6	818,5
101,5	101,8	250,4	248,7	398,2	394,1	545,0	531,1	690,7	673,2	836,9	821,9
104,9	105,5	253,7	252,0	401,5	397,3	548,3	534,1	694,0	676,9	840,3	825,4
108,2	109,0	257,1	255,3	404,8	400,4	551,7	537,4	697,4	680,1	843,6	828,8
111,7	112,2	260,5	258,7	408,2	403,6	555,0	540,5	700,5	683,3	847,0	832,4
115,1	115,6	264,0	262,0	411,4	406,8	558,3	543,9	703,8	686,7	850,3	835,9
118,4	118,6	267,4	265,4	414,9	410,0	561,8	547,1	707,2	690,1	853,7	839,3
121,9	121,9	270,7	268,6	418,2	413,3	565,1	550,4	710,4	693,6	857,2	842,8
125,4	125,3	274,2	272,0	421,4	416,3	568,3	553,5	713,8	696,9	860,5	846,4
128,9	128,7	277,6	275,4	424,7	419,3	571,7	556,9	717,0	700,2	864,0	850,1
132,3	132,4	280,9	278,6	428,1	422,5	575,0	559,9	720,3	703,9	867,4	853,3
135,7	135,7	284,3	281,9	431,4	425,6	578,2	563,0	723,7	707,2	870,6	856,7
139,1	139,0	287,6	285,1	434,7	428,8	581,5	566,2	727,0	710,5	873,7	860,3
142,5	142,4	290,9	288,3	438,3	432,0	584,9	569,6	730,3	713,8	877,1	863,9
145,9	145,7	294,3	291,5	441,6	435,0	588,2	573,1	733,7	717,2	880,4	867,2
149,2	149,1	297,6	294,6	445,0	438,1	591,6	575,9	737,0	720,6		
152,6	152,4	300,8	297,5	448,3	441,3	594,8	579,1	740,3	723,9		
156,0	155,8	304,1	300,6	451,6	444,3	598,3	582,2	743,6	727,3		
159,5	159,4	307,4	304,0	454,9	447,4	601,6	585,4	746,8	730,5		

Tabelle C.7: Entfernungsmessung über 9 Meter (gefiltert mit N=32 und M=6)

C.9 Fehler der Koordinatenberechnung (Abbildung 9.2 und 9.3)

Die Fehler der Entfernungsmessung gehen auch in die Berechnung der Koordinaten ein und können sich dort weiter verstärken. Sie bilden daher die Grundlage für die Genauigkeit der Ortung, letztendlich sind sie für das Gesamtsystem aber nicht das wichtigste Gütekriterium. Entscheidend ist, die genau die am Ende ausgegebenen Koordinaten sind.

Zwei Beispiele sollen die in der Praxis erreichte Genauigkeit belegen. Dazu wurde ein Feld mit einer Größe von 2m×2m aufgebaut und der Empfänger darin auf einer Kreisbahn bewegt. Die Meßwerte der ersten Tabelle entstanden bei einem Aufbau nach Szenario 2 unter Benutzung des Funksignals. Sie weichen kaum sichbar von der Ideallinie ab.

x [cm]	**y [cm]**	**x [cm]**	**y [cm]**	**x [cm]**	**y [cm]**
95,1	26,7	38,7	120,0	151,8	124,8
88,6	27,1	42,9	127,2	155,4	117,7
74,1	30,0	47,9	133,7	158,4	108,6
67,8	32,3	53,7	140,0	160,2	98,9
61,4	36,3	60,2	145,0	160,0	85,8
55,0	40,9	68,2	149,7	158,2	75,8
48,6	46,9	77,2	152,9	154,8	66,2
43,0	54,1	94,4	156,1	149,6	55,8
39,3	60,2	109,4	154,6	143,4	47,7
35,3	69,4	118,2	152,4	136,9	41,2
33,1	79,6	126,3	149,0	129,5	36,4
32,4	90,6	133,0	144,6	123,4	32,8
33,2	102,3	140,1	139,1	115,5	29,4
35,7	112,7	146,3	132,2	108,0	27,3

Tabelle C.8: Koordinaten einer Kreisbahn, berechnet mit absoluten Laufzeiten

In der zweiten Tabelle wird der selbe Kreis dargestellt, allerdings wurden die Koordinaten alleine auf der Basis der Laufzeitunterschiede berechnet, hier kam das Szenario 4 zum Einsatz, das in Y-Richtung etwas größere Fehler liefert.

x [cm]	**y [cm]**	**x [cm]**	**y [cm]**	**x [cm]**	**y [cm]**
95,2	25,4	42,7	113,3	152,4	115,8
86,4	24,4	47,4	121,9	156,2	106,8
79,0	25,2	50,7	132,4	157,5	93,5
72,7	25,7	57,4	138,7	159,2	86,3
65,3	32,2	66,0	144,9	158,0	76,7
58,4	36,0	72,5	147,6	155,6	67,6
52,0	41,9	80,9	151,0	152,3	61,4
46,8	47,9	93,3	150,7	148,2	54,8
42,5	54,7	105,6	151,9	142,1	47,4
37,4	64,9	117,8	149,2	136,8	42,4
34,8	72,4	126,9	144,6	129,2	36,3
35,2	83,5	134,9	139,7	119,5	31,7
35,7	94,4	140,9	130,1	111,9	29,1
37,5	105,8	147,6	122,3	104,5	27,0

Tabelle C.9: Koordinaten einer Kreisbahn, berechnet mit Laufzeitunterschieden

Anhang D

Platinenherstellung

Wenn der Prototyp einer Schaltung auf dem Steckbrett aufgebaut und ausgiebig getestet wurde, soll er in der Regel dauerhaft und stabil als Platine realisiert werden. Der Aufbau wird dadurch kleiner, reagiert weniger empfindlich auf Störungen und läßt sich mechanisch wesentlich besser handhaben. Eine einfache Möglichkeit ist der Einsatz von Lochraster- oder Lochstreifenplatinen, die es lötfertig zu kaufen gibt, und mit denen einfache Schaltungen schnell aufgebaut sind. Bei komplexeren Schaltungen oder wenig verfügbarem Platz kommen nur noch gedruckte Schaltungen in Frage, bei denen Bauteile und die verbindenden Leiterbahnen nahezu beliebig plaziert werden können.

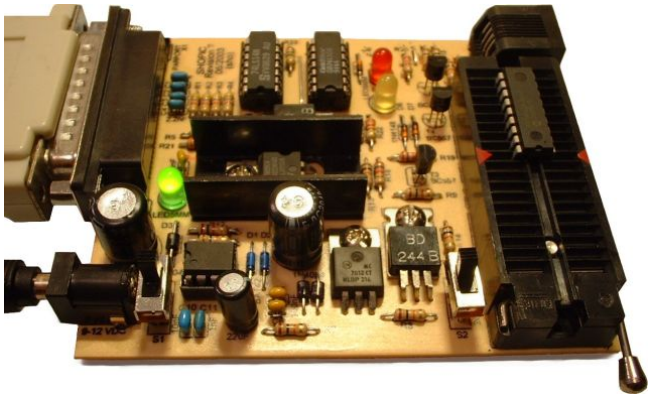


Abbildung D.1: Beispiel für eine gedruckte Schaltung

In der Regel werden gedruckte Schaltungen industriell produziert und sind in kleinen Stückzahlen relativ teuer, sie können aber auch mit ein wenig Arbeit selbst hergestellt werden. Als Ausgangsmaterial dient dabei eine stabile, nichtleitende Trägerplatte, auf die eine Kupferschicht mit einer Dicke von typischerweise 35 µm aufgebracht ist. Eine dünne Schicht Fotolack bedeckt das gesamte Kupfer, die wiederum durch eine lichtundurchlässige Klebefolie während der Lagerung geschützt ist.

