hochschule für angewandte wissenschaften
FACHBEREICH ELEKTROTECHNIK UND INFORMATIK hamburg
university of applied sciences

Diplomarbeit

Oliver Dahlmann

Entwicklung eines kostengünstigen Distanzmesssystems für mobile Roboter

Studiengang Technische Informatik
Betreuender Prüfer: Prof. Dr. Gunter Klemke
Zweitgutachter: Prof. Dr. Kai von Luck
Abgegeben am 15. August 2003

Berliner Tor 3 · D-20099 Hamburg

Entwicklung eines kostengünstigen Distanzmesssystems für mobile Roboter

Stichworte

Mobiler Roboter, Ultraschall, Infrarot, RCX

Zusammenfassung

Diese Diplomarbeit beschreibt die Entwicklung eines kostengünstigen, auf der Basis von Ultraschall- und Infrarot-Sensoren basierenden Messsystems, das mit einem Mikrocontroller ausgestattet die Messwerte vorverarbeitet und dem Zielsystem in geeigneter Form einen kanonischen Entfernungswert übermittelt. Es erlaubt einen einfachen Anschluss an das verbreitete RCX-Modul des LEGO Robotik-Lehrsystems ‚Mindstorms'. Die Hardware basiert auf einem AVR-Microcontroller, die Software ist in Assembler implementiert. Für den im RCX integrierten Hitachi-H8-Controller wird eine C/Assembler-Library zur Kommunikation mit dem Modul bereitgestellt.

Development of a lowcost distance measurement system for use with mobile robots

Keywords

mobile robot, ultrasonic, infrared, RCX

Abstract

This paper describes the development of a lowcost, ultrasonic- and infrared-sensors based measuring system which equipped with a microcontroller preprocesses measured data and transmits a canonic distance value to the host system in an appropriate form. It permits a simple connection to the wide-spread RCX module of the LEGO Robotics Invention System 'Mindstorms'. The hardware is based on an Atmel AVR-Controller, the software is implemented in assembler. A C-Library for the Hitachi H8 microcontroller, integrated in the RCX Computer, has been developed.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die anhaltend sinkenden Preise und die zunehmende Komplexität mikroelektronischer Bauelemente bei gleichzeitig geringerem Stromverbrauch machen mobile Roboter nicht mehr nur in Forschung und Industrie, sondern auch im häuslichen Bereich in Form von Haushaltsrobotern und Lehrsystemen möglich.

Firmen wie iRobot versuchen sich erfolgreich an der Entwicklung und Markteinführung von mobilen Robotern (Abbildung 1.1), die weniger den Spaß am Basteln als vielmehr einen gewissen Teil zumeist unangenehmer Arbeit abzunehmen versprechen: „Der Erste seiner Art ist ein mobiler, autonom arbeitender Staubsauger namens Roomba. Die Philosophie, die hinter Roomba steckt, ist einfach: Um Roboter in jedes Haus zu bekommen, müssen sie vor allem eines sein: Einfach und billig.“ (Frei übersetztes Zitat aus einem Interview mit Brad Stone, [Stone]).

Abbildung 1.1: Staubsauger-Robot 'Roomba'

Nicht ganz diese Philosophie treffend, da mehr als siebenmal teurer (~ €1.500), kommt der etwas edler aussehende Trilobite der schwedischen Firma Electrolux daher (Abbildung 1.2).

Dafür erhält der Käufer einen Staubsauger, der selbständig den Weg zurück zur Ladestation findet und sich während des Saugens die Positionen der Hindernisse merkt.

Abbildung 1.2: Trilobite Staubsauger-Roboter

Auch mobile Roboter, die ohne fremdes Zutun innerhalb einer von Begrenzungsobjekten aufgespannten Fläche vollautomatisch den Rasen mähen, befinden sich bereits auf dem Weg zur kommerziellen Nutzung (Massey University in Zusammenarbeit mit dem Mischkonzern Husqvarna, siehe [Husqvarna]). Das Prinzip unterscheidet sich nicht wesentlich vom Prinzip der beiden Staubsauger, der Entwicklungstand kann indes noch nicht mit jenen mithalten. Zur Zeit ist noch ein auf dem Rasenmäher montiertes Notebook für die Steuerung erforderlich.

Für Ausbildung und Lehre, aber auch als Spielzeug mit einem grossen Spaßfaktor, hat der Spielzeughersteller Lego vor einigen Jahren mit dem Mindstorms Robotics Invention System einen in Zusammenarbeit mit dem Massachusetts Institute of Technology entstandenen Baukasten vorgestellt, mit dem sich mobile Roboter bauen lassen (Abbildung 1.3).

Handelsübliche Legosteine und einige Spezialteile wie Motoren, Sensoren und der RCX, ein batteriebetriebener Computer mit integrierten Motortreibern und Sensoranschlüssen sind die Komponenten dieses Baukastens.

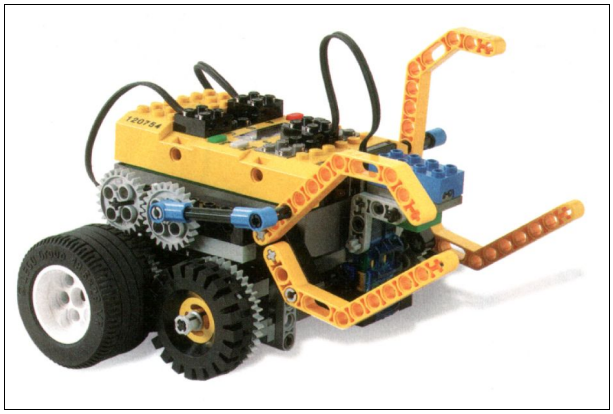

Abbildung 1.3: Mobiler Roboter, mit dem Lego Robotics Invention System gebaut

Allen mobilen Systemen ist eines gemein: „An intelligent robot is a machine able to extract information from its environment and use knowledge about its world to move safely in a meaningful and purposive manner.“ (Zitat aus [Arkin]). Ein 'intelligenter' Roboter muss also in der Lage sein, Informationen aus seiner Umgebung zu ziehen, und Wissen über die Welt zu nutzen, um sich sicher in geordneter Weise bewegen zu können.

Um dies zu gewährleisten, werden Sensoren benötigt, die Umwelteindrücke in geeignete, vom Zielsystem interpretierbare Signale umsetzen.

Die dem Lego Mindstorms Robotics Invention System beiliegenden Sensoren (einfache Taster für Berührungserkennung sowie ein Infrarot-Sensor für die Erkennung kontrastreicher Linien auf dem Untergrund) reichen für die ersten Programmier-Erfahrungen. Im Raum befindliche Hindernisse lassen sich hiermit jedoch nicht berührungslos erkennen und umfahren.

Die Entwicklung eines kostengünstigen Messsystems für die Hindernis-Erkennung und Distanzmessung, dass sich an den Lego-RCX-Baustein anschliessen lässt, ist Gegenstand dieser Diplomarbeit.

Da ein Sensor allein oftmals nicht genügend Informationen liefert bzw. in einer Situation versagt, in denen ein anderer Sensortyp bessere Ergebnisse liefert, sollen mehrere Sensoren verschiedenen Typs von einem Microcontroller abgefragt werden. Aus den verschiedenen und voneinander mehr oder weniger stark abweichenden Messwerten soll ein kanonischer Messwert gebildet werden.

1.2 Aufbau der Arbeit

Kapitel 2 analysiert die Anforderungen an ein geeignetes Distanzmessmodul und beleuchtet den Hardwaremarkt hinsichtlich für mobile Roboter geeigneter Sensoren. In Kapitel 3 wird ein Design für das Messmodul erarbeitet und in Kapitel 4 umgesetzt. Kapitel 5 bietet eine Zusammenfassung und einen Ausblick.

2 Analyse

2.1 Anforderungen

Die Hochschule für Angewandte Wissenschaften Hamburg hat in den Aufbau eines Robotik-Bereiches investiert, um es ihren Studenten zu ermöglichen, Erfahrungen in der Programmierung und Entwicklung mobiler Roboter u.a. auf Basis des Lego Mindstorms Robotics Invention Systems, im Folgenden kurz Mindstorms RIS genannt, zu sammeln.

Mindstorms RIS basiert auf einem Computer (RCX) mit drei PWM-Motorausgängen, drei analogen Sensor-Eingängen und einer Infrarot-Schnittstelle zur Kommunikation mit einem PC.

Im Lieferumfang des Mindstorms RIS sind Taster, die den direkten Berührungs-Kontakt zu einem Hindernis signalisieren und ein Licht-Sensor, der den erkannten Helligkeitsbereich in ein analoges Signal umsetzt. Es gibt jedoch keinen Sensor, der eine Distanzmessung zu Objekten ermöglichen würde. Ein solches Sensor-Modul soll hier konzipiert werden.

Das RCX-Modul wird im Auslieferungszustand mit einer auf symbolischen Ablaufplänen basierenden grafischen Sprache programmiert, kurze Zeit nach Erscheinen des Systems entstand jedoch die OpenSource Betriebssystem-Alternative LEGos, mittlerweile in brickOS umbenannt, die eine Programmierung des RCX-Bausteins in C, C++ und Assembler ermöglicht. Im RCX-Modul kommt ein Mikrocontroller vom Typ Hitachi/Renesas H8/3292 zum Einsatz. Hierbei handelt es sich um einen Mikrocontroller mit H8/300-Kern, 16Bit- und 8Bit-Timer, A/D-Wandler (8 Kanal, 10Bit) und 43 I/O-Leitungen. Die Registerbreite beträgt 16Bit, Low- und Highbyte jedes Registers sind jedoch auch getrennt ansprechbar.

Das Betriebssystem brickOS sowie die selbstgeschriebenen Programme lassen sich komfortabel ohne jegliche Hardware-Änderungen am RCX-Baustein über die Infrarot-Verbindung übermitteln. Für das Betriebssystem und die eigenen Programme steht eine Speicherkapazität von 32KB zur Verfügung.

2.1.1 Hardware-Anforderungen

Folgende Anforderungen an die Hardware sollen erfüllt werden:

- geringe Grösse, möglichst in zu LEGO-Bausteinen mechanisch kompatibles Gehäuse eingebaut

- geringes Gewicht, um die Mobilität des Roboters nicht einzuschränken
- geringer Strombedarf, Versorgung durch Batterie oder Akkumulator

2.1.2 Funktionale Anforderungen

Die funktionalen Anforderungen, die an das Modul gestellt werden:

- ausreichend genaue Entfernungsmessung zu verschiedenen Objekten, wie Wänden, Personen, Möbeln im Nahbereich von einigen Zentimetern bis zu einigen Metern
- elektrische und softwaretechnische Kompatibilität zu LEGO RCX-System
- leichte Verwendbarkeit in eigenen brickOS-Programmen

2.1.3 Sonstige Anforderungen

Sonstige Anforderungen, die sich durch die Zielgruppe, nämlich studentische Projekte und Hobby-Elektroniker, ergeben:

- geringer Preis
- möglichst Verwendung von Standardbauteilen
- Nachbaubarkeit

2.2 Aktive und passive Sensoren

„Sensoren können unterteilt werden in passive und aktive Sensoren. Passive Sensoren leiten Entfernungsinformationen aus der Umgebungsenergie ab, typischerweise durch die Analyse mehrerer Videobilder einer Szene. Entweder werden gleichzeitig aufgenommene Bilder mehrerer Kameras oder räumlich versetzt aufgenommene Bilder einer Kamera benutzt. ...“ (aus dem Englischen übersetzt aus [Singh, West]).

Aus dem Wissen um den räumlichen Abstand der einzelnen Bilder lassen sich Distanzen zwischen oder zu Objekten errechnen.

„Im Gegensatz dazu fügen aktive Sensoren der Szenerie Energie hinzu und sind somit nicht abhängig von Umgebungslichtbedingungen...“ ([Singh, West]).

Aktive Infrarot-Sensoren senden also Infrarot-Licht aus, um die hierbei entstehende Reflektion messen zu können. Ebenso senden Ultraschall-Sensoren ein kurzes Ultraschall-Paket aus, um dessen Reflektion messen zu können.

2.3 Marktanalyse

Im Folgenden soll untersucht werden, welche Möglichkeiten der Markt bietet, wenn es um Positions-, Richtungs- und Objektdistanz-Sensoren in Form von Fertigmodulen oder Bausätzen für mobile Roboter geht. Hierbei wird nicht nur auf zum LEGO-System kompatible Module eingegangen, da sich alle analogen Messgrössen mit skalarem Charakter mit vertretbarem Aufwand in einen vom RCX verarbeitbaren Wert umsetzen lassen.

2.3.1 Infrarot-Sensoren

Lynxmotion: Die Firma Lynxmotion bietet kleine, recht preiswerte Sensor-Module (Abbildung 2.1) für den Einbau in eigene Roboter-Konstruktionen an, u.a. ein Infrarot-Modul (Part No. #3-573, $34,95), das Hindernisse, die das ausgestrahlte Infrarotlicht reflektieren, melden kann.
Zu diesem Zweck ist ein Infrarot-Sensor von Panasonic vom Typ PNA4602M (in der vorigen Version war es ein SHARP GP1U581Y) zusammen mit zwei Infrarot-LED's auf einer kleinen Platine untergebracht.
Mittels der zwei einzeln ansteuerbaren LED's ist es möglich, ein links, rechts oder frontal zum Sensor liegendes Hindernis zu detektieren. Zwei rote 3mm-LED's dienen als visuelle Kontrolle: Erkennt der Sensor ein Hindernis, so leuchtet die der gerade leuchtenden IR-LED benachbarte rote LED.
Eine Entfernungsmessung wird nicht direkt durchgeführt, vielmehr lässt sich über die LED-Helligkeit ein Schwellwert im Bereich von etwa 8“ bis 26“, abhängig von den Objekt-Eigenschaften, vorgeben.
Mittels zweier Potentiometer lassen sich LED-Helligkeit und Trägerfrequenz (durch Modulation des LED-Stromes und durch Verwendung eines frequenzselektiven Empfängers ist Betrieb bei direkter Sonnenlichteinstahlung möglich) verändern. Die Modulationsfrequenz liegt bei etwa 38kHz (Schaltplan siehe Abbildung 2.2).
Der Anschluss an das Zielsystem geschieht über zwei Leitungen, mit denen die linke und rechte LED einzeln angesteuert werden können und eine Leitung, deren Pegel ein Hindernis meldet (siehe [Robotstore1]).