Die Leiterbahnen werden auf fotochemischem Wege erzeugt. Als Vorlage dient dabei eine Transparentfolie, auf welche die späteren Leiterbahnen als schwarze Flächen gedruckt sind. Sie wird direkt auf die lackierte Seite der Platine gelegt. Eine Lichtquelle bestrahlt dann die Platine durch die Folie mit UV-haltigem Licht. In den nicht geschwärzten Bereichen kann das Licht die Folie passieren und greift den Fotolack an. So entsteht im Lack ein Abbild der Vorlage, das in einem Entwicklerbad herausgearbeitet wird. Der Entwickler löst den Fotolack an allen belichteten Stellen ab, so daß das Kupfer freiliegt und chemisch entfernt werden kann. Die noch von intaktem Lack bedeckten Stellen bleiben dabei unversehrt und bilden jetzt die Leiterbahnen. Abschließend werden Löcher für die Bauteilanschlüsse gebohrt, und die Platine wird beschriftet, lackiert und bestückt.

Platinen können Kupferschichten sowohl auf der Unter- als auch der Oberseite tragen, in der industriellen Fertigung werden oft auch Leiterbahnebenen im Inneren der Platine schichtweise angelegt. Aber auch die doppelseitigen Platinen sind in der Herstellung bereits deutlich aufwendiger, weil die Belichtungsfolien auf Ober- und Unterseite exakt aufeinander ausgerichtet werden müssen. Außerdem sind die Verbindungen zwischen beiden Seiten (Durchkontaktierungen oder Vias genannt) nicht ganz einfach zu löten. Mit einem gut optimierten Leiterbahnlayout und wenigen Drahtbrücken lassen Schaltungen sich aber häufig genauso gut auf einseitigen Platinen realisieren.

Ein großer Vorteil der gedruckten Schaltungen ist, daß das Layout mit Computerunterstützung erstellt werden kann. Die Werkzeuge nehmen dem Entwickler eine Reihe von Arbeitsschritten ab und können viele Fehler durch automatisiertes Testen verhindern. Trotzdem ist das Erstellen einer Platine ein aufwendiger Prozeß, der aus vielen Schritten besteht, und jeder einzelne muß sorgfältig und fehlerfrei ausgeführt werden, um ein gutes Ergebnis zu erhalten. Im Folgenden wird der gesamte Prozeß ausführlich erklärt, basierend auf eigenen Erfahrungen und Quellen wie [Bredendiek] und [dseFAQ].

D.1 Erstellen der Layoutvorlage

Das Erstellen einer Platine beginnt damit, daß der Schaltplan in ein CAD/CAM-Programm eingegeben wird, mit dem das Platinenlayout, also die Plazierung der Bauteile und das Verlegen der Leiterbahnen, durchgeführt wird. Ein solches Programm ist Eagle von [Cadsoft], mit dem auch alle Platinen in dieser Arbeit erstellt wurden. Für Windows und Linux gibt es eine kostenlose Version von Eagle, die auf eine maximale Platinengröße von 100 x 80 mm beschränkt ist, was aber für einfache Projekte vollkommen ausreicht. Abbildung D.2 zeigt eine Platine, die mit dieser Version entworfen wurde.

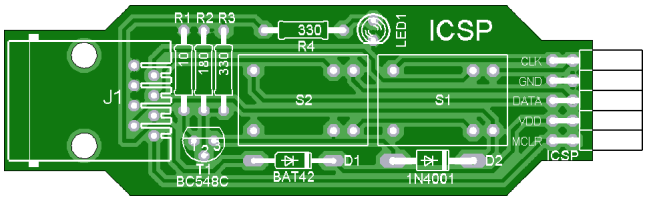


Abbildung D.2: Beispiel für eine mit Eagle entworfene Platine

Der Vorteil beim Einsatz eines solchen Werkzeugs ist, daß der Designprozeß durch den Computer unterstützt und kontrolliert wird. Die Gehäuseformen gängiger Bauteile liegen bereits als Bibliotheken vor, so daß der Benutzer sich nicht um Details wie die Position und die Belegung der Anschlüsse, den Durchmesser der Anschlußdrähte und ähnliches kümmern muß. Eine Reihe von Tests überprüft die Schaltung auf einfache Fehler und untersucht das Leiterbahnlayout auf die Einhaltung definierbarer Designregeln. Viele Fehler, die beim manuellen Layouten leicht passieren können, werden dadurch effektiv verhindert. CADSoft bietet eine Reihe von Tutorials an, die eine gute Einführung in die Bedienung des Programms darstellen. Ein gutes Layout damit zu erstellen, erfordert aber einige praktische Erfahrung. Wie sich viele der häufigsten Fehler vermeiden lassen, beschreiben die folgenden Punkte.

- Beim Entwurf der Schaltung sollte bereits das spätere Layout bedacht werden. Querverbindungen in der Schaltung führen oft zu sich überkreuzenden Leiterbahnen, vor allem bei räumlich getrennt liegenden Funktionsgruppen. Sie sollten vermieden werden, wo immer dies möglich ist.
- Wenn die Platine in ein Gehäuse eingebaut werden soll, müssen dessen Eigenheiten möglichst früh in das Layout mit einbezogen werden. Dazu gehören die Platinengröße, Aussparungen, Bohrungen für Befestigungsschrauben und die Position der Anschlüsse.