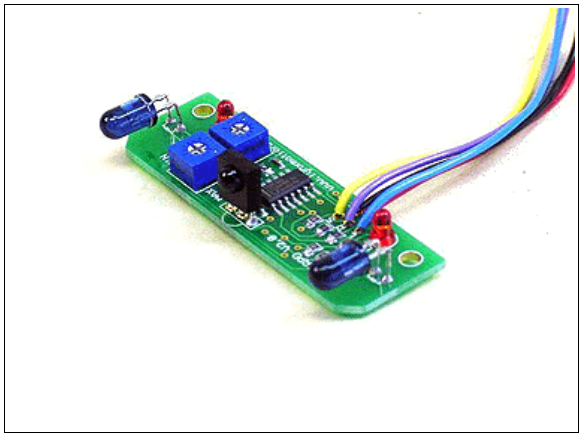

Abbildung 2.1: Infrarot-Sensor (Platine)

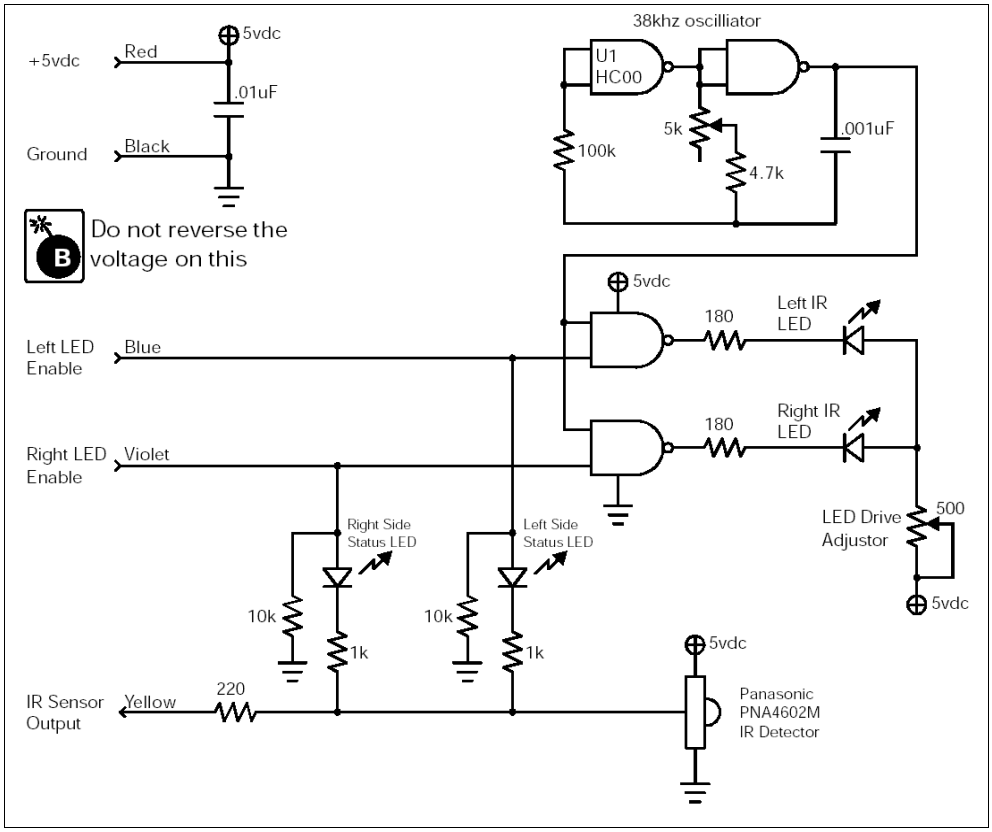

Abbildung 2.2: Infrarot-Sensor (Schaltplan)

Von SHARP gibt es einige sehr kleine und leichte Sensoren (Übersicht unter [SharpIR]), mit denen es möglich ist, Entfernungen zu Objekten sehr viel genauer aufzulösen als mit dem zuvor beschriebenen. Diese Sensoren

arbeiten alle nach dem Triangulationsprinzip, auf das später in dieser Arbeit näher eingegangen wird. Die Unterschiede zwischen den Sensoren liegen im Messbereich und im Interface:

Alle Sensoren verfügen über ein Drei- oder Vierdraht-Interface, wobei eine Leitung für die positive Versorgungsspannung (V_{CC} 5V), eine andere mit Masse (GND) belegt ist.

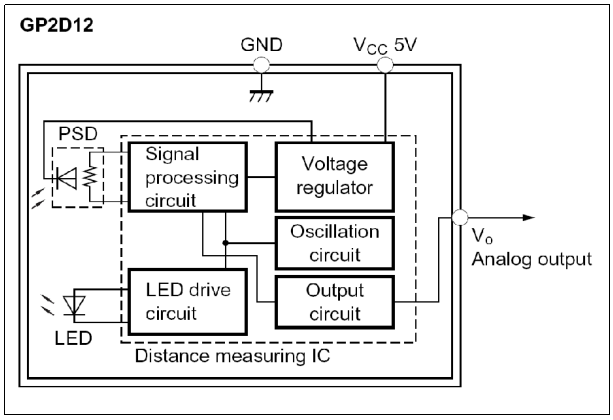

Abbildung 2.3: Blockschaltbild Sharp GP2D12

Die dritte Leitung (V_O) ist der Signalausgang: Während andere Typen einen digitalen Ausgang besitzen, der bei Über- oder Unterschreiten einer fest eingestellten Distanz zum Objekt den Pegel wechselt, verfügt der GP2D12 über einen analogen Ausgang, der im Bereich von 10 cm – 80 cm einen Spannungshub von etwa 2 V ausgibt.

Einer dieser Sensoren (GPD2D02) verfügt über einen seriellen Digital-Ausgang, wobei eine vierte Leitung für ein Taktsignal erforderlich ist. Dieser Sensortyp ist jedoch auf der entsprechenden Webseite des Herstellers nicht (mehr) zu finden, daher wird auf ihn im Folgenden nicht weiter eingegangen.

Der Spannungshub des Ausgangs des GP2D12 lässt sich über einen A/D-Wandler in digitale Werte umwandeln und vom Mikrocontroller weiterverarbeiten.

Der Sharp-Sensor arbeitet aktiv, d.h. er beleuchtet die Szenerie mit einer Infrarot-Leuchtdiode. Der Lichtstrahl ist stark gebündelt. Ein Position Sensitive Detector (PSD), der in einem festen Abstand und Winkel zur Lichtquelle angebracht ist, wandelt das vom Objekt reflektierte Licht in eine Spannung um.

2.3.2 Ultraschall-Sensoren

Beim Robotstore sind ebenfalls Ultraschall-Sensoren erhältlich, hier Ultraschall-Owl-Scanner (Part No. #3-705, $129) genannt. Dieses Modul basiert auf dem Instrument-Grade-Sensor von Polaroid, bringt jedoch eine eigene Platine (siehe Abbildung 2.4) mit, auf der die Pulserzeugung, Ultraschallempfänger sowie ein Mikrocontroller (BasicStamp2) mit seriellem Interface untergebracht sind. Mit dem Zielsystem wird über die serielle Verbindung mit 9.600 Baud kommuniziert.

Einfache Befehle (repetierende Messung, Einzelmessung) können hierüber übermittelt werden. Der Messbereich liegt zwischen 15 cm und etwa 2,70 m, die Auflösung beträgt etwa 1 cm. Der Messwert wird ausserdem als Analog-Signal ausgegeben (0-5 V). Ein Servo zum Drehen des Ultraschallsensors lässt sich anschliessen (siehe [Robotstore2]).

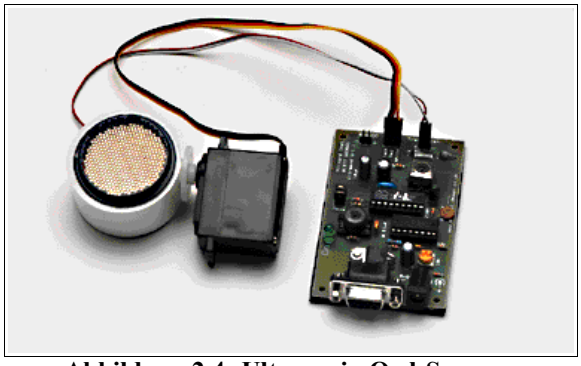

Abbildung 2.4: Ultrasonic-Owl-Scanner

Bei Robot Electronics, UK, ist das SRF08 Ultra sonic range finder Modul (Abbildung 2.5, 2.6) zum Preis von £25,50 erhältlich. Die Kommunikation mit dem Zielsystem findet über einen I2C-Bus statt, als Controller kommt auf dem Board ein PIC16F872 zum Einsatz. Der Erkennungsbereich liegt bei 3 cm bis etwa 6 m, die analoge Verstärkung des Ultraschall-Empfangsverstärkers ist in 32 Schritten einstellbar. Das Modul arbeitet mit getrennten Kapseln für Ultraschall-Sender und –Empfänger. Zusätzlich bietet das Modul einen Lichtsensor in Form eines LDR (lichtabhängiger Widerstand). In einer speziellen Betriebsart, die für die Weiterverarbeitung der Entfernungs-Daten in neuronalen Netzwerken besonders geeignet sein soll, arbeitet das Modul mit verringerter Messgenauigkeit (der Messbereich wird in eine zweistellige Anzahl diskreter Bereiche unterteilt), jedoch der Möglichkeit, mehrere Echos von hintereinanderliegenden Objekten zu erfassen.

Genaue Informationen zur Entstehungsgeschichte des Moduls, Software und Schaltplan findet man unter [SRF08].

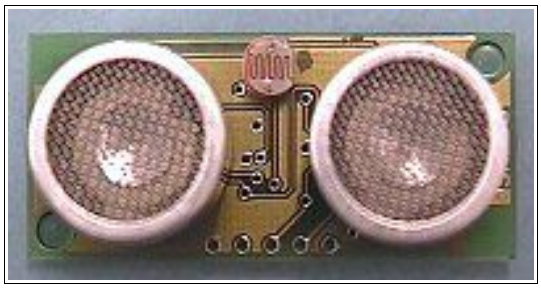

Abbildung 2.5: SRF08 Ultrasonic Range Finder

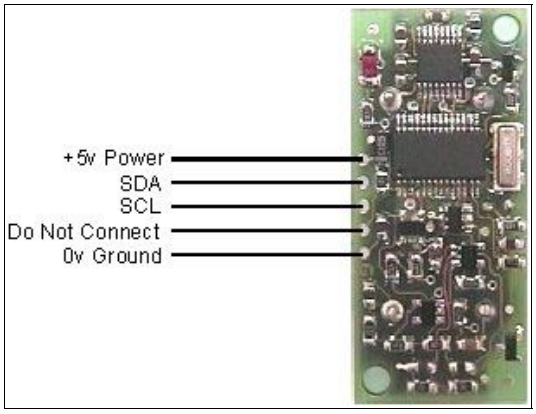

Abbildung 2.6: SRF08 Platine (Lötseite)

Vom Hersteller des beim Ultrasonic-Owl-Scanner verwendeten Sensors, Polaroid, ist eine Platine (siehe auch Abbildung 2.7) erhältlich, die die komplette Elektronik zur Ultraschall-Impuls- und Hochspannungserzeugung, Verstärkung und Echodetektion beinhaltet.

Diese Platine kann mit wenig Aufwand von einem Mikrocontroller angesteuert werden. Nötig ist nur ein Timer zur Messung der Zeitdifferenz zwischen Pulserzeugung und Reflektionserkennung.

Der daran verwendete Ultraschall-Sensor (Typbezeichung 'Series 600, Instrument Grade') dient gleichermaßen als Ultraschallsender und -empfänger und wird daher auch Ultraschall-Transducer bezeichnet.

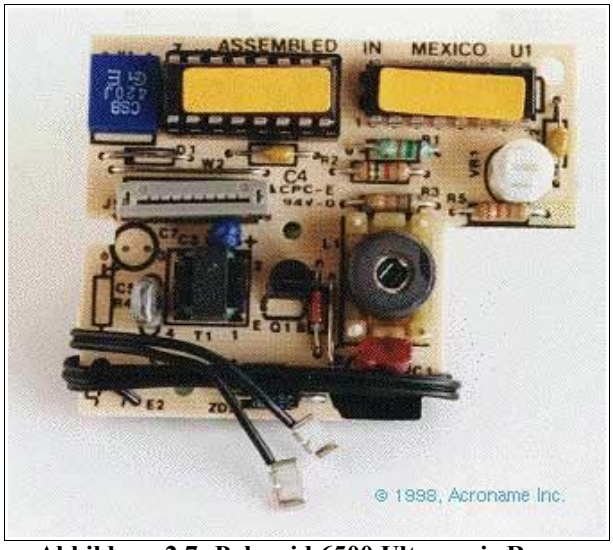

Abbildung 2.7: Polaroid 6500 Ultrasonic Range Finder

In Verbindung mit dieser Platine liegt der Messbereich bei etwa 15 cm bis maximal 5 m, abhängig von den Objekt-Eigenschaften.

Die verwendeten Ultraschallkapseln sind keine für mobile Roboter neuentwickelten Sensortypen: Sie werden schon seit Jahrzehnten in Kameras oder Bewegungsmeldern für den Einbruchmeldeanlagenbereich verwendet.

Hier werden mit Hilfe von Ultraschall- oder Infrarot-Sensoren Räume auf Veränderungen, wie Bewegungen von Personen und anderen Objekten, überwacht. Das abgestrahlte Ultraschall-Signal im Bereich von 20 bis 50kHz wird durch sich bewegende Objekte mit leicht veränderter Frequenz (Dopplereffekt) reflektiert, was sich mit wenig Aufwand auswerten lässt.

Die Abbildung 2.8 zeigt die Platinen zweier Ultraschall-Bewegungsmelder. Die rechte, kleinere, jüngere und grösstenteils in SMD-Technik bestückte Platine ist aus einem sogenannten Dualmelder entnommen, der zusätzlich zur Ultraschallmessung eine Infrarot-Messung (PIR-Prinzip) durchführt. Beide Melder arbeiten mit getrennten Ultraschall-Sender- und Ultraschall-Empfänger-Kapseln wie im bereits beschriebenen SRF08.