- Die Bauteile dürfen anfangs nicht zu dicht gepackt werden. Kleine Änderungen wie die Anpassung einer Gehäuseform oder Modifikationen an der Schaltung wären sonst mit sehr vielen Korrekturarbeiten verbunden. Das Layout sollte daher zunächst großzügig angelegt und später bei Bedarf verkleinert werden.
- Einige Baugruppen wie Anschlüsse, Bedienelemente, Jumper oder Trimmer brauchen ausreichend freien Platz um sich herum, damit sie bedienbar bleiben. Dies muß beim Layout bedacht und möglichst mit einem Modell der Platine überprüft werden.
- Die Anordnung der Bauteile ist maßgeblich dafür, wie viele Überschneidungen von Leiterbahnen es im Layout geben wird. Es lohnt sich also, Zeit in eine gute Planung zu investieren.
- Es ist eine gute Strategie, zuerst alle Bauteile zu plazieren, die an festen Positionen liegen sollen (Bedienelemente, Anschlüsse und ähnliches). Dann werden die restlichen Teile in Funktionsgruppen aufgeteilt, die untereinander möglichst wenig verbunden sind, und getrennt layoutet. Zum Schluß werden die Funktionsgruppen im Layout zusammengefügt.
- Bauteile wie Widerstände und Dioden können sowohl stehend als auch liegend eingelötet werden. Bei liegender Montage brauchen sie mehr Platz, sind aber stabiler gegenüber mechanischer Belastung. Hier muß je nach den Anforderungen abgewogen werden.
- Bei der Eingabe des Schaltplanes wird für jedes Bauteil die Gehäuseform abgefragt. Hier sollte möglichst gleich die Form ausgewählt werden, die später auch tatsächlich eingelötet wird. Anderenfalls kann sich hier ein Fehler einschleichen, der erst sehr spät bemerkt wird. Davon sind insbesondere Elektrolytkondensatoren betroffen, deren Formen erheblich variieren können.
- Der Autorouter kann die Leiterbahnen weitgehend automatisch verlegen, liefert aber schlechtere Ergebnisse als manuelles Layout (jedenfalls bei semiprofessionellen Programmen wie Eagle). Das Ergebnis hat unnötig viele Vias und ist oft nur schlecht lötbar.
- Je kleiner und enger die Strukturen auf der Leiterbahnseite der Platine sind, desto schwieriger wird das Löten. Insbesondere steigt die Gefahr, benachbarte Leiterbahnen zu verbinden.
- Das Bohren von Löchern wird wesentlich einfacher, wenn die späteren Löcher von einem Kupferring umrandet sind. Der Bohrer wandert dadurch beim Ansetzen in die Mitte des Loches und zentriert sich so von selbst. Eagle erstellt bei vielen Bauteilanschlüssen diese Form automatisch.
- Vias müssen einen ausreichenden Durchmesser haben, um von Hand gelötet werden zu können. Die Standardeinstellungen sind für chemisches Durchkontaktieren ausgelegt und damit zu klein, sie müssen also deutlich vergrößert werden. Eine bessere Alternative ist, die Anschlüsse von Widerständen und Dioden als Vias zu benutzen.
- Drahtbrücken bei einseitigen Platinen können als Leiterbahnen auf der sonst nicht vorhandenen Oberseite realisiert werden. Die von Eagle beim Wechsel zwischen Ober- und Unterseite erzeugten Vias bilden dann die Lötaugen für die Drahtbrücke.
- Die Versorgungsleitungen sollten so breit wie möglich sein, damit sie nur einen geringen Widerstand haben und auch größere Ströme transportieren können.
- In unbenutzten Bereichen der Platine sollte die Kupferschicht stehen bleiben und möglichst mit Masse verbunden werden. Dies schont die Säure, weil weniger Kupfer weggeätzt werden muß, und verbessert zudem die Störempfindlichkeit der Schaltung.
- Leiterbahnen, die hochfrequente Signale transportieren, können bei Bedarf durch benachbarte Masseleitungen abgeschirmt werden, um Störungen zu minimieren.

Viele potentielle Probleme der Platinenform und der Anordnung der Bauteile lassen sich auf dem Monitor nur schlecht erkennen. Ein realistisches Modell der Platine hilft da deutlich weiter. Ein einfacher aber sehr effektiver Trick ist, das Layout auf Papier oder dünnen Karton zu drucken, die Bohrlöcher

mit einer Nadel zu durchstechen und alle Bauteile in die Löcher zu drücken. Auf diese Weise entsteht ein realistisches Modell, in dem Designfehler sichtbar werden, bevor eine echte Platine hergestellt ist.

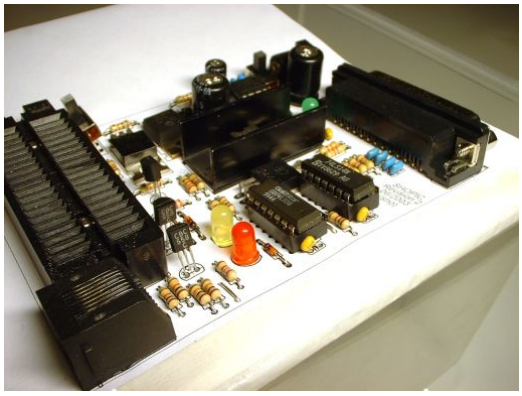


Abbildung D.3: Papiermodell einer Platine, befestigt auf einer Plastikdose

Wenn das Platinenlayout in Ordnung ist, wird der Kupferlayer maßstabsgetreu auf eine Transparentfolie gedruckt. Die bedruckte Seite sollte später direkt auf der Platine aufliegen, damit Lichtreflexionen in der Folie nicht zur Belichtung eigentlich abgedeckter Stellen führen. Beim Ausdrucken ist besonders wichtig, die dunklen Bereiche der Folie möglichst undurchlässig für UV-Licht zu bekommen. Gleichzeitig muß der Ausdruck konturscharf sein und darf nicht verwischen. Je nach Art des verwendeten Druckers sind dafür verschiedene Einstellungen nötig.

- Laserdrucker liefern ein gutes Druckbild, erreichen häufig aber nur eine mittelmäßige Deckung, die über Treibereinstellungen nur wenig erhöht werden kann. Sollte das nicht reichen, muß die Folie zweimal übereinander gedruckt werden. Die Auswahl der Folie ist unproblematisch, besonders gut eignet sich zum Beispiel Zweckform 3491 (Laserfolien für die Druckvorlagenerstellung).
- Tintenstrahldrucker erreichen eine deutlich bessere Deckung. Die Einstellungen des Druckertreibers sollten auf höchstmögliche Auflösung und Qualität sowie monochromen Druck eingestellt sein. Zusammen mit der richtigen Folie entsteht dann ein sehr guter Ausdruck. Die Folie darf keinen optischen Aufheller enthalten und sollte leicht aufgerauht und staubfrei sein. Nach einer ausreichend langen Trockenzeit, die von der Tintenmenge abhängt, ist die Folie einsetzbar.

Die Belichtungsvorlage in Abbildung D.4 wurde mit einem Tintenstrahldrucker auf Folie gedruckt.

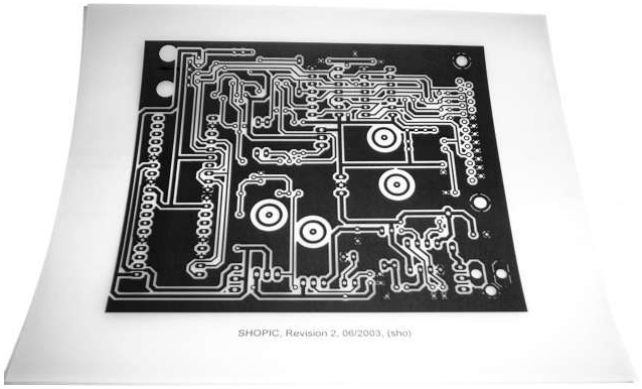


Abbildung D.4: Beispiel für eine Belichtungsvorlage

Die Folie wird in hellem Gegenlicht auf Deckungsgrad und eventuelle Lichtdurchbrüche kontrolliert. Kleine Fehler lassen sich notfalls mit einem Folienstift ausbessern, dann wird die Folie mit großzügigem Rand ausgeschnitten.

D.2 Platinenmaterial und seine Bearbeitung

Bei Platinen wird als Trägermaterial in der Regel Epoxyd oder Pertinax eingesetzt. Epoxyd ist ein stabiler Glasfaserverbundstoff, der in der industriellen Produktion fast ausschließlich eingesetzt wird. Er läßt sich allerdings mit normalen Werkzeugen nur schlecht bearbeiten, weil diese sehr schnell stumpf werden. Pertinax (auch Hartpapier genannt) erreicht nicht die Stabilität von Epoxyd, läßt sich dafür mit normalen Werkzeugen gut bearbeiten und ist daher in vielen Fällen die bessere Wahl. Der Herstellungsprozeß ist bei beiden Platinensorten aber im Prinzip identisch.

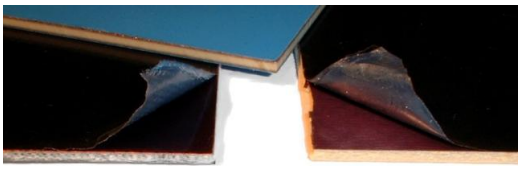


Abbildung D.5: Platinenmaterialien: Epoxyd (links) und Pertinax (rechts, oben)

Auf dem Trägermaterial sitzt eine Kupferschicht mit einer Dicke von normalerweise 35 µm. Es gibt Platinen auch mit 70 µm Kupferauflage, diese sind aber nur für Leistungselektronik erforderlich, um die dort auftretenden Ströme besser transportieren zu können. Häufig müssen aus einer handelsüblichen Platine mit Standardmaßen Teile der gewünschten Form und Größe herausgeschnitten werden. Dazu werden die Abmessungen auf dem Platinenmaterial angezeichnet, eine ausgedruckte Schablone in der passenden Größe kann dabei eine gute Hilfe sein kann. Dann wird das Stück mit einer Trennscheibe oder einer feine Metallsäge herausgeschnitten.

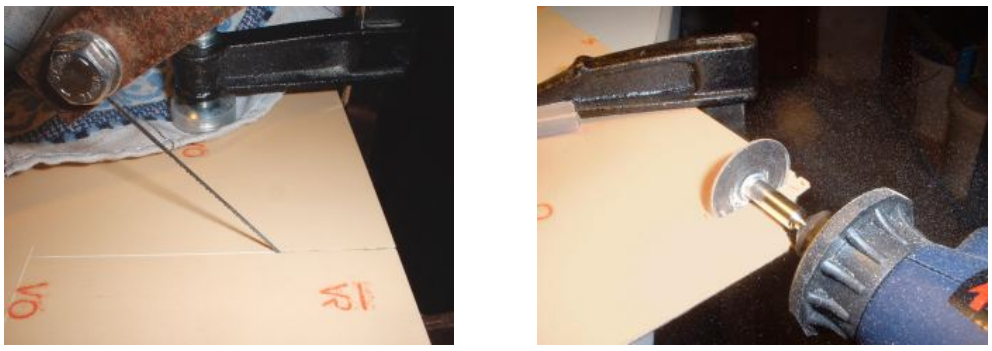


Abbildung D.6: Zuschnitt von Platinenmaterial

Der auf dem Kupfer aufliegende Fotolack altert relativ schnell und wird bereits ein Jahr nach der Produktion unbrauchbar. Er ist außerdem relativ empfindlich und darf beim Sägen der Platine keinen großen mechanischen Belastungen ausgesetzt werden. Schmutzkrümel oder eine zu fest angesetzte Schraubzwinge können die Schutzfolie beschädigen oder den Fotolack ankratzen. Im Zweifelsfall hilft Schaumstoff oder ein Lappen als Polsterung. Zu große Hitzeentwicklung, wie sie beim Arbeiten mit einer Trennscheibe auftreten kann, ist ebenfalls in der Lage, den Lack zu beschädigen. Daher ist es sinnvoll, die Platine erst etwas zu groß auszuschneiden und am Ende des Prozesses auf die genaue

Größe zu bringen. Die besten Ergebnisse lassen sich mit einer Laubsäge erzielen, insbesondere bei unregelmäßigen Formen. Zum Schluß werden die Kanten des ausgeschnittenen Stücks glattgeschliffen.

D.3 Fotochemisches Erzeugen der Leiterbahnen

Das Leiterbahnlayout kann jetzt mit Hilfe von UV-haltigem Licht von der Vorlagenfolie in den Fotolack der Platine übertragen werden. Damit die Ausleuchtung gleichmäßig und unter konstanten Bedingungen erfolgen kann, ist eine entsprechende Apparatur nötig, ein sogenannter Belichter. Er besteht im Wesentlichen aus einer Lichtquelle und einer Möglichkeit, Platine und Folie zu fixieren.

D.3.1 Aufbau eines Belichters

Bei der Konstruktion eines Belichters ist es wichtig, den UV-Anteil im Licht möglichst hoch zu halten. Dies beginnt mit der Auswahl einer Lichtquelle, die einen möglichst hohen UV-Anteil im Spektrum haben sollte. Beim Abstand der Lichtquelle zur Platine muß ein Kompromiß gefunden werden. Das Licht ist zwar um so intensiver, je geringer der Abstand zur Lampe ist, andererseits erwärmen Platine und Folie sich auch stärker und das Licht verteilt sich weniger gleichmäßig. Schließlich dürfen alle Bauteile im Lichtweg möglichst kein UV-Licht absorbieren. Dies gilt insbesondere für die Glasplatten, die es teilweise mit UV-Filter zu kaufen gibt.

Abbildung D.7: Belichter aus Leuchtstoffröhren in einem Scannergehäuse

Die Platinen für diese Arbeit wurden mit dem Belichter aus Abbildung D.7 hergestellt, der auf einer Idee von [Bredendiek] basiert. Das UV-haltige Licht wird von sieben Werkstattlampen erzeugt, die eine 8 W-Leuchtstoffröhre und die nötige Vorschaltelektronik enthalten. Die Röhren sind zu einem Paket zusammengefügt und in das Gehäuse eines Flachbettscanners eingebaut. Der Scannerdeckel ist mit schwarzem Filz beklebt, um zu verhindern, daß bei doppelseitigen Platinen die Rückseite mit belichtet wird. Außerdem sorgt der Deckel durch sein Gewicht für den nötigen Anpreßdruck.

Der Aufbau hat einige Vorteile. Er ist klein und verfügt über ein robustes Gehäuse. Die Folie kann nicht verrutschen und die Bedingungen beim Belichten sind immer gleich. Insbesondere können externe Lichtquellen durch die Klappe den Belichtungsvorgang nicht stören. Der Stromverbrauch ist mit 56 W relativ gering, die Röhren entwickeln kaum Wärme und die Belichtungszeit beträgt nur zwischen fünf und zehn Minuten, obwohl es sich nicht um spezielle UV-Röhren handelt.

D.3.2 Belichten der Platine

Zum Belichten wird das Licht im Raum gedämpft, die lichtundurchlässige Schutzfolie von der Platine abgezogen und die Platine mit Folie zügig in den Belichter gebracht. Dabei sollte nicht zu viel Zeit vergehen, um die Platine nicht dem Umgebungslicht auszusetzen. Der Belichter wird eingeschaltet und die Platine so lange wie nötig. Wie lang diese Zeit genau ist, muß durch eine Meßreihe ermittelt werden. Details zu dem Verfahren folgen weiter hinten in diesem Abschnitt.

D.3.3 Entwickeln der Platine

Während der Belichtungszeit kann bereits die Lösung zum Entwickeln hergestellt werden. Dazu wird Natriumhydroxid in Wasser gelöst, die Dosierung liegt in der Regel bei 1 g auf 100 ml Wasser. Beim Umgang mit der Lösung ist Vorsicht geboten, sie ist ätzend und greift auch Haut und Kleidung an. Es lohnt sich nicht, Entwickler auf Vorrat herzustellen, da die Lösung beim Kontakt mit dem Kohlendioxid der Luft zerfällt. In einer luftdicht verschlossenen Flasche ist sie aber einige Tage haltbar.