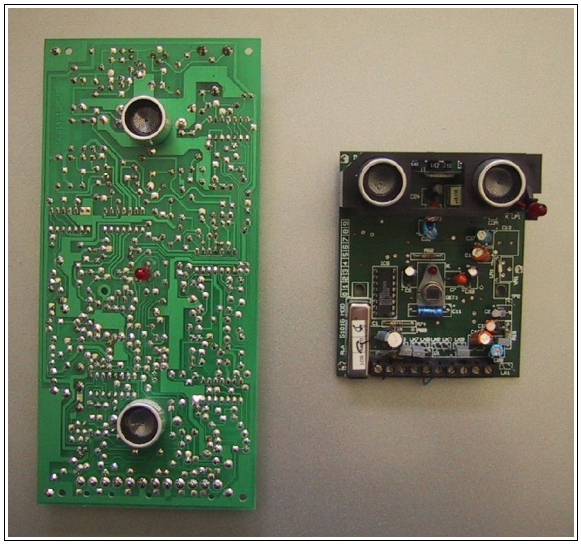

Abbildung 2.8: Bewegungsmelder

2.3.3 Laserscanner

Herkömmliche Laserscanner, die z.B. benutzt werden um Gefahrenbereiche abzusichern, tasten nur eine radiale Zeile der Umgebung ab. Über eine mit einem rotierenden Spiegel realisierte Ablenkeinheit wird ein modulierter (gepulster) Laserstrahl horizontal über das zu erfassende Areal geführt. Das Laserlicht wird vom Objekt reflektiert, die Zeit zwischen Aussendung eines Pulses und dem Empfang der Reflektion wird gemessen. Das Messergebnis ist direkt proportional zur Entfernung (siehe Abbildung 2.9, Funktionsweise Laserscanner).

Bei einer Ausbreitungsgeschwindigkeit des Lichts von etwa 300.000 km/s legt das Licht die Strecke von 20 cm in etwa $6{,}66 \cdot 10^{-11}$ s zurück, also 66,6 ps. Der kurze Lichtpuls wird vom Hindernis zum Scanner zurückreflektiert, legt also bis zur Detektion die doppelte Entfernung zwischen Scanner und Hindernis zurück. Will man daher die Entfernung mit ungefähr ±5 cm Genauigkeit messen, so werden Timer benötigt, die etwa 60 ps Auflösung aufweisen, also mit einer Taktfrequenz von etwa 3 GHz arbeiten. Der Laser muss sich entsprechend schnell in seiner Intensität modulieren lassen.

Der 2D-Laserscanner PLS des Herstellers Sick AG ([Sick]) verfügt über eine RS-422 / RS-232-Schnittstelle, auf der die Einzelmesswerte eines Zeilen-Scans ausgegeben werden.

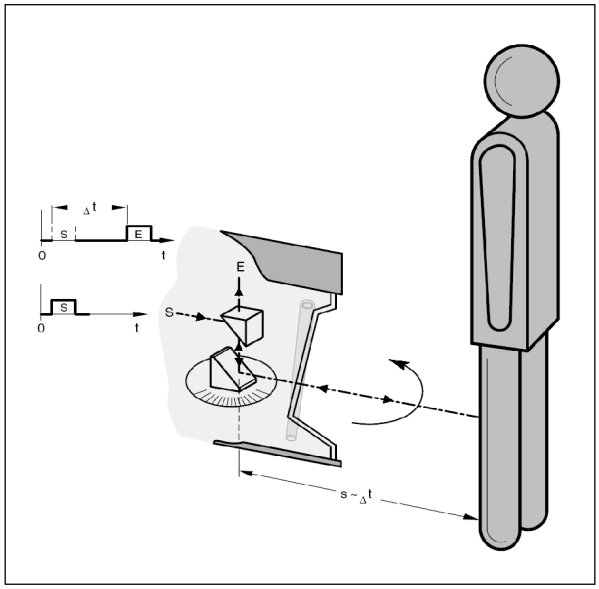

Abbildung 2.9: Funktionsweise Laserscanner

Das Fraunhofer-Institut hat mit dem AID-3D (Abbildung 2.10) einen auf dem SICK PLS-Scanner basierenden 3D-Scanner vorgestellt. An das Gehäuse des 2D-Scanners wurde ein Servo-Motor angebracht, der es ermöglicht, einen vertikalen Winkel von 90° zu überstreichen. Hiermit lassen sich nun nacheinander mehrere Zeilen erfassen, so dass sich eine Matrix aus Entfernungswerten ergibt.

Der AID-3D ermittelt in 12 Sekunden die Entfernung zu 115200 Punkten in einem Bereich von 150° Horizontal und 90° Vertikal. Durch Verringerung der Punktanzahl auf 22500 erhöht sich die Geschwindigkeit auf 4 s / Erfassung. Die maximal messbare Entfernung zum Objekt liegt bei 60m. Bei einer Objekt-Distanz von 60m muss das Licht also 120m Entfernung zurücklegen, dies entspricht einer Zeitdifferenz von etwa 200ns.

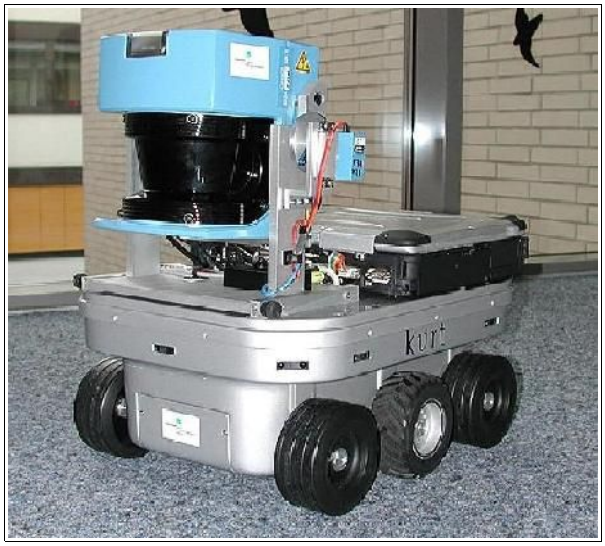

Abbildung 2.10: mit dem AID-3D ausgestatteter Roboter (Kurt2, Fraunhofer Institut)

Laserscanner sind aufgrund der aufwendigen Optoelektronik und Mechanik noch sehr teuer. Sie liefern aufgrund der hohen Menge an Messwerten ein sehr viel besseres Bild von der Umwelt, als dies einfache Ultraschall- oder Infrarot-Sensoren können, jedoch geht hiermit auch ein sehr viel höherer Aufwand in der Verarbeitung der gelieferten Daten einher.

2.3.4 Kameras

Ein interessantes kommerziell erhältliches Modul ist das CMUcam Vision System, eine kleine CCD-Kamera mit eigenem Mikrocontroller, das selbständig die Position und Grösse (bei bekannter Objekt-Grösse die Entfernung) von sich farbig oder aufgrund ihrer Helligkeit vom Hintergrund abhebenden Objekte bestimmen kann (Abbildung 2.11).

Das Modul wird über ein serielles Interface mit dem Zielsystem verbunden. Hierüber können neben Objekt-Position auch Informationen zur Farbverteilung des Gesamtbildes oder die komplette Bitmap eines Bildes abgefragt werden.

Eine der Betriebsarten erlaubt die Erfassung und automatische Verfolgung (Tracking) des ersten, nach dem Einschalten gesehenen Objektes. Dies kann auch physisch durch ein angeschlossenes, die Kamera nachführendes Servo geschehen, das vom Mikrocontroller des Moduls gesteuert wird.

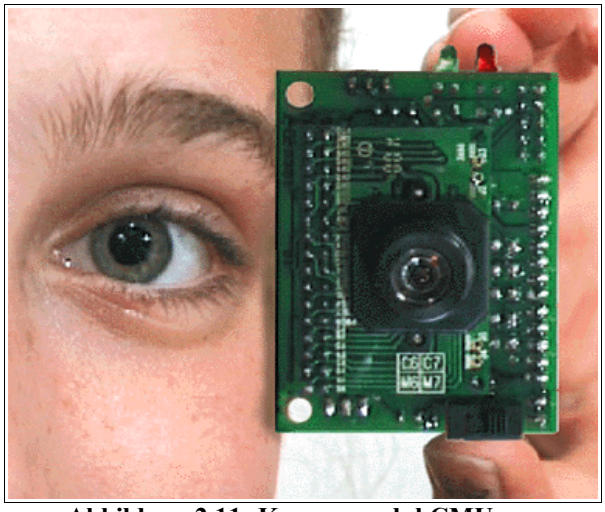

Abbildung 2.11: Kameramodul CMUcam

Bei entsprechendem Aufwand kann mittels einer Kamera auch eine Entfernungsbestimmung durchgeführt werden. Hierzu müssen dem Zielsystem Angaben über die genaue Grösse aller im Blickfeld vorkommen könnenden Objekte zur Verfügung stehen, weiterhin muss das Zielsystem in der Lage sein, die sichtbaren Objekte zu erkennen (zu unterscheiden).

Eine weitere Möglichkeit ist die Verwendung von (mindestens) zwei Kameras, die dem Roboter eine Stereosicht ermöglichen. Die Kameras sind hierzu nebeneinander in einem bekannten Abstand montiert und betrachten die Szenerie mit leicht verschiedenen Winkeln, ähnlich wie alle Säugetiere dies mit ihren Augen tun. Aus den beiden Bildern lässt sich mit recht hohem Rechenaufwand die Entfernung zu den erfassten Objekten bestimmen.

Während die bisher aufgeführten Sensoren sowohl eine Bestimmung der Position in statischen, kartografierten Räumen als auch die Erkennung von Objekten in dynamischen Szenarien ermöglichen, sollen nun als Ergänzung Sensoren betrachtet werden, die zur Positions- und Richtungsbestimmung des Roboters geeignet sind.

2.3.5 Kompasse

Von der Firma Devantech gibt es ein preiswertes ($46) Kompass-Modul, das mit einer Genauigkeit von 3 - 4 Grad sehr gute Ergebnisse erzielt (Abbildung 2.12). Es lässt sich durch seine zwei Ausgänge universell einsetzen und mit verschiedensten Zielsystemen bei wenig Aufwand koppeln.

Ausgang1 sendet ein pulsbreiten-moduliertes Rechtecksignal, dessen Pulsbreite mit 1 ms bis 37 ms proportional zum gemessenen Winkel von 0..360° ist.

Ausgang2 liefert über ein I2C-Interface ein serielles Bitmuster (0..255 oder 0..3599).

Der Kompass basiert auf zwei in 90°-Winkel angeordneten Magnetfeld-Sensoren von Philips (KMZ51), die auf magnetoresistiven Wheatstone-Messbrücken basieren.

Der KMZ51 verfügt über integrierte Spulen, um eine Messbrücke, die einem starken Magnetfeld ausgesetzt war, wieder zu kompensieren. (Datenblatt unter [KMZ51]).

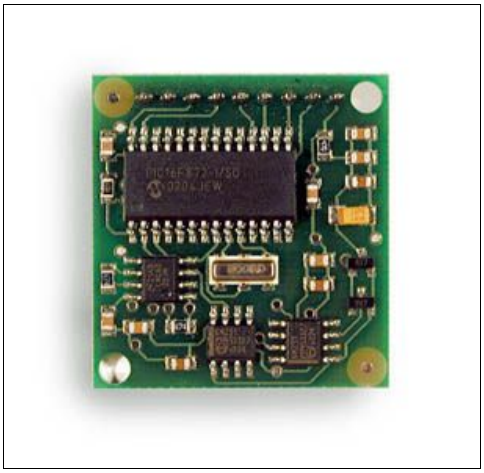

Abbildung 2.12: Kompass Devantech

Das von RobotStore angebotene Kompass-Modul basiert auf dem Dinsmore Sensor 1490, einem mechanischen, flüssigkeitsgefüllten Magnet-Kompass, dessen Nadelwinkelstellung mit Magnetfeldsensoren nach dem Hall-Prinzip ausgewertet wird: Vier Hall-Sensoren (einer pro Himmelsrichtung) mit Open-Kollektor-Ausgängen sind im Kompass enthalten. Hiermit lassen sich 8 Richtungen (N, NO, O, SO, S, SW, W, NW) detektieren, also eine Winkelgenauigkeit von 45° erzielen (Abbildung 2.13).

Durch den mechanischen Aufbau und der daraus resultierenden Massenträgheit der Kompassnadel ist jede Bewegung des Roboters mit Schwingungen der Nadel verbunden.

Mit schlechterer Genauigkeit und wahrscheinlich auch sehr viel grösserer Anfälligkeit gegenüber äußeren Magnetfeldern als der KMZ51 lässt sich der 1490-Sensor jedoch sehr viel einfacher als dieser mit beliebigen Mikrocontrollern verbinden.
Alternativ ist von Dinsmore ein Kompass-Sensor mit analogen Hall-Sensor-Ausgängen verfügbar, der sich über A/D-Wandler genauer abfragen lässt. Das Problem der schwingenden Masse, das beim KMZ51 prinzipbedingt nicht auftritt, bleibt jedoch.

Abbildung 2.13: Kompass (mechanisch, mit Hall-Sensoren)

2.3.6 GPS-Empfänger

GPS (Global Positioning System) wurde von den USA zunächst für militärische Zwecke, wie z.B. genaue Zielführung von Raketen, eingeführt. Seit einigen Jahren ist GPS auch zivil nutzbar, wovon viele in Kraftfahrzeuge und in der Schifffahrt eingesetzte Navigationssysteme Gebrauch machen.

GPS-Empfänger messen die Entfernung zu mehreren Satelliten, indem die Laufzeit der Satelliten-Signale über empfangene Zeitstempel errechnet wird. Hiermit lässt sich eine sehr genaue Positionsbestimmung durchführen.
Der in der Abbildung 2.14 gezeigte, etwa kreditkartengrosse GPS-Empfänger arbeitet mit 12 Kanälen, der Preis des Moduls liegt bei etwa $70 (weitere Informationen auf der Webseite des Herstellers: [GPS]).

Im Sekundenabstand werden Positions-Messungen durchgeführt. Zwei serielle Ports mit TTL-kompatiblen Signalen stehen zur Anbindung an das Zielsystem zur Verfügung. Die Stromaufnahme liegt bei durchschnittlich 50mA (3,3V).