Abbildung D.8: Entwickeln der Platine in einer Fotoschale

Am Ende der Belichtungszeit wird der komplett gelöste Entwickler in eine Kunststoffschale gegossen und die Platine aus dem Belichter entnommen. Bei vielen Platinensorten sind die belichteten Stellen im Lack bereits an schwachen Farbunterschieden zu erkennen. Die Platine wird mit der Kupferseite nach oben für etwa eine Minute in die Entwicklerlösung gelegt. Währenddessen wird die Schale vorsichtig hin- und hergeschwenkt, so daß die Platine umspült wird. Am Ende der Minute müssen die belichteten Konturen hervortreten und deutlich erkennbar sein. Dabei kann eine Reihe von Fehlern auftreten.

- **Das Layout wird nicht sichtbar**: Die Konzentration des Entwicklers ist zu schwach, oder die Belichtungszeit war zu kurz.
- **Das Layout erscheint sehr schnell und verschwindet wieder**: Die Belichtungszeit war zu lang, oder die Konzentration des Entwicklers ist zu hoch.

- **Das Layout ist unscharf**: Die Vorlagenfolie ist nicht deckend genug oder wurde nicht fest genug auf die Platine gedrückt.
- **Das Layout ist fleckig**: Wahrscheinlich ist der Fotolack nicht mehr in Ordnung.
- **Das Layout läßt sich kaum herausarbeiten**: Möglicherweise ist das Platinenmaterial zu alt und der Fotolack funktioniert nicht mehr einwandfrei. Wenn Teile der Platine sofort und andere fast gar nicht erscheinen, kann die Platine unter Umständen noch gerettet werden. Dazu wird die Entwicklerlösung über den noch nicht sichtbaren Teilen des Layouts vorsichtig mit einem sehr weichen Pinsel bewegt. Der Vorgang beschleunigt sich dadurch lokal.

Die Konturen müssen scharf und deutlich herausgearbeitet werden, bevor der Entwickler den nicht belichteten Fotolack angreift. Die fertige Platine wird mit einer Zange aus dem Bad geholt und mit klarem Wasser abgespült. Sie sollte dann etwa wie in Abbildung D.9 aussehen.

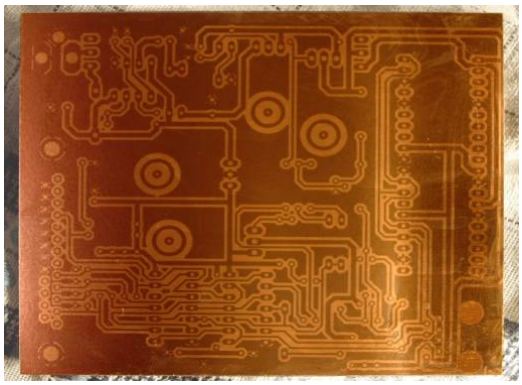


Abbildung D.9: Fertig entwickelte Fotoplatine

D.3.4 Die optimale Belichtungsdauer

Die Dauer der Belichtung ist entscheidend für die Platinenherstellung. Sie hängt von vielen Faktoren wie dem Platinenmaterial und dem Aufbau des Belichters ab. Es ist daher nicht möglich ist, allgemeingültige Richtwerte anzugeben, stattdessen muß die Zeit mit einer Meßreihe ermittelt werden. Dazu kann beispielsweise eine Folie mit zehn numerierten Bereichen ausgedruckt und auf einen Streifen Fotoplatine geklebt werden. Ein lichtundurchlässiges Stück Papier wird zwischen Platine und Folie geschoben (siehe Abbildung D.10) und alles zusammen in den eingeschalteten Belichter gebracht. Jetzt kann das Stück Papier in regelmäßigen Abständen ein wenig weiter herausgezogen werden, so daß ein weiterer Bereich dem Licht ausgesetzt ist.

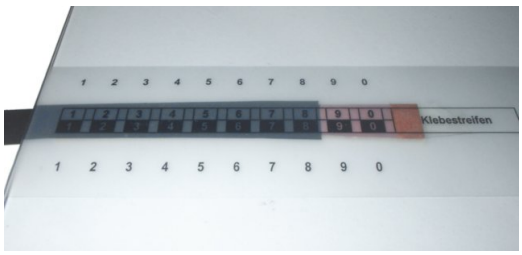


Abbildung D.10: Aufbau zur Ermittlung der optimalen Belichtungsdauer

Zu jedem Bereich wird protokolliert, wie lange er genau belichtet wurde, also die Zeit zwischen dem Freilegen des Bereiches und dem Ende der Meßreihe. Anschließend wird die Platine genau eine Minute

lang entwickelt. Dabei sollten idealerweise unter- und überbelichtete Bereiche an den Rändern sowie ein brauchbarer Bereich in der Mitte herauskommen. Ist das nicht der Fall, wird die Reihe für kürzere bzw. längere Zeiten wiederholt oder die Abstände werden entsprechend angepaßt. Ist der optimale Bereich gefunden, läßt sich mit Hilfe des Protokolls die dazugehörige Belichtungszeit ermitteln. Sie ist nur für das beim Messen verwendete Platinenmaterial und den benutzten Belichter richtig. Sobald sich nur einer dieser Parameter ändert, muß eine neue Meßreihe durchgeführt werden.

D.3.5 Ätzen der Platine

Nach dem Entwickeln wird die Platine in einen beheizbaren Ätztank gehängt, der mit einer Lösung von Natriumpersulfat gefüllt ist. Auf diese Weise wird das Kupfer an allen freigelegten Stellen entfernt, so daß die Leiterbahnen entstehen. Die Konzentration der Lösung liegt bei 270g auf einen Liter Wasser, sie ist klar und verfärbt sich nach mehreren Benutzungen blau.

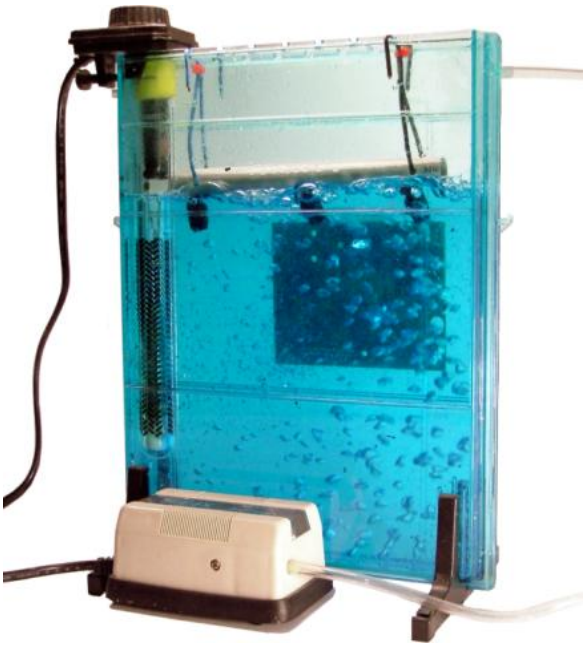


Abbildung D.11: Ätztank für Platinen

Die Temperatur im Tank muß um 50°C liegen. Unter 40°C ätzt die Lösung fast gar nicht, und bei mehr als 60°C kristallisiert sie aus. In der Regel ist in einen solchen Tank auch eine Reihe von Düsen eingebaut, mit denen Luft in die Lösung geblasen werden kann. Dadurch wird das Ätzmittel durchmischt und Sauerstoff eingebracht, was die chemische Reaktion beschleunigt. Die Bauteile für einen Ätztank gibt es zwar im Aquarienhandel zu kaufen, allerdings ist ein fertiger Tank unwesentlich teurer als die Einzelteile, so daß sich ein Selbstbau nicht lohnt.