Abbildung 2.14: GPS-Empfänger

2.3.7 Radar-Sensor KMY24

Das Wort Radar kommt aus dem Englischen und steht als Abkürzung für „**Ra**dio **D**etection **a**nd **R**anging“. „...Ein Radargerät arbeitet im Frequenzbereich von 500 MHz bis 40 GHz, was Wellenlängen von 60 cm bis 0,75 cm entspricht. Das Funktionsprinzip beruht auf der Tatsache, dass Funkwellen von festen Gegenständen, besonders von solchen aus Metall, reflektiert werden.“ [Radar]

Günstige Radar-Sensoren wie der Siemens KMY24 (€51,10, Conrad Elektronik) geben Aufschluss über die Differenzgeschwindigkeit des mobilen Systems zu Objekten der Umgebung. Der KMY24 strahlt über die in das Gehäuse integrierte Antenne ein Mikrowellensignal (f_0 = 2,45 GHz) ab.

Objekte, die sich im Ausbreitungsfeld des Signals befinden, reflektieren dieses. Bewegen sich die Objekte, verändert sich das Signal geringfügig in der Frequenz (im Bereich einiger Hz, Dopplereffekt).

Das reflektierte Signal wird vom KMY24 wieder empfangen und über eine Mischerstufe, aufgebaut aus zwei Mischerdioden mit dem Signal des HF-Oszillators gemischt. Die dabei auftretende Differenzfrequenz ist proportional zur Geschwindigkeit des Objektes, der Phasengang der beiden Ausgänge liefert Informationen über die Bewegungsrichtung.

Die Entfernung zu einem Objekt lässt sich mit diesem Sensor nicht bestimmen. [KMY24]

Mit Radar-Sensoren lassen sich auch Entfernungsmessungen vornehmen, indem die Laufzeit eines Impulspaketes bestimmt wird:

Bei einer Ausbreitungsgeschwindigkeit der Mikrowellen nahe der Lichtgeschwindigkeit ist eine hohe zeitliche Auflösung nur mit hohem Aufwand möglich: Ein 1 m vom Sender entferntes Objekt würde unter der Annahme, die Ausbreitungsgeschwindigkeit des elektromagnetischen Feldes würde $3 \cdot 10^8$ m/s betragen, Signallaufzeiten von ungefähr 6,6 ns hervorrufen. Mit den in Mikrocontrollern integrierten Timern lassen sich derart kleine Laufzeiten nicht bestimmen, mit externen GHz-tauglichen Zählerbausteinen wäre eine Genauigkeit im Dezimeter-Bereich denkbar.

Eine andere Möglichkeit wäre die Messung der Phasendifferenz des abgestrahlten und des empfangenen Signals: Auch hieraus lässt sich die Entfernung zu einem Objekt bestimmen, jedoch nur im Bereich einer Wellenlänge (Die Länge einer Periode von f_0 = 500 MHz beträgt etwa 60 cm).

Eine Kombination beider Verfahren (Laufzeit-Bestimmung mit einem kurzen Impulspaket und Phasendifferenz-Messung) dürfte zu genauen Ergebnissen auch bei geringen Entfernungen führen.

2.3.8 Fazit

Es gibt mittlerweile ein breites Spektrum unterschiedlichster Sensoren, die für mobile Roboter geeignet und dabei trotzdem, mit Ausnahme des Laserscanners, kostengünstig sind.

Im Folgenden eine tabellarische Zusammenfassung der bisher untersuchten Sensortypen:

Sensortyp	***Kosten***	***Aufwand bei Auswertung***	***Gewicht***	***Liefert Daten über***
Infrarot	gering	gering	gering	Distanz
Ultraschall	gering	gering/mittel	gering	Distanz
Kamera	mittel	mittel/hoch	gering	Farbe und Helligkeit, bei hohem Aufwand Distanz
Kompass	mittel	gering/mittel	gering	absolute Ausrichtung
GPS	mittel	gering/mittel	gering/ mittel	absolute Position
Laserscanner	hoch	hoch	hoch	Distanz (zu sehr vielen Messpunkten), Reflektionseigenschaften

Einige Sensoren wie Kompasse sind nur für die Richtungsbestimmung geeignet, während Sensoren wie GPS-Empfänger genaue Informationen über die Position des mobilen Roboters liefern können.
Kamerabasierte Sensoren lassen sich leicht einsetzen, wenn nur Objekte erkannt werden sollen, die sich anhand ihrer Farbe oder ihrer Helligkeit von der Umgebung abheben. Entfernungsmessungen sind nur mit beträchtlichem Aufwand möglich.

Laserscanner sind für die Entfernungsmessung und die automatische Kartierung sehr gut geeignet, jedoch für den hier verfolgten Anwendungszweck (noch) viel zu teuer. Da ultraschnelle Laser samt passender Treiberbausteine aufgrund der Massenfertigung immer billiger werden (kommen in CD- und DVD-Writern zum Einsatz, von der Leistung für geringere Entfernungen sicher ausreichend) wird sich dies sicherlich in der nächsten Zeit ändern.

Solange Laserscanner jedoch nur in extrem hohen Preisregionen zu finden sind, werden Ultraschall- und Infrarot-Sensoren für die Distanzmessung weiterhin für viele Anwendungen die erste Wahl sein.

3 Design

3.1 Distanzmessmodul

Das Distanzmessmodul soll zu geringen Kosten produzierbar sein. Es soll ein in geschlossenen Räumen interessanter Messbereich von 0 mm (Berührung) bis hin zu einigen Metern erfasst werden.

Um den gewünschten Bereich mit akzeptabler Genauigkeit zu erfassen, werden mehrere Sensoren verwendet:

- Zwei an der Front des Roboters angebrachte Taster, die die Berührung eines Hindernisses signalisieren,
- ein Infrarot-Sensor für die Entfernungsbestimmung im Nahbereich (Entfernung <1 m)
- sowie ein Ultraschall-Sensor, der Entfernungen im Bereich einiger cm bis zu einigen Metern, je nach Grösse und Beschaffenheit des Hindernisses, erfassen kann.

Das Modul soll einen analogen Wert zur Verfügung stellen, der vom beschriebenen RCX-Computer weiterverarbeitet werden kann. Um universellen Einsatz zu gewährleisten, soll auch ein serieller RS-232C-Anschluss für die Datenausgabe und Funktionssteuerung genutzt werden können.

Der Stromverbrauch soll im Hinblick auf den Einsatzzweck möglichst gering sein, so dass eine Speisung durch Batterien/Akkumulatoren auch über längere Zeit möglich ist.

3.2 Polaroid Ultraschall-Sensor

Der Ultraschall-Sensor (Typbezeichung 'Series 600, Instrument Grade') von Polaroid, im Folgenden nur Polaroid-Sensor genannt, ist ein nach dem piezoelektrischen Prinzip arbeitender Sensor. Er besteht im wesentlichen aus einer vergoldeten Membran, die als Kontaktfläche für eine Piezo-Kristallscheibe dient, die innerhalb eines runden Metallgehäuses mit etwa 3,8 cm Durchmesser untergebracht ist.

Der Sensor dient gleichermassen als Ultraschallsender und -empfänger und wird daher auch Ultraschall-Transducer bezeichnet. Anlegen einer hohen elektrischen Spannung führt zur schnellen mechanischen Verformung des Piezokristalls. Umgekehrt führen mechanische Bewegungen der Membran zur Erzeugung einer Spannung.

Die Entfernungsmessung geschieht nach folgendem Prinzip: Ein kurzes Ultraschallpaket, typischerweise 16 Schwingungen mit einer Frequenz von etwa 50kHz, wird abgestrahlt. Hierzu sind Impulse mit einer Spannungsstärke von etwa $400V_{SS}$ erforderlich.

Nach Aussendung des Impulspakets, in englischsprachigen Funktionsbeschreibungen Burst genannt, wartet die Auswerteelektronik einige Millisekunden, um die Eigenschwingungen des Sensors nicht als Echo zu interpretieren.

Nach Ablauf dieser Wartezeit wird die Kapsel als Empfänger benutzt. Ein in Ausbreitungsrichtung stehendes Objekt reflektiert den Burst zur Ultraschallkapsel zurück. Die zeitliche Differenz zwischen Burst-Aussendung und Empfang kann gemessen werden.

Aus der gemessenen Zeitdifferenz kann die Entfernung mit hoher Genauigkeit errechnet werden. Durchdringt der Schall vor der Reflektion andere Medien als Luft, so kann das Messergebnis verfälscht werden (siehe [Physik], S.394). Weiterhin ist zu beachten, dass die Schallgeschwindigkeit von der Temperatur beeinflusst wird (siehe [Schallgeschwindigkeit]). Mit hinreichender Genauigkeit gilt:

$$c \approx 331\,m/s + 0{,}6 \cdot \vartheta \; mit \; \vartheta = Temperatur\,[^\circ C]$$

Bei einer Temperatur von 20°C gilt also c ≈ 343m/s.

Wird die Laufzeit des Schalls von Aussendung bis zur Echo-Detektion gemessen, so ergibt als Entfernung zwischen Objekt und Sensor:

$$Entfernung\,[mm] = \frac{c \cdot Laufzeit\,[\mu s]}{2 \cdot 1000}$$

In der Formel ist berücksichtigt, dass der Schall wegen des Hin- und Rücklaufs die doppelte Strecke zurückgelegt hat.

Mit zunehmender Entfernung des Objekts nehmen die reflektierten Schallwellen in der Intensität ab, daher ist ein Verstärker erforderlich, dessen Verstärkungsfaktor (auch Gain-Faktor genannt) in Abhängigkeit von der Zeit erhöht werden kann.

Vom Hersteller Polaroid wird passend zu diesem Sensor eine Platine angeboten, auf der die Impuls- und Hochspannungserzeugung, Verstärkung

und Echodetektion untergebracht ist. Diese Platine kann mit wenig Aufwand von einem Mikrocontroller angesteuert werden.

In Verbindung mit dieser Platine liegt der Messbereich bei etwa 15 cm bis 5 m.

3.3 Sharp Infrarot-Sensor

Der Infrarot-Sensor GP2D12 des Herstellers SHARP, im Folgenden nur noch Sharp-Sensor genannt, ist ein Bauteil aus einer Familie verschiedener Infrarot-Sensoren, die alle nach dem Triangulationsprinzip arbeiten.

Die Sensoren unterscheiden sich hinsichtlich des Messbereichs (10 cm - 80 cm bzw. 20 cm - 150 cm) und dem Punktdurchmesser des Infrarot-Lichtstrahls (Divergenz).

Der Spannungshub des Ausgangs des GP2D12 lässt sich über einen A/D-Wandler in digitale Werte umwandeln und vom Mikrocontroller weiterverarbeiten. Alternativ kann ein digital arbeitender Sensor (GP2D02) verwendet werden.

Der Sharp-Sensor arbeitet aktiv, d.h. er beleuchtet die Szenerie mit einer Infrarot-Leuchtdiode. Der Lichtstrahl ist stark gebündelt. Ein Position Sensitive Detector (PSD), der in einem festen Abstand und Winkel zur Lichtquelle angebracht ist, wandelt das vom Objekt reflektierte Licht in eine Spannung um. Hierbei ist weniger entscheidend, mit welcher Intensität das Licht auf den PSD strahlt, als vielmehr der Ort, der auf dem länglichen Bauteil beleuchtet wird. Dieser ist maßgeblich von der Entfernung zum Objekt abhängig.

Aus der Zeichnung (Abbildung 3.1) ist ersichtlich, dass die Auflösung im Nahbereich deutlich besser sein muss als bei höheren Entfernungen, da der Winkel und die Entfernung nicht proportional zueinander sind.

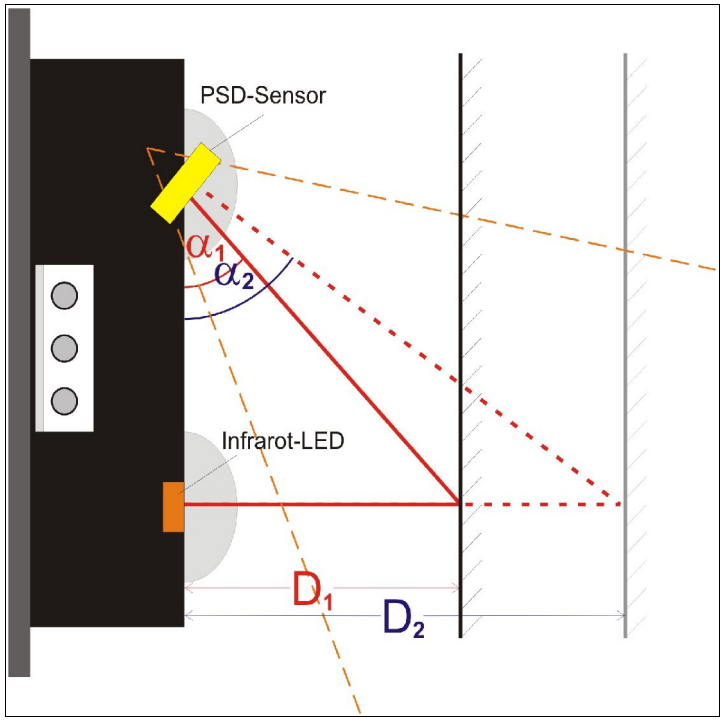

Abbildung 3.1: Triangulationsprinzip

Eine Schwäche des Sensors ist, dass der Ausgangswert nicht monoton fallend zur Entfernung verläuft. Bei Entfernungen unterhalb von etwa 8 cm nimmt der Spannungswert ebenso wie bei grösser werdenden Entfernungen oberhalb von 8 cm ab. Um einem Messwert also einen eindeutigen Entfernungswert zuordnen zu können, muss

a) entweder mechanisch dafür gesorgt werden, dass eine Mindestentfernung zwischen Objekt und Sensor nicht unterschritten werden kann oder

b) ein weiterer Sensor das Unter-/Überschreiten des Messwert-Maximums erkennen, so dass die nachfolgende Messwertverarbeitung den Fehler korrigieren kann.

Beim Einsatz des hier entwickelten Distanz-Messmoduls ist darauf zu achten, eine Mindestentfernung zum Messobjekt einzuhalten, damit der beschriebene Fall nicht eintreten kann.