Die entwickelte Platine wird an Kunststoffklammern aufgehängt und in den vorbereiteten Ätztank gehängt. Das freiliegende Kupfer wird sehr schnell matt und verfärbt sich dunkelrot. Es dauert einige Minuten, bis die Kupferschicht an den ersten Stellen ganz verschwunden ist, dann löst der Rest sich innerhalb kurzer Zeit auf. Wenn die Leiterbahnen komplett herausgearbeitet sind, wird die Platine in klarem Wasser abgewaschen und vorsichtig getrocknet. Beim Ätzen entstehen Gase, so daß für eine gute Belüftung gesorgt sein sollte. Die lagernde Säure bildet ebenfalls Gase, so daß sie nicht in einem luftdichten Behälter aufbewahrt werden darf. Das Ätzmittel darf nicht mit Haut oder Kleidung in Kontakt kommen und muß, wenn es verbraucht ist, als Sondermüll entsorgt werden.

D.4 Bohren, Beschriften und Versiegeln

Nach dem Ätzen folgen noch einige mechanische Bearbeitungsschritte. Als erstes werden die erforderlichen Löcher in die Platine gebohrt. Dazu ist eine schnell drehende Bohrmaschine mit Präzisionsbohrern und einem Bohrständer unbedingt erforderlich. Wenn jede Bohrstelle von einem Kupferring umgeben ist, richtet der Bohrer sich selbst in der Lochmitte aus. Es reicht dann, die Platine mit der Hand unter den Bohrer zu führen und mit wenig Druck und viel Gefühl zu bohren.

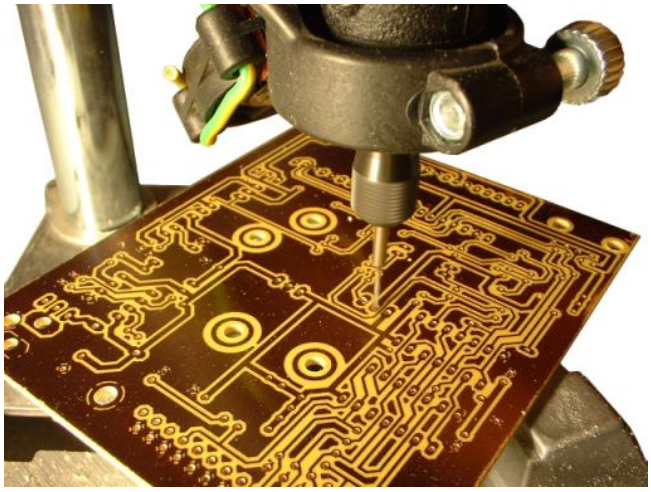


Abbildung D.12: Bohrständer, der kleine Ventilator bläst den Bohrstaub weg

Falls erforderlich, wird die Platine noch mit Feinwerkzeugen in die endgültige Form gebracht. Schließlich wird sie vom Bohrstaub befreit und ein letztes Mal gereinigt. Der verbliebene Fotolack läßt sich mit Spiritus entfernen.

Abbildung D.13: Transfer der Platinenbeschriftung

Die Platine kann anschließend mit einer Beschriftung versehen werden. Diese wird im Layoutprogramm vorbereitet und dann mit einem Laserdrucker spiegelverkehrt auf entsprechendes Transfermaterial gedruckt. Mit Hilfe eines möglichst stark aufgeheizten Bügeleisens wird der Toner vom Transfermaterial auf die Platine übertragen. Geeignete Transfermaterialien sind zum Beispiel Backpapier und die Trägerbögen, auf denen Aufkleber vertrieben werden.

Der Prozeß erfordert einige Übung, bevor die Beschriftung brauchbar wird. Das Bügeleisen wird zügig und mit gleichmäßiger Geschwindigkeit über den Träger gezogen, wobei ständig ein gewisser Druck ausgeübt werden muß. Anschließend wird das Bügeleisen auf die Seite gestellt und der Träger von einer Ecke aus langsam angehoben. Ist ein Teil der Beschriftung noch nicht übertragen, wird an dieser Stelle gezielt mit der Spitze des Bügeleisens nachgebessert.

Abbildung D.14: Lackierte Platine, zum Trocknen aufgestellt

Um die Oberflächen der Platine vor Korrosion der Leiterbahnen und Abrieb der Beschriftung zu schützen, wird sie auf beiden Seiten dünn mit Sprühlack überzogen. Die Kupferseite wird mit Lötlack versiegelt, der die Leiterbahnen schützt und die Lötbarkeit sicherstellt. Auf der Beschriftungsseite kann jeder Lack verwendet werden, der den Toner nicht ablöst, also zum Beispiel Lötlack oder transparentes Plastikspray. Die Lackschichten sollten die ganze Platine bedecken, dabei aber so dünn wie möglich sein. Beide Seiten müssen gründlich trocknen, dann kann die Platine bestückt werden.

D.5 Bestücken der Platine

Wenn das Platinenlayout gut gewesen ist, sollte es beim Bestücken keine Probleme geben. Die Bauteile müssen natürlich frei von Oxidschichten sein, notfalls können diese mit Schmirgelpapier oder einer feinen Drahtbürste entfernt werden. Wichtig ist auch, daß die Platine beim Löten nicht zu stark erhitzt wird, damit die Kupferschicht sich nicht vom Trägermaterial ablöst.

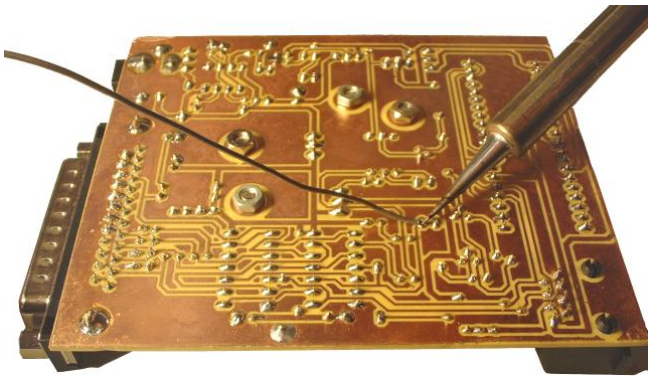


Abbildung D.15: Bestücken der fertigen Platine

Nach dem Löten ist die Platine voll einsatzbereit. Wenn der Prototyp auf der Steckplatine funktioniert hat und alle Schritte in der Platinenherstellung ohne Fehler ausgeführt wurden, wird die Platine mit

großer Wahrscheinlichkeit auch funktionieren. Nach einem kurzen Test kann die Platine an ihrem Bestimmungsort eingebaut und in Betrieb genommen werden.

D.6 Herstellung doppelseitiger Platinen

Die Herstellung von doppelseitigen Platinen verläuft im Prinzip genau wie bei einseitigen. Es werden zwei Folien für Ober- und Unterseite gedruckt und zu einer Tasche zusammengeklebt, wobei darauf geachtet werden muß, daß die Bohrlöcher genau aufeinander ausgerichtet werden. Reststücke von anderen Platinen dienen dabei als Abstandshalter und Randmarkierungen. Die Fotoplatine wird zum Belichten in die Tasche gesteckt und kann so umgedreht werden, ohne zu verrutschen. Auf diese Art werden beide Seiten nacheinander belichtet, bevor die Platine komplett entwickelt, geätzt und nachbehandelt wird, wie es auch bei einer einseitigen der Fall wäre.

Literaturverzeichnis

Lego Mindstorms, Sensoren und Aktoren

[Baum] Dave Baum, Michael Gasperi, Ralph Hempel, Luis Villa: Extreme Mindstorms – An Advanced Guide to LEGO Mindstorms. Apress 2000.
Das Buch gibt einen Überblick über die verschiedensten Projekte rund um Lego Mindstorms. Die Autoren zählen zu den bekanntesten Personen der Mindstorms Community.

[BrickOS] BrickOS project homepage: `http://brickos.sourceforge.net/`

[Gasperi] Michael Gasperi: Mindstorms RCX Sensor Input Page.
`http://www.plazaearth.com/usr/gasperi/lego.htm`
Die größte Sammlung von Informationen zum Thema Mindstorms Sensoren.

[Hurbain] Philippe Hurbain: Philo's Home Page. `http://www.philohome.com/`
Mindstorms-Projekte, Sensoren und detaillierte Informationen zu den Lego Motoren.

[Janz] Hendrik Janz: Codeerzeugung von Matlab-Simulink-Stateflow Modellen für Lego Mindstorms. Studienarbeit am Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme, Universität Kiel, März 2005.
Die Arbeit beschreibt ein invertiertes Pendel auf der Basis von Mindstorms.

[Knudsen] Jonathan Knudsen: The Unofficial Guide to LEGO Mindstorms Robots. O'Reilly, 1999.
Einführung in Mindstorms mit Vorstellung der verschiedenen Programmiersprachen.

[Krimer] Evgeni Krimer, Roman Barsky: BrickOS improvements project. Center and Laboratory for Intelligent Systems (CIS and ISL), Technion - Israel Institute of Technology.
`http://www.cs.technion.ac.il/~barsky/brickos_files/brickos.html`.
Eine Arbeit, die sich mit Fließkommazahlen unter brickOS beschäftigt.

[Ldraw] LDraw.org Centralized LDraw Resources: `http://www.ldraw.org/`
CAD-Programme und Utilities zum Erstellen von Lego-Modellen.

[Legokiel] Stephan Höhrmann: Lego Mindstorms.
`http://www.informatik.uni-kiel.de/inf/von-Hanxleden/mindstorms/`
Einführung in Lego Mindstorms von der Universität Kiel.

[Legores] Lego Mindstorms Advanced User Resources.
`http://mindstorms.lego.com/eng/community/resources/`
Links auf viele Ressourcen zu Mindstorms, unter anderem das Lego SDK.

[LegoUSB] LegoUSB project homepage: `http://sourceforge.net/projects/legousb/`

[lnphost] lnphost project homepage: `http://lnphost.sourceforge.net/`
Die in dieser Arbeit entwickelte C-Bibliothek zur Ansteuerung des Mindstorms Towers.

[Martin] Fred G. Martin: The Art of Lego Design. The Robotics Practitioner: The Journal for Robot Builders, volume 1, number 2, Spring 1995.
`http://www.eecs.cwru.edu/courses/lego375/papers/artoflego.pdf`
Mechanische Grundlagen von Lego Technic.

[Mientki] Mattijs, Stef und Robbert Mientki: Lego / Mindstorms / Knex.
`http://oase.uci.kun.nl/~mientki/Lego_Knex/`
Informationen zu Mindstorms Aktoren, Sensoren und Infrarotkommunikation.

[Nelson] Russell Nelson: Lego Mindstorms Internals.
`http://www.crynwr.com/lego-robotics/`
Umfangreiche Referenz, entspricht einer Mindstorms FAQ.

[Nielsson] Stig Nielsson: Introduction to the legOS kernel. September 2000.
`http://legos.sourceforge.net/docs/kerneldoc.pdf`
Übersichtliche Beschreibung der zentralen Funktionen des legOS/brickOS Kernels.

[NQC] NQC project homepage: `http://bricxcc.sourceforge.net/nqc/`

[Pfeifer] Tom Pfeifer: LEGO frequently asked questions.
`http://www.faqs.org/faqs/LEGO-faq/`
FAQ der Newsgroup rec.toys.lego, enthält Abmessungen der Legosteine.

[Proudfoot] Kekoa Proudfoot: Mindstorms Internals.
`http://graphics.stanford.edu/~kekoa/rcx/`
Reverse Engineering von Mindstorms, insbesondere Firmware und IR-Kommunikation.

Mikrocontroller und ihre Programmierung

[Atmel] Atmel Corporation: `http://www.atmel.com/`

[Bredendiek] Jörg Bredendieks Homepage: `http://www.sprut.de/`
Informationen zu PICs, Programmiergeräten, LC-Displays und Platinenherstellung.

[Gputils] GNU PIC Utilities project homepage: `http://gputils.sourceforge.net/`

[Intelhex] Intel Corporation: Intel Hexadecimal Object File Format Specification (Revision A), 6.1.1988. Weitere Beschreibungen des Originalformates und seiner zahlreicher Varianten sind auf `http://www.wotsit.org/` zu finden.

[Microchip] Microchip Technology Inc: `http://www.microchip.com/`

[Parport] `http://www.lvr.com/parport.htm`
`http://www.beyondlogic.org`
Technische Informationen zum Parallelport und zu anderen Schnittstellen im PC.

[Piclinks] C-Compiler für PIC Prozessoren:
`http://bknd.com/cc5x/`
`http://sdcc.sourceforge.net/`

Assembler Entwicklungsumgebungen für PIC Prozessoren:
`http://www.microchip.com/`
`http://pikdev.free.fr/`
`http://www.mtoussaint.de/yapide.html`

Allgemeine Informationen, Tutorials, Linksammlungen:
`http://www.voti.nl/swp/`

http://www.sprut.de/electronic/pic/
http://www.mikrocontroller.net/
http://www.piclist.com/
http://www.geocities.com/SiliconValley/Way/5807/dat.html

Verschiedene Beispiele für Programmiergeräte:
http://www.sprut.de/electronic/pic/brenner/
http://www.finitesite.com/d3jsys/
http://www.finitesite.com/d3jsys/proghvp.html
http://www.covingtoninnovations.com/noppp/
http://www.cs.uiowa.edu/~jones/cross/pic/
http://www.lancos.com/prog.html

[Tait] David Tait: Internet PIC Resources.
http://people.man.ac.uk/~mbhstdj/piclinks.html
David Taits Programmiergerät ist die Basis für viele andere Geräte.

Ortung per Ultraschall

[Borenstein] J. Borenstein, H. R. Everett, L. Feng. Where Am I? Sensors and Methods for Mobile Robot Positioning. University of Michigan, Department of Mechanical Engineering and Applied Mechanics, Mobile Robotics Laboratory, April 1996.
ftp://ftp.eecs.umich.edu/people/johannb/pos96rep.pdf
Umfassende Darstellung der Sensoren und Techniken zur Positionsbestimmung.

[Ghidary] Saeed Shiry Ghidary, Takahiro Tani, Toshi Takamori, Motofumi Hattori. A new Home Robot Positioning System (HRPS) using IR switched multi ultrasonic sensors. IEEE SMC Conference, Tokyo, Japan, October 1999.
http://citeseer.ist.psu.edu/ghidary99new.html
Ortungssystem, das einen Roboter in einem Raum lokalisieren kann. Der Roboter sendet Ultraschall- und Funkimpulse an eine Reihe von Empfängern an der Decke des Raumes.