Nachfolgend einige Grafiken (Abbildung 3.2 bis 3.6, Quelle: [Breheny]), die den Zusammenhang zwischen dem Messwert in Abhängigkeit der Entfernung bei verschiedenen Farben und Abmessungen des Objekts verdeutlichen. Die Messergebnisse basieren auf dem Sensor GP2D02, der

im Gegensatz zum besprochenen GP2D12 über einen digitalen, bitseriellen Ausgang verfügt. Von außen wird ein Taktsignal angelegt, der Sensor taktet daraufhin ein 8bittiges Messergebnis auf den Ausgang.

Zu sehen ist, dass Höhe und Breite des Objektes keinen grossen Einfluss auf die Messgenauigkeit ausüben. Die Farbe eines Objekts hingegen kann das Messergebnis signifikant beeinflussen, oberhalb Entfernungen von etwa 20 cm beträgt der Messfehler zwischen einem weißen und einem schwarzen Objekt bis zu etwa 10-15 cm.

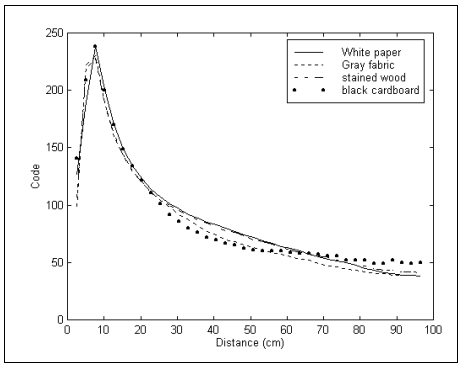

Abbildung 3.2: Abhängigkeit von Materialeigenschaften, -farbe

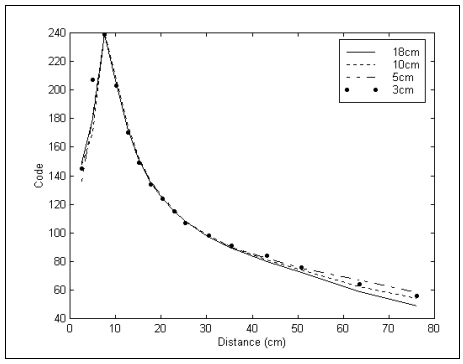

Abbildung 3.3: Abhängigkeit von der Höhe des Objekts

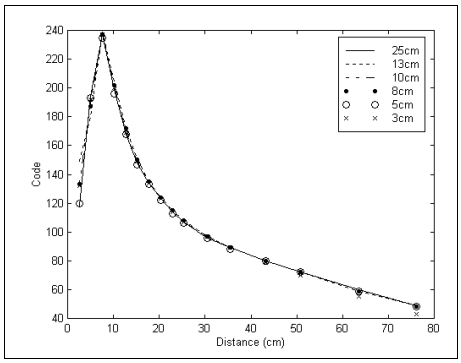

Abbildung 3.4: Abhängigkeit von der Breite des Objekts

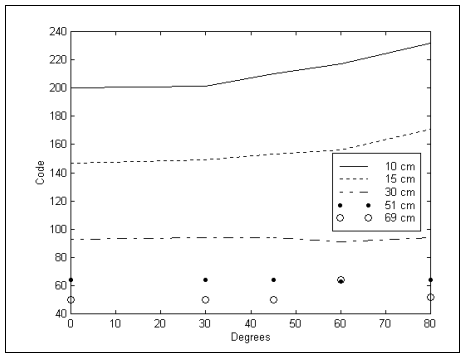

Abbildung 3.5: Abhängigkeit vom horizontalen Winkel (Graues Objekt)

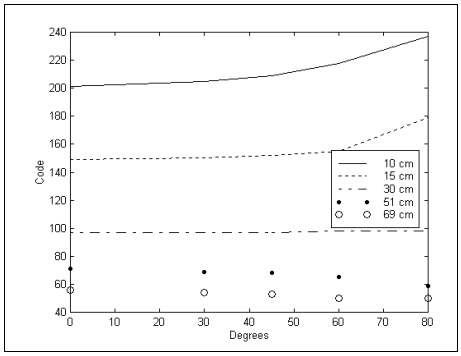

Abbildung 3.6: Abhängigkeit vom horizontalen Winkel (Weißes Objekt)

4 Realisierung

4.1 Aspekte zur Auswahl des Mikrocontrollers

Bei der Auswahl des Mikrocontrollers sind verschiedene Aspekte zu beachten:
Stromverbrauch, Grösse und somit Platinenfläche, Integration verschiedener Funktionsblöcke, Geschwindigkeit, Preis, Verfügbarkeit und nicht zuletzt die Entwicklungsumgebung.

Würde es nur auf Preis, Verfügbarkeit und in grosser Zahl vorhandener, auch freier, Entwicklungsumgebungen ankommen, könnte man sicher auf den seit über 20 Jahren eingesetzten Ur-Mikrocontroller 8031 zurückgreifen, der ursprünglich von Intel entwickelt wurde, später aber von vielen Herstellern, u.a. Philips, produziert wurde.

Was ihn von anderen aus dieser Zeit stammenden Mikroprozessoren unterscheidet, ist das Vorhandensein integrierter Funktionsblöcke wie Timer und UART (serielle Schnittstelle), was die Anzahl externer Komponenten auf der Platine deutlich verringert hat.

Ein grosser Nachteil dieses Controllers gegenüber modernen ist jedoch, dass wesentliche Bestandteile des Gesamtsystems wie RAM und ROM (oder auch E(E)PROM als Programm- und Tabellen-Speicher) nicht im Mikrocontroller integriert sind, sondern in Form externer Komponenten zum Einsatz kommen müssen. Dies verursacht zusätzliche Kosten und belegt gerade bei mobilen Systemen wertvolle Platinenfläche.

Ein weiterer wesentlicher Nachteil, der sich aus einem nicht in den Controller integrierten Programm-Speicher ergibt, tritt bei der Software-Entwicklung oder späteren Software-Updates auf: Ein solches System lässt sich nicht in-circuit programmieren, das E(E)PROM muss zur Neuprogrammierung aus dem Sockel genommen und in einem EPROM-Programmiergerät ggf. gelöscht und neu beschrieben werden.

Moderne Controller bieten internes RAM und einen internen nichtflüchtigen Programm- und Datenspeicher, der meist als Flashspeicher oder EEPROM realisiert ist. Sie lassen sich in der Schaltung (über eine serielle Verbindung) programmieren.

Um Strom- und Platinenfläche sparen zu können, wird die Wahl auf einen Mikrocontroller fallen, der über folgende Merkmale verfügt:

- integrierter A/D-Wandler für die Abfrage von Sensoren, die nur analoge Werte liefern (hier: SHARP GP2D12)
- integrierter UART, damit mit möglichst wenig externem Aufwand Kommunikation mit einem Terminal (PC) stattfinden kann
- integrierte Timer zur Zeitmessung (zur Laufzeitmessung des Ultraschall-Signals)
- integrierter RAM- und ROM-Speicher
- möglichst kleine Bauform
- möglichst geringer Strombedarf

Nicht zuletzt entscheidet die Leistungsfähigkeit der verfügbaren Entwicklungsumgebung über die Auswahl des geeigneten Controllers.

4.2 Mögliche Mikrocontroller

4.2.1 Infineon C5xx (vormals Siemens SAB 80C5xx)

Infineon C5xx-Mikrocontroller. Arbeitet nach dem CISC-Prinzip (Complex Instruction Set Computer), verfügt also über viele und komplexe Befehle. Diese Mikrocontroller-Familie ist schon länger am Markt verfügbar, einige Controller bieten 3 Timer, einen UART, internes RAM und ROM. Leider ist das interne ROM entweder maskenprogrammierbar, muss also vom Hersteller beschrieben werden, oder OTP (One Time Programmable, nur einmal programmierbar). Alternativ kann jedoch ein externer Programmspeicher in Form eines E(E)PROMs zum Einsatz kommen, für diesen Einsatz aufgrund der erforderlichen zusätzlichen Fläche generell ein Nachteil.

4.2.2 Microchip PICmicro

PICmicro®-Mikrocontroller des Herstellers Microchip. Dieser nach dem RISC-Prinzip (Reduced Instruction Set Computer) arbeitende Mikrocontroller lässt sich mit wenigen, einfachen Befehlen programmieren, die dafür jedoch sehr viel schneller ausgeführt werden als die Befehle eines CISC-Prozessors. PICmicro®-Mikrocontroller sind in vielen verschiedenen Grössen und Ausstattungen (Anzahl der I/O-Leitungen, Gehäuseversionen, mit/ohne A/D-Wandler, etc.) verfügbar.

In Frage käme z.B. der 16F870 (das ‚F' in der Typenbezeichnung steht für Flash, also mehrfach programmierbar, ‚C'-Typen sind OTP) mit folgenden Merkmalen:

- Befehlssatz mit 35 verschiedenen Befehlen
- 2KWord-Programmspeicher, 14Bit Befehlswortbreite
- 128 Bytes RAM Datenspeicher

- 64 Byte EEPROM Datenspeicher
- bis zu 11 Interrupt-Quellen
- SLEEP-Stromspar-Mode
- zwei 8Bit- und einem 16Bit-Timer
- 10Bit A/D-Wandler mit 5 Eingängen (Multiplexer)
- integrierter USART (Universal Synchronous Asynchronous Receiver Transmitter)
- integrierter Hardwarestack mit 8 Ebenen

Der 16F870 verbraucht sehr wenig Strom (unter 1,6 mA bei 4MHz Taktfrequenz) und lässt sich in der Schaltung programmieren. Eine Programmierumgebung für die Programmierung in Assembler ist kostenlos auf der Webseite des Herstellers verfügbar. [Microchip]

4.2.3 Atmel AVR

Atmel AVR 8Bit-RISC Mikrocontroller. AVR-Mikrocontroller sind mit einem Flash-Programmspeicher ausgestattet und in-circuit programmable.

Die typische Befehlsausführungszeit liegt wie bei den PICmicro's bei einem Taktzyklus pro Befehl, ein mit 8MHz getakteter Prozessor bringt daher etwa 8MIPS Geschwindigkeit.

Ein geeigneter Mikrocontroller aus dieser Familie ist der AT90S8535 bzw. der AT90S4433 (Angaben zum 4433 in Klammern). Er bietet folgende Möglichkeiten:

- Befehlssatz mit 118 verschiedenen Befehlen
- 32 Register
- 8 (4) KByte-Programmspeicher, min. 1000mal wiederbeschreibbar
- 512 (128) Byte RAM Datenspeicher
- 512 (256) Byte EEPROM Datenspeicher
- Stromsparmode (Power Down, per Interrupt weckbar)
- ein (zwei) 8Bit- und ein 16Bit-Timer
- 10Bit A/D-Wandler mit 6 (8) Eingängen (Multiplexer)
- Analog-Komparator
- integrierter UART
- dynamischer wachsender Stack im RAM-Speicher

Wie für den PICmicro ist auch für den AVR eine kostenlose Entwicklungsumgebung auf der Webseite des Herstellers verfügbar. [ATMEL]

4.3 Unterschiede zwischen PIC und AVR

Da ich vor dieser Diplomarbeit weder mit dem Atmel- noch dem Microchip-Mikrocontroller Erfahrungen gesammelt hatte, fiel die Entscheidung für einen dieser für das Projekt gleichermaßen geeigneten Controller ohne eine Beeinflussung durch eine eventuelle Gewöhnung an die eine oder andere Architektur, stattdessen fielen mir am Konzept des PICmicro einige Dinge auf, die ich als unnötige Einschränkung empfand.

Da wäre der begrenzte Hardwarestack, dessen Tiefe von 8 Ebenen durchaus, wenn auch selten, bei der Verschachtelung von Unterprogrammen zu einem Problem werden kann. Atmel-Controller verwenden einen Stack dynamischer Grösse, der zu kleiner werdenden Adressen hin wächst. Die Wurzel des Stacks wird nach dem Reset durch das Schreiben der Startadresse in das oder die (abhängig von der Grösse des RAM-Speichers) entsprechende(n) Register festgelegt.

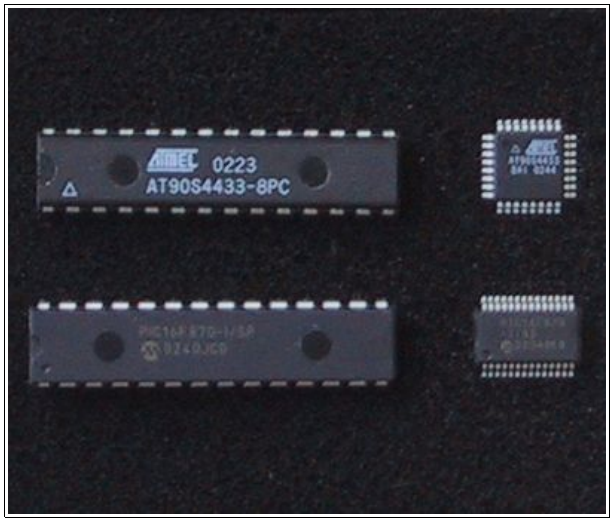

Abbildung 4.1: AT90S4433 (oben) und PIC16F870 (unten), verschiedene Bauformen

Die Hardwarefunktionen wie auch das interne RAM werden beim PICmicro über mehrere ‚Registerbänke' angesprochen, die einen gemeinsamen Adreßraum belegen. Diese Registerbänke werden über Bank-Switching umgeschaltet, d.h. wahlweise in den Adreßraum eingeblendet. Nötig ist dies, da die Befehlswörter je nach Typ nur 12- oder 14Bit breit sind und für die im Befehlswort eingebettete Adresse nur 7Bit übrig bleiben. Daher lässt sich in einem Befehlswort nicht die vollständige Adresse übermitteln, die höherwertigen Bits müssen vor dem Zugriff über Bank-Switching-Bits (RP1, RP0) im STATUS-Register gesetzt bzw. gelöscht werden.

Indirekte Adressierung ist beim PIC nur über ein 8bittiges Adreßregister (FSR) möglich. Bei PICs mit 9Bit Adreßraum ist daher auch an dieser Stelle die Aufteilung des Adreßraumes in Bänke nötig (STATUS, Bit IRP).

Der AVR-Controller unterstützt nur bedingt die direkte Addressierung, bei RAM-Zugriffen möglich, bei Lese- und Schreibzugriffen auf den Flash-Speicher ist nur die indirekte Addressierung über die 16bittigen X,Y oder Z-Register (je zwei Register R26..R31 zusammengefaßt) möglich.

Das Lesen von Konstanten aus dem Programm-Speicher-Flash ist bei PIC-Prozessoren nur über Umwege möglich, hierzu sind längere Befehlsfolgen ähnlich dem Zugriff auf das EEPROM nötig (siehe auch [PIC], Abschnitt 3.5), bei AVR-Controllern kann über die indirekte Adressierung direkt aus dem Flash-Speicher gelesen werden.

Das Konzept der Register ist bei beiden Mikrocontroller-Familien sehr unterschiedlich implementiert: PICmicro-Programmierern stehen ein Akku (hier w, Working Register, genannt) und das direkt addressierbare RAM (Register File Address) zur Verfügung. Arithmetische und logische Befehle, die Daten aus dem RAM verarbeiten, verfügen über ein Flag (0 oder 1), das darüber entscheidet, ob das Ergebnis in das w-Register oder zurück in das RAM geschrieben wird. Ein sehr leistungsfähiges Konzept, wenn auch der Quelltext etwas schwieriger zu lesen ist. Dadurch, dass durch die Unterscheidung mittels des Flags viele Befehle eine Doppelfunktion erfüllen, relativiert sich die geringe Befehlsanzahl des PIC-Prozessors wieder.
Der AVR-Programmierer greift hingegen auf 32 Register (r0..r31) zurück, von denen r0..r15 nur eingeschränkt nutzbar sind. Direkte Manipulation der im RAM befindlichen Daten ist ausgeschlossen, nur Registerinhalte oder Konstanten werden von den logischen und arithmetischen Befehlen verarbeitet.

Bedingte Sprünge schließlich werden beim PIC-Prozessor über einen Vergleichsbefehl *btfsc/btfss Status,X* (Vergleich, ob Bit X in Status-Register gesetzt oder gelöscht, abhängig vom Ergebnis des Vergleichs den nächsten Befehl überspringen) und einem nachfolgenden, unbedingtem Sprungbefehl *goto* ausgeführt, beim Atmel-Prozessor über das schon vom 6502 bekannte Konzept der bedingten Sprungbefehle. Durch solche Techniken kommt der PIC-Prozessor mit sehr viel weniger Befehlen aus als der Atmel-Prozessor, die Programmierung ist dafür meiner Ansicht nach unübersichtlicher (zumindest sehr ungewohnt) und die Code-Ausführung bei gleicher Taktfrequenz langsamer. Der PIC-Prozessor benötigt für die Befehlsfolge

btfsc..., *goto...* im Sprungfall 3 Taktzyklen, der AVR für die Verzweigungsbefehle (z.B. *breq*) im Sprungfall 2 Taktzyklen.

Ich konnte bei der Durchsicht der Datenblätter des AVR- und des PIC-Chips eher eine Verwandschaft des Assembler-Befehlssatzes zwischen AVR und den bisher von mir verwendeten Prozessoren (65xx, 803x/5x, 68HCxx und 680xx) erkennen, was teilweise ausschlaggebend für die Entscheidung für den AVR-Prozessor war.

Bei ersten Experimenten mit der Entwicklungsumgebung AVR Studio 4 fiel mir positiv der integrierte Simulator auf, mit dem sich u.a. Registerinhalte während der simulierten Programmausführung ansehen und ändern lassen. Ganz perfekt ist indes auch diese Software nicht; z.B. führt der Assembler keinen Test durch, ob möglicherweise mehr RAM belegt wird als physikalisch im gewählten Mikrocontroller vorhanden ist. In der folgenden Abbildung sind Warnmeldungen des Assemblers zu sehen, die auf ungerade Mengen an Bytes innerhalb eines Datenfeldes hinweisen, jedoch befindet sich an der angegebenen Position Quelltext, keine .db-Zeile. Ungeachtet solcher (kleineren) Probleme lässt sich mit der IDE sehr gut arbeiten.

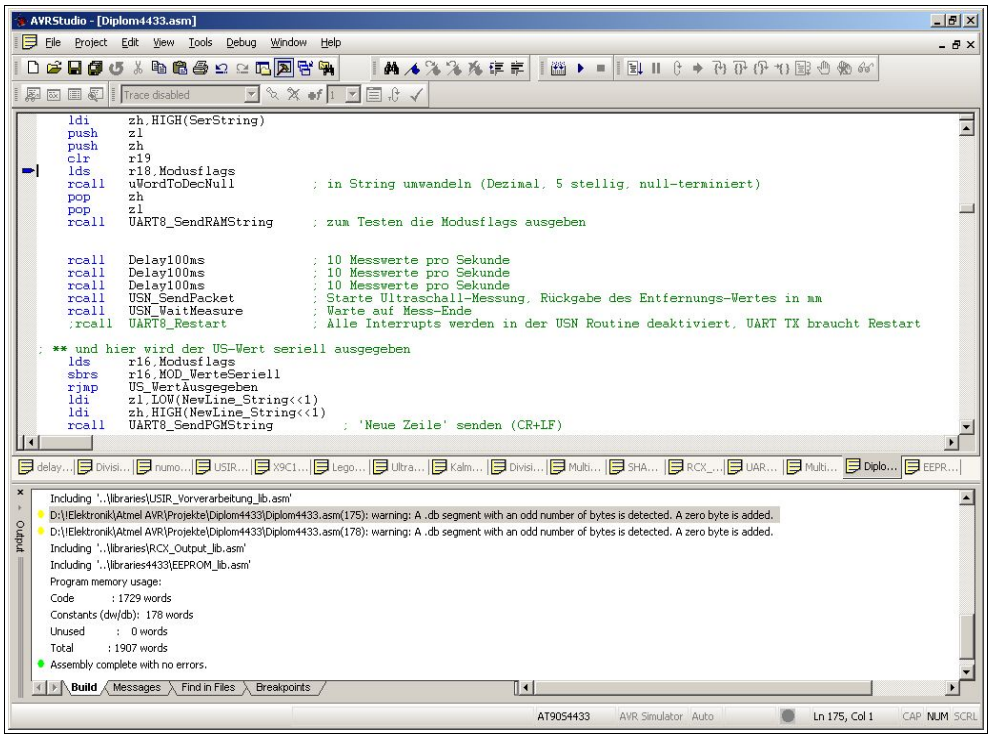

Abbildung 4.2: AVR Studio 4, geöffnetes Projekt

4.4 Entscheidung für den AVR

Ich bin mir sicher, das das hier behandelte Problem genausogut mit einem PIC-Prozessor hätte gelöst werden können - aufgrund der hervorragenden IDE und dem etwas gewohnter wirkenden Assembler-Befehlssatz fiel die Wahl jedoch auf einen Atmel-Mikrocontroller, zunächst auf den AT90S8535, der im Laufe des Designs aufgrund der kleineren Abmessungen durch den in weiten Teilen übereinstimmenden AT90S4433 ersetzt wurde.

Der AT90S4433 verfügt nur über 4KB Flashspeicher, 128 Byte RAM, 256 Byte EEPROM und etwas weniger I/O-Leitungen als der AT90S8535, reicht aber für dieses Projekt aus.

Nachteilig ist das Nichtvorhandensein komplexer arithmetischer Befehle wie Multiplikation und Division, diese müssen durch Software nachgebildet werden. Auch die neueren ATmega-Controller helfen hier nur wenig weiter, da nur 8Bit-Multiplikationsbefehle und keine Divisionsbefehle vorhanden sind.

Im konkreten Anwendungsfall führt der Mikrocontroller die Messungen durch, verarbeitet die Messwerte zu einem möglichst wahrscheinlichen Entfernungswert und gibt den Wert in geeigneter Form an das Zielsystem (Lego RCX-Controller oder serielle Schnittstelle) weiter. Ausserdem kann das Zielsystem über ein einfaches Protokoll verschiedene Verhaltensweisen des Moduls steuern, z.B. verschiedene Entfernungsbereiche, Stromsparmodi, etc.

4.5 Physikalisches Design

4.5.1 Ultraschall-Entfernungsmessung

Die Ultraschallmessung wird mit einem Transducer des Herstellers Polaroid durchgeführt, der auch in Sofortbildkameras dieses Herstellers zum Einsatz kam.

Zunächst wurde vorgesehen, eine ebenfalls von Polaroid gefertigte Elektronik (Polaroid 6500 Ranging Module) zu nutzen, die die gesamte für die Ultraschall-Laufzeit-Messung erforderliche Analog- und einen beträchtlichen Teil der erforderlichen Digital-Elektronik wie

- programmierbarer Verstärker (11 Stufen),
- Takt- und Ultraschall-Signal-Erzeugung,
- Zähler für GAIN-Control (zeitabhängige Verstärkung)

auf einer Platine beinhaltet. Diese Platine kam ursprünglich in AutoFocus-Kameras zum Einsatz. Abbildung 4.3 zeigt den Schaltplan dieses Moduls.

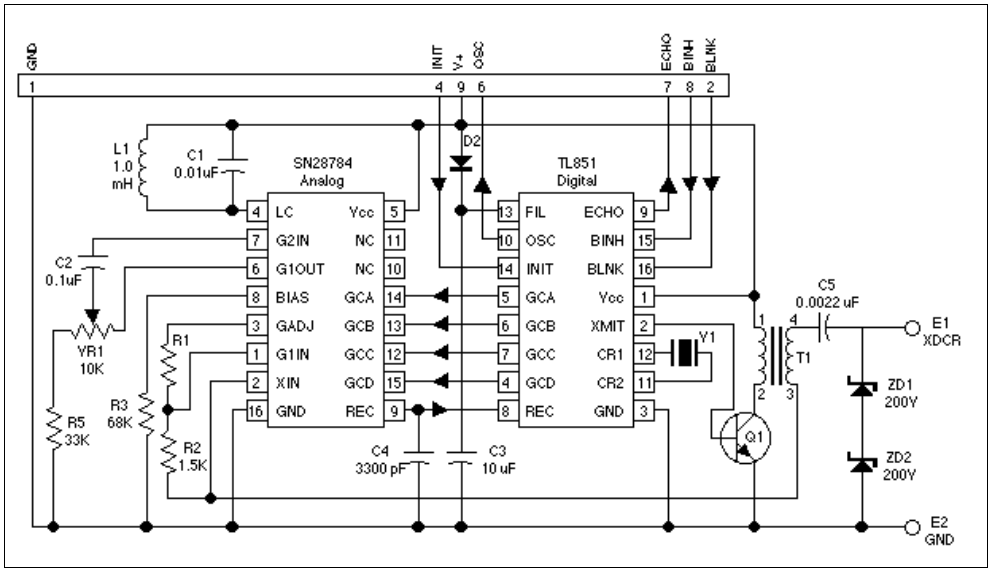

Abbildung 4.3: Schaltplan Polaroid 6500 Series Sonar Ranging Module

Diese Elektronik erzeugt bei Triggerung des Einganges ‚INIT' einen Ultraschall-Burst mit 16 Schwingungen (50kHz) und misst die Dauer bis zum Eintreffen eines von einem Objekt reflektierten Echos. Der Signalverlauf einer Messung ist in Abbildung 4.4 zu sehen. Das Signal 'Internal Blanking' stellt die Echounterdrückung dar, ohne die bereits der auszusendende Burst bzw. dessen Nachschwingen als Echo erkannt werden würde. Diese Echounterdrückung wird in der gewählten Betriebsart (durch BINH, BINL = Low) vom Digital-IC TL851 selbst vorgenommen. Für die Erläuterung der anderen Betriebsarten sei auf das Datenblatt des Sonar Ranging Moduls unter [Acroname] verwiesen.

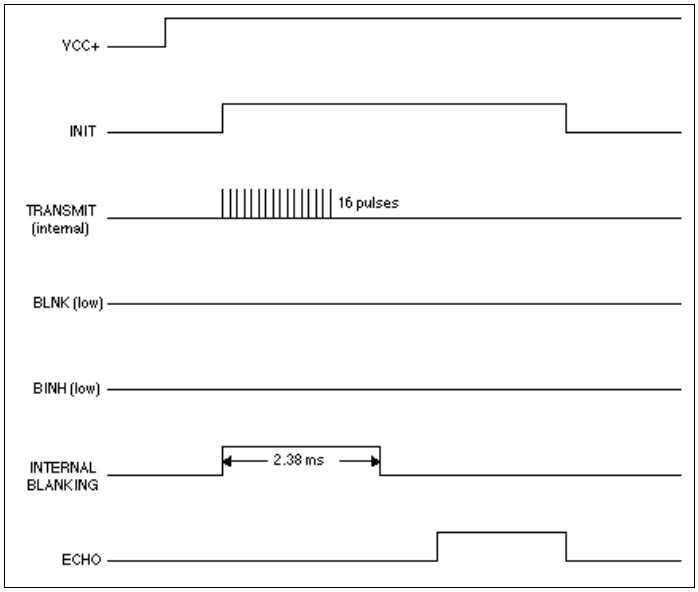

Abbildung 4.4: Signalverlauf einer Messung

Da das Echo-Signal mit zunehmender Entfernung des zu messenden Objekts vom Ultraschall-Transducer immer schwächer wird, wird die Verstärkung des Eingangs-Verstärkers in insgesamt 11 Stufen während der ersten 40 ms nach Aussendung des Bursts erhöht. Der Empfang eines Echos führt zu einer Pegelveränderung der Ausgangsleitung ‚ECHO'.

Weitere Informationen zu diesem für $34 in den USA erhältlichen Modul sind unter [Acroname] zu finden.

Die Funktionalität des Moduls wäre für die Integration in das Distanzmessmodul gut geeignet, da die Platine jedoch nicht in SMD-Technik aufgebaut ist, sondern aus bedrahteten Bauteilen und integrierten Schaltugen im DIL-Gehäuse, ist sie entsprechend gross. Ein Nachbau des Schaltungsteils in SMD-Technik ist auch nicht möglich, da speziell die verwendeten IC's einerseits nur im DIL-Gehäuse verfügbar sind, andererseits ausserhalb der USA nicht beschaffbar zu sein scheinen. Auch die hohe Ruhestromaufnahme der Schaltung von 100 mA, die Verwendung eines u.a. aus einer mechanisch grossen Spule aufgebautem LC-Filter und nicht zuletzt der hohe Preis lässt sie für den geplanten Einsatzzweck nicht geeignet erscheinen.

Aus diesen Gründen habe ich mich entschlossen, dieses zunächst eingeplante Modul durch eine eigene Schaltung zu ersetzen, die

grundlegende Funktionsweise ist mit der des Polaroid-Moduls zu vergleichen.

Der gesamte digitale Schaltungsteil, also Erzeugung des Ultraschall-Bursts und die Verstärkungssteuerung konnte durch den Mikrocontroller und wenige Bauteile vollständig ersetzt werden. Die Funktion des analogen Schaltungsteils wird mit einem Standard-Operationsverstärker, einem digital steuerbaren Potentiometer, einem Transistor und einigen passiven Bauteilen nachgebildet (siehe Abbildung 4.5).

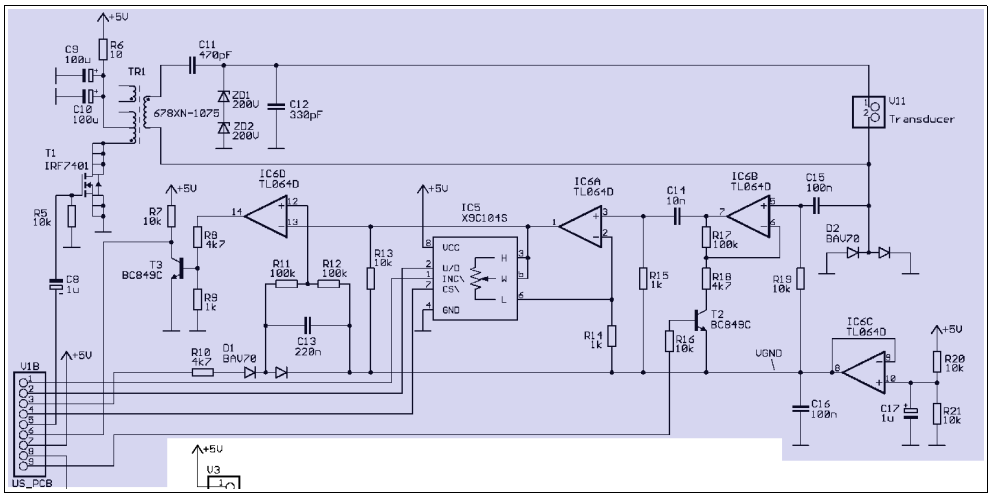

Abbildung 4.5: Schaltungsteil 'Ultraschall-Sensorik'

4.5.2 Ultraschall-Impuls-Erzeugung

Sofern nicht explizit angegeben, beziehen sich Bauteilebezeichnungen im Folgenden auf die im Text eingebetteten Ausschnitte des Schaltplans bzw. auf den im Anhang zu findenden Gesamtschaltplan des Distanzmessmoduls.

An einer Ausgangsleitung des Mikrocontrollers wird ein Schwingungspaket mit 16 Schwingungen, 50kHz, $5V_{SS}$ erzeugt. Ein mit diesem Signal angesteuerter MOSFET (T_1) steuert einen Transformator (TR_1) an, der hieraus ein Schwingungspaket mit $400V_{SS}$ erzeugt.

Die hierfür notwendige Energie wird zwei parallel geschalteten 100µF-Elkos (C_9 und C_{10}) entnommen, die über den Widerstand R_6 mit 5V geladen werden. Hierdurch wird der Stromfluss aus der 5V-Versorgung begrenzt. Ohne $R_6/C_{9,10}$ würde sich der Ultraschall-Burst als starke Schwankung auf der 5V-Versorgung fortsetzen.

Zwei in Reihe geschaltete 200V-Zenerdioden (ZD_1, ZD_2) sorgen dafür, dass die Spitzenspannung des an C_{11} anliegenden Schingungspakets auf $400V_{SS}$ begrenzt wird. Der Ultraschall-Transducers wird sowohl als Sender als auch als Empfänger benutzt.

Als hochspannungserzeugender Transformator wurde zunächst der der Polaroid-Platine verwendet, der jedoch ein Beschaffungsproblem darstellt.
Der Hersteller TOKO bietet problemlos erhältliche Transformatoren an, die für die Hochspannungserzeugung für den Betrieb von Kaltkathodenröhren (CCFL, eingesetzt als Hintergrundbeleuchtung von TFT-Displays) konzipiert wurden. Diese Transformatoren gibt es mit ähnlichen Wicklungsverhältnissen (~1:100), die benötigte Platinenfläche ist etwas grösser, dafür ist der TOKO-Transformator flacher als das Polaroid-Pendant und für die SMD-Montage geeignet.

In den Abbildungen 4.6 und 4.7 ist die Reflektion eines etwa 40 cm entfernt stehenden Hindernisses zu sehen. Gemessen wurde am Eingang des Ultraschallverstärkers (zwischen GND und D_2/C_{15}).

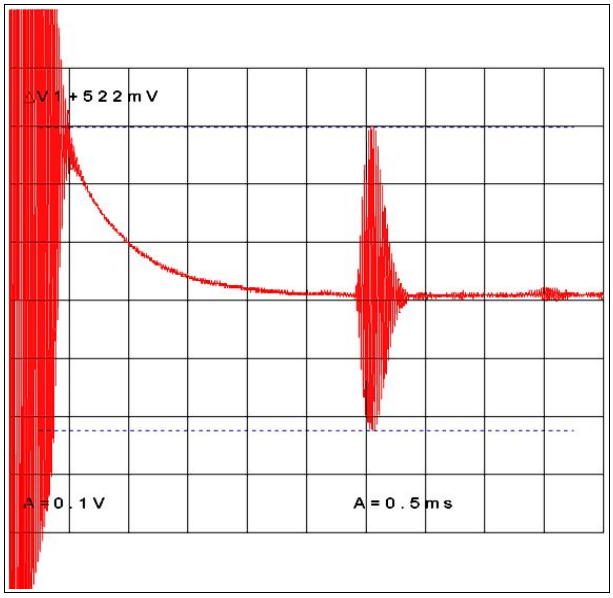

Abbildung 4.6: Reflektion, Polaroid-Transformator

Das Wicklungsverhältnis stimmt zwar mit dem Original Polaroid-Transformator etwa überein, die Induktivität leider nicht; die der Sekundärwicklung ist etwa 5mal (115mH statt 21mH) höher. Daraus ergibt sich leider eine andere Resonanz-Frequenz, zu sehen beim Ausschwingen, also direkt nach der Erzeugung der Ultraschall-Impulse.

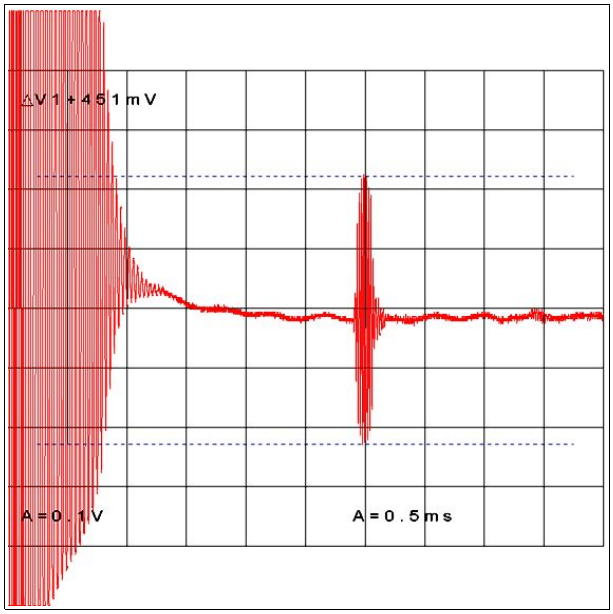

Abbildung 4.7: Reflektion, TOKO-Transformator

$$f_0 = \frac{1}{2 \cdot \pi \cdot \sqrt{L \cdot C_{ges}}} \quad \text{mit L = 21mH und } C_{ges} = 407\text{pF} \quad \rightarrow \quad f_0 = 54\text{kHz}$$

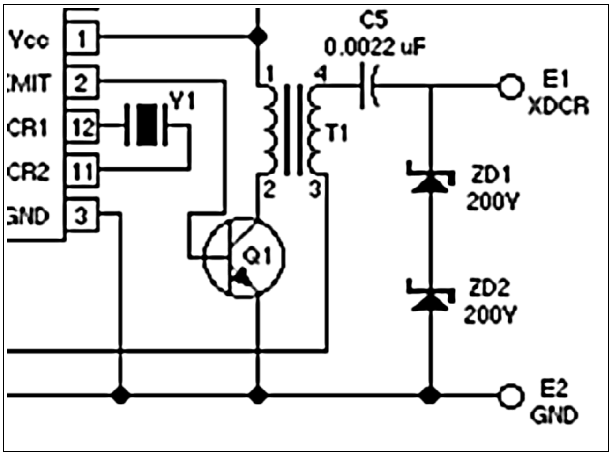

Abbildung 4.8: Ausschnitt Transducer-Treiber des Polaroid Sonar Ranging Moduls

C_{ges} ergibt sich aus der Reihenschaltung von C_5 (Bauteilebezeichnung aus dem Schaltplan des Polaroid-Moduls, siehe Abbildung 4.8) und der Eigenkapazität des Transducers von etwa 500pF.

Mit dem TOKO-Transformator und der hier angegebenen Dimensionierung liegt die Resonanzfrequenz des aus Sekundärwicklung, C_{11}, C_{12} (330pF-Kondensator) und der Kapazität des Ultraschall-Transducers bestehenden Schwingkreises bei etwa 29kHz (gemessen). Da C_{11} mit der Eigenkapazität des Ultraschall-Transducers einen kapazitiven Spannungsteiler ergibt, kann er nicht auf 110pF (ergibt sich aus obiger Formel mit der Induktivität des TOKO-Transformators) verkleinert werden. Hierbei ergäbe sich zwar eine Resonanzfrequenz von etwa 50kHz, jedoch würde die Spannung über dem Ultraschall-Transducer zusammenbrechen.

Um mit dem TOKO-Transformator (678XN-1075) gute Ergebnisse zu erzielen, wurde parallel zu den beiden Zenerdioden ein Kondensator C_{12} von 330pF (experimentell ermittelt) geschaltet. Der Reihenkondensator C_{11} wurde durch 470pF ersetzt.

In den beiden Oszillogrammen (Abbildung 4.6, 4.7) ist zu sehen, dass der Burst bei Verwendung des TOKO-Transformators etwa doppelt so lange nachschwingt (in der englischsprachigen Literatur auch als Ringing bezeichnet) als bei Verwendung des Polaroid-Transformators. Hierdurch verschlechtert sich der minimal messbare Mindestabstand zum Messobjekt (wird grösser), da während des Nachschwingens kein Echo detektiert werden kann. Ebenso hat die Amplitude des reflektierten Signals abgenommen. Das empfangene Echo ist bei Verwendung des TOKO-Transformators schärfer, da es nicht zu Nachschwingungen aufgrund der Resonanzfrequenz führt.
In der Praxis ist trotzdem eine gute Empfindlichkeit zu verzeichnen, die Verschlechterung im Nahbereich um einige cm wird durch den Sharp-Infrarot-Sensor gut ausgeglichen.

4.5.3 Ultraschall-Echo-Detektion

Das Empfangssignal wird über zwei antiparallel geschaltete Dioden auf $0{,}6V_{SS}$ begrenzt einem Operationsverstärker (IC_{6B}) zugeführt, dessen Verstärkung über einen im Rückkopplungszweig befindlichen Transistor (T_2) zwischen 1fach und 20fach umgeschaltet werden kann. Ein nachfolgender Kondensator (C_{14}) dient in Verbindung mit R_{15} als Hochpassfilter und entfernt eventuelle Gleichspannungsanteile aus dem Signal, die u.a. durch Offset-Fehler entstanden sein können.

Es folgt eine weitere Verstärkungsstufe, deren Verstärkung quasilinear zwischen etwa 2fach und 100fach mittels eines digital steuerbaren Potentiometers (IC_5) gesteuert werden kann.

Der Mikrocontroller erhöht im Verlauf der Messung kontinuierlich die Empfindlichkeit des Verstärkers. Zunächst wird nach Ausschwingen des Bursts die Empfindlichkeit des gesamten Verstärkers von 2x auf etwa 40x erhöht, indem der Transistor T_2 vom Mikrocontroller angesteuert wird. Im weiteren Verlauf wird die Verstärkung durch stufenweises Erhöhen des Widerstands des Digitalpotis IC_5 erhöht. Die theoretisch maximal mögliche Verstärkung beträgt V=2000. Im Kapitel 4.7.1 (Ultraschallmessung) werden die zeitlichen Zusammenhänge der Verstärkungserhöhung erklärt.

Der aus den Widerständen R_{20} und R_{21} aufgebaute Spannungsteiler mit einem nachfolgenden Impedanzwandler (IC_{6C}) sorgt für eine virtuelle Masse dieses Schaltungsteils, deren Potential etwa 2,5V über GND liegt.
Auf diese Weise kann auf eine symmetrische Spannungsversorgung des Operationsverstärkers verzichtet werden. Die so erzeugte, über C_{17} und C_{16} geglättete, virtuelle Masse ist alles andere als sauber und schwankt stark gegenüber GND, dies hat jedoch keine Auswirkungen auf die Funktion der Schaltung, da sowohl das verstärkte Signal als auch das Vergleichssignal an C_{13} diese Masse als Bezugspunkt verwenden.

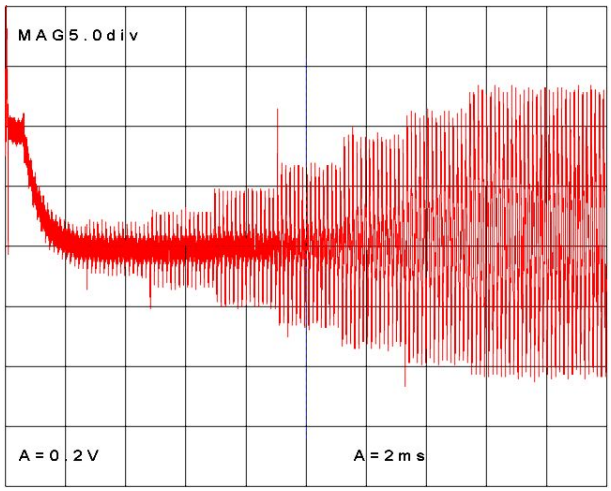

Abbildung 4.9: sinusförmiges 50kHz Signal, GAIN-Erhöhung

Abbildung 4.9 zeigt die Verstärkung eines sinusförmigen Eingangssignals mit kontinuierlicher Erhöhung des Verstärkungsfaktors über die ersten 16 ms. Gemessen wurde an Pin1, IC_{6A}, am Ausgang der durch das Digitalpoti geregelten Verstärkerstufe.

Das Ausgangssignal dieses Verstärkers wird einem Komparator (IC_{6D}) zugeführt, der es mit einer an einem sich entladenden Kondensator (C_{13}) abfallenden Spannung (genauer: der Hälfte der an C_{13} abfallenden Spannung, Spannungsteiler R_{11}/R_{12}) vergleicht, die im Laufe der Messung auf das Potential der virtuellen Masse abfällt, was die Schaltschwelle (Treshold) des Komparators kontinuierlich sinken und somit die Echo-Detektions-Empfindlichkeit des Empfängers stetig steigen lässt.

Der Kondensator C_{13} wird vom Mikrocontroller zu Beginn der Messung auf etwa 0,6V, begrenzt durch die im Schaltbild rechts zu sehende Hälfte der Doppeldiode D_1, aufgeladen. Die andere Hälfte der Doppeldiode dient dazu, eine Entladung des Kondensators über den Ausgang des Mikrocontrollers zu vermeiden. Die Zeitkonstante des RC-Glieds aus C_{13} und R_{11}/R_{12} beträgt etwa 40 ms.

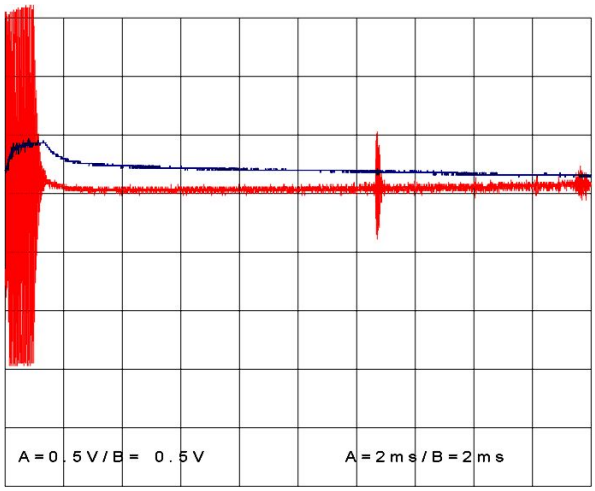

Abbildung 4.10: Signale am Komparator IC_{6D}

In Abbildung 4.10 sind die beiden am Komparator (IC_{6D}) anliegenden Signale zu sehen, in etwa 2m Entfernung befindet sich ein Hindernis (Bürostuhl):

Der Transistor T_3 konvertiert den Ausgangspegel des Komparators in logikgerechte TTL-Pegel. Ein empfangenes Echo-Signal führt zu LOW-Pegel am Kollektor des Transistors T_3 (siehe Abbildung 4.11, links der erzeugte Ultraschall-Burst, in der Schirmmitte ein Echo von einem etwa 80 cm entfernten Objekt, rechts das von einer 130 cm entfernten Wand zurückgeworfene Signal).

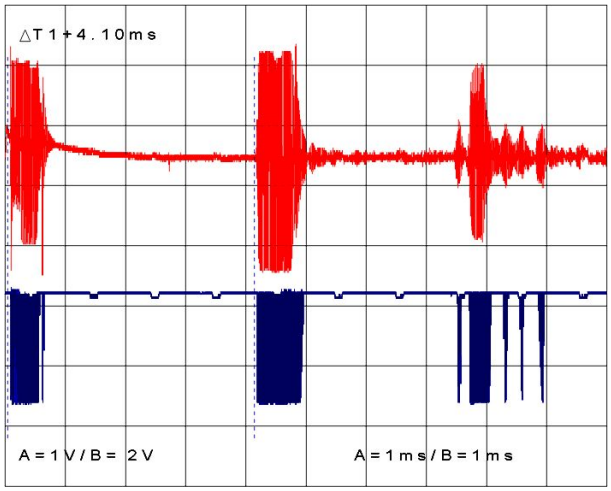

Abbildung 4.11: Ultraschallsignal, Kanal1 (rot): Pin7 IC6b, Kanal2 (blau): Ausgang Kollektor T3

Im Folgenden wird betrachtet, inwieweit das von entfernten Objekten reflektierte Echo bei Änderung der Impulsanzahl des Bursts ebenfalls eine Änderung erfährt. Abbildung 4.12 stellt den Burst, das nachfolgende Ausschwingen des Schwingkreises Transformator/Transducer sowie das reflektierte Echo eines etwa 25 cm entfernten Objektes dar. Beim blau dargestellten Signalzug besteht das Burstpaket aus nur einer Schwingung, beim rot dargestellten aus acht Schwingungen mit einer Frequenz von 54kHz. Wie zu erwarten war, ist die Reflektion um so ausgeprägter, je länger der Burst ist. Gleichzeitig verlängert sich mit Verlängerung des Bursts auch der Totbereich, in dem keine Entfernungsmessung stattfinden kann, wenn dieser auch deutlich stärker vom Ausschwingen des Gesamtsystems Transformator, Kapazitäten C_{11} und C_{12}, Transducer abhängt.

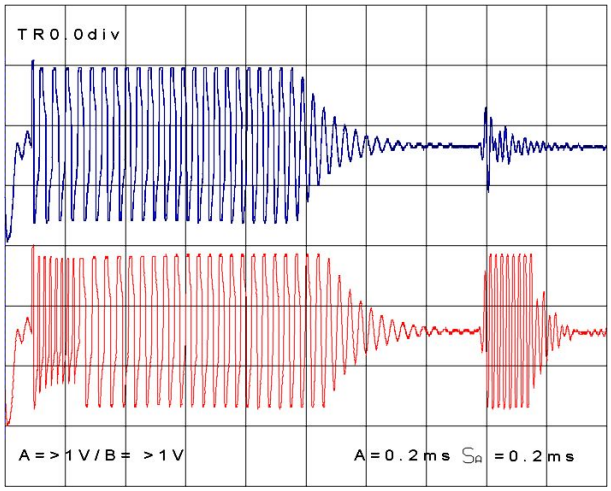

Abbildung 4.12: Burstdauer 1 Impuls (blau), 8 Impulse (rot)

4.5.4 Infrarot-Entfernungsmessung

Für den Infrarotteil ist nur wenig Beschaltung erforderlich: Ein Infrarot-Sensor vom Typ Sharp GP2D12 kommt zum Einsatz. Dieser liefert abhängig von der Entfernung zum Objekt eine Ausgangsspannung von etwa 1V bis 3V. Um diese Spannung in einen weiterverarbeitbaren Digital-Wert zu konvertieren, wird ein Kanal des im Mikrocontroller vorhandenen sechskanaligen Analog-Digital-Wandlers benutzt. Der A/D-Wandler benötigt eine möglichst saubere Versorgungsspannung, die über den Filter aus L_1 und C_6 (siehe Abbildung 4.13) aus der 5V-Versorgung gewonnen wird. Das analoge Eingangssignal wird in einen digitalen Wert mit 10Bit-Auflösung gewandelt.

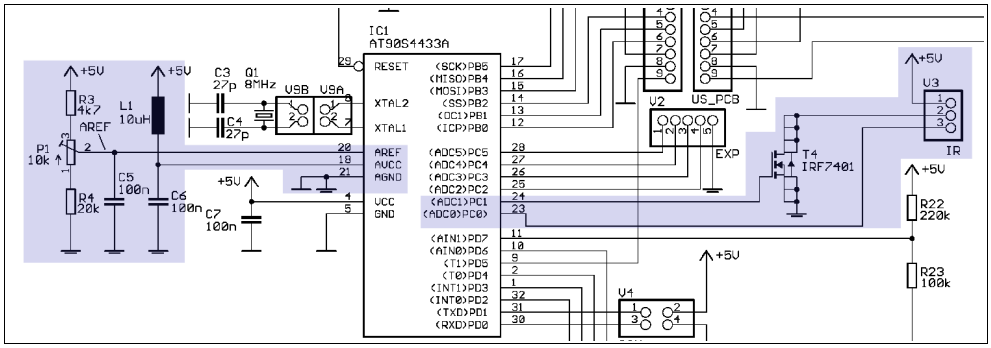

Abbildung 4.13: Schaltungsteil 'Infrarot-Sensorik'

Die untere Grenze der quantifizierbaren Eingangsspannung wird durch AGND festgelegt, die obere Grenze durch die Referenzspannung am Pin AREF des Mikrocontrollers. AREF muss also bei knapp über 3V liegen, um

einerseits den möglichen Wertebereich fast vollständig ausnutzen zu können, andererseits um Übersteuerungen des AD-Wandlers zu verhindern.

Die Stromaufnahme des Infrarot-Sensors liegt bei etwa 27,5mA bei 5,0V Versorgungsspannung. Um Strom sparen zu können, wird der IR-Sensor nicht dauerhaft mit der Versorgungsspannung von 5V versorgt, sondern von einem MOSFET (T_4) eingeschaltet.

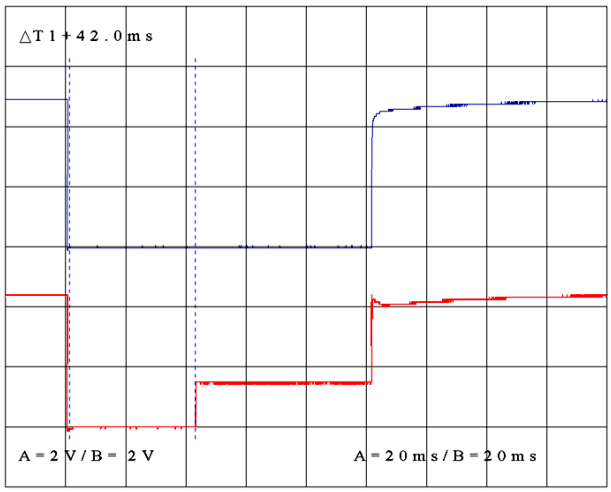

Abbildung 4.14: Kanal1: Geschalteter Masse-Anschluss des IR-Sensors, Kanal2: Analoges Ausgangssignal

Nach dem Einschalten der Versorgungsspannung (da ein n-Kanal-MOSFET mit $U_{GS} > 4V$ verwendet wird, wird hier nicht der +5V-, sondern der Masse-Anschluss geschaltet) benötigt der Sensor einige Zeit (hier 42 ms, Abbildung 4.14), bis der Ausgang stabile Messwerte liefert. Diese Verzögerung ist bei der Software-Implementierung zu beachten.

Die Stromaufnahme des SHARP-Infrarot-Sensors besteht zeitlich betrachtet aus einem etwa 0,5 ms währenden Muster aus kurzen Impulsen von bis zu 55 mA, die sich im Abstand einer Millisekunde wiederholen.

Wenn pro Sekunde zwei Messungen ausreichend sind und pro Messung eine Einschaltdauer von etwa 75 ms angenommen wird, sinkt der mittlere Stromverbrauch des Sensors durch Einsatz des MOSFETs auf etwa 4,1mA.

4.5.5 Berührungserkennung

Als letzte Instanz dienen zwei an der Front des Roboters angebrachte Taster der Berührungserkennung, also der Erkennung der Distanz d=0 zum

Hindernis. Sie werden, wie in Abbildung 4.15 dargestellt, an den mit EXP bezeichneten Erweiterungsanschluss des Distanzmesssystems angeschlossen.
Es wurden sehr leichtgängige, umschaltende Taster gewählt, von denen hier nur der Öffner verwendet wird. Die Eingangsleitungen des Mikrocontrollers verfügen über zuschaltbare Pullup-Widerstände, die in diesem Fall aktiviert sind. Pin EXP5 liegt auf GND. Im Ruhezustand liegt Low-Pegel auf den Leitungen EXP1 und EXP2. Wird einer der Taster gedrückt, so springt der entsprechende Eingang EXP1 oder EXP2 auf High-Pegel.

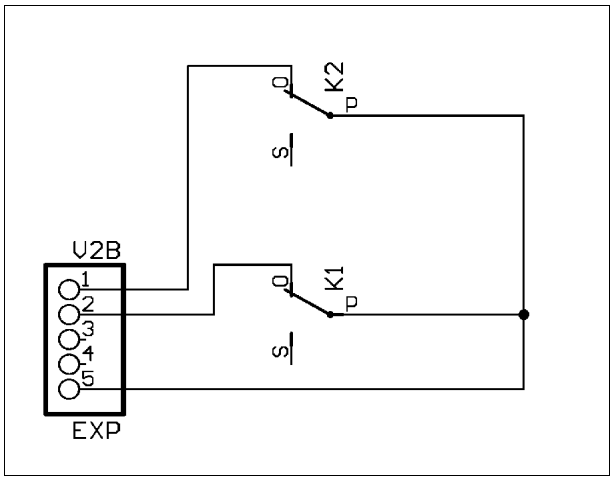

Abbildung 4.15: Taster zur Berührungserkennung

4.5.6 Datenübermittlung zum Zielsystem

Das Ergebnis einer Entfernungsmessung kann in digitaler Form am seriellen Anschluss des Mikrocontrollers gelesen werden. Das Zielsystem, das Lego RCX-Modul, hat aber keinen seriellen Port, verfügt jedoch über analoge Eingänge.

Daher wird ein Analog-Wert benötigt, der entweder über einen Timer mittels Pulsbreitenmodulation (PWM) und ein nachfolgendes RC-Glied erzeugt werden kann (billigste Lösung) oder wie hier von einem Digital-Analog-Wandler, der einen analogen Spannungswert höherer Genauigkeit ausgeben kann.

Beide Lösungen sind praktikabel und im Rahmen der hier erforderten Genauigkeit ausreichend, aus Gründen der Erweiterbarkeit um genauere Sensoren und da der verwendete Mikrocontroller nur über zwei