[Jörg] Klaus-Werner Jörg. Echtzeitfähige Multisensor-Integration für Autonome Mobile Roboter. BI Wissenschaftsverlag, Reihe Informatik, Band 91, 1994. Herausgegeben von K. H. Böhling, U. Kulisch, H. Maurer.
Guter Überblick über den Stand der Technik bei Sensoren.

[Kleeman] Lindsay Kleeman. Optimal Estimation of Position and Heading for Mobile Robots Using Ultrasonic Beacons and Dead-reckoning. IEEE International Conference on Robotics and Automation, Nice, France, pp 2582-2587, May 10-15 1992.
http://calvin.eng.monash.edu.au/IRRC/LKPubs/ICRA92a.PDF
Beschreibt die Kombination eines Ultraschall-Ortungssystems mit Koppelnavigation.

[Mahajan] Ajay Mahajan, Maurice Walworth. 3D Position Sensing Under the Differences in the Time-of-Flights From a Wave Source to Various Receivers. IEEE Transactions on Robotics and Automation, vol. 17, no. 1, pp. 91-94, February 2001.
Aktualisierte Version unter http://asl.lssu.edu/asme_97_paper.htm
Berechnung von Positionen anhand von Laufzeitunterschieden.

[Marquardt] Marquardt Datenblatt: Air Ultrasonic Ceramic Transducers 400ST/R160

[Navarro] Luis E. Navarro-Serment, Christiaan J. J. Paredis, Pradeep K. Khosla. A Beacon System for the Localization of Distributed Robotic Teams. Proceedings of the International Conference on Field and Service Robotics, Pittsburgh, PA, August 29-31, 1999.

http://www.contrib.andrew.cmu.edu/usr/rjg/publications/FSR99.pdf
Beschreibung eines Ortungssystems für eine Gruppe von Robotern, wobei einige als stationäre Leuchtfeuer arbeiten und die Positionsdaten für die übrigen Roboter liefern.

[Restena] Convict Episcopal de Luxembourg.
http://www.restena.lu/convict/
Luxemburgisches Internat, dessen Lehrer und Schüler eine Vielzahl von Mindstorms-Projekten realisiert haben. Darunter befindet sich auch ein ein Ortungssystem auf der Basis von Ultraschall- und Infrarotsignalen.

[Turowski] Thomas Turowski: Ortsaufnahme durch Ultraschall mit besonderer Anwendung auf nichtlineare Systeme. Staatsexamensarbeit am Institut für Experimentalphysik der Christian-Albrechts-Universität zu Kiel, April 1996.
Bei dieser Examensarbeit wird ein System beschrieben, das die Position eines Pendels in zwei Dimensionen mit Hilfe eines Computers bestimmen kann.

Elektronik und Platinenherstellung

[Cadsoft] CadSoft Computer Inc: http://www.cadsoft.de/

[dseFAQ] FAQ der Newsgroup news:de.sci.electronics:
http://dse-faq.elektronik-kompendium.de/

[HorHill] Paul Horowitz, Winfield Hill: The Art of Electronics (second edition). Cambridge University Press, 1989.
Standardreferenz der Elektronik.

[TiSch] Ulrich Tietze, Christoph Schenk: Halbleiter-Schaltungstechnik (11. Auflage). Springer-Verlag, 1999.
Umfangreiches Standardwerk, weniger zum Lesen, aber sehr zum Nachschlagen geeignet.

Verschiedenes

[Messmer] Hans-Peter Messmer: PC-Hardwarebuch. Aufbau, Funktionsweise, Programmierung, Ein Handbuch nicht nur für Profis (3. Auflage). Addison-Wesley 1995.

[Strangio] Christopher E. Strangio: The RS232 Standard. A Tutorial with Signal Names and Definitions, CAMI Research Inc., Lexington, Massachusetts.
http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html
Gute, vollständige und verständliche Beschreibung des RS232/EIA232 Standards.

[Tischer] Michael Tischer: PC Intern 4. Systemprogrammierung (1. Auflage). Data Becker, 1994.
Referenz zu vielen Standardfunktionen und Interfaces im PC.

Inhalt der CD

Dieser Arbeit liegt eine CD bei, auf der alle erstellten Schaltungen, Platinenlayouts, Steuerprogramme Fotos und natürlich die Ausarbeitung gespeichert sind. Dazu kommen natürlich viele der verwendeten Quellen und Programme.

- **Ausarbeitung** - PDF-Datei mit dem Inhalt dieser Ausarbeitung
- **Dimensionierung** - Hilfsprogramme zum Ausrechnen der Dimensionierung einer Schaltung
- **Material** - Verwendete Materialien aus unterschiedlichen Quellen
  - **3D-Modelle** - LDraw-Modelle von Lego-Robotern
  - **Datasheets** - Datenblätter für alle Teile, die in der Arbeit verbaut worden sind
  - **Druckvorlagen** - Schablonen und Beschriftungen zum Ausdrucken
  - **Eagle** - Eigene und reparierte Bauteilbibliotheken sowie Skripte für Eagle
  - **Literatur** - Einige der zitierten Quellen und Dokumente
  - **Originalgrafiken** - Alle verwendeten Fotos in hoher Auflösung
  - **Software** - Verwendete externe Programme und Zusatzdateien
  - **Webseite** - Mindstorms-Webseite des Lehrstuhls Echtzeit- und eingebettete Systeme
- **Messwerte** - Die Ergebnisse der verschiedenen Messungen in Zahlen und Diagrammen
- **Programmcode** - Im Laufe dieser Arbeit entwickelte Programme
  - **Debugmonitor** - Monitorprogramm für das serielle Debug-Interface für Mikrocontroller
  - **Koordinaten** - Fehlersimulation für die Koordinatenberechnung
  - **LNP** - Programmierinterface für den Mindstorms-Tower (lnphost)
  - **Mindstorms** - Verschiedene Beispielprogramme für brickOS
  - **PIC** - Steuerprogramme für Schaltungen mit PIC Mikrocontrollern
  - **Programmer** - Programme zum Brennen und Verarbeiten der Firmware
  - **brickos-0.9.0** - brickOS mit allen Patches
- **Schaltungen** - Schaltpläne und Platinenlayouts
  - **Ausarbeitung** - Schaltpläne, die nur als Skizzen in der Arbeit enthalten sind
  - **Development** - Entwicklungssystem für PIC Mikrocontroller
  - **Joystick** - IR-Fernbedienung in Form eines Joysticks
  - **Mindstorms** - Schallsensor und allgemeines Sensorinterface
  - **Ortungssystem** - Die verschiedenen Module des Ortungssystems
  - **Timer** - Programmierbare Zeitschaltuhr zum Belichten von Platinen

Quellennachweise

Ein Teil der Abbildungen und Fotos aus dieser Arbeit stammt aus fremden Quellen und ist teilweise nachbearbeitet worden. Diese Quellen sind im Folgenden aufgeführt.

Abbildung 1.1: `http://www.walmart.com/`

Abbildung 1.2: `http://www.informatik.uni-kiel.de/inf/von-Hanxleden/`

Abbildung 1.4: `http://www.restena.lu/convict/`

Abbildung 2.1: `http://www.ceeo.tufts.edu/`

Abbildung 3.12: `http://pikdev.free.fr/`

Abbildung 9.5: `http://www.informatik.uni-kiel.de/inf/von-Hanxleden/`

Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig verfaßt und ausschließlich die angegebenen Hilfsmittel und Quellen verwendet habe.

Kiel, den _____ _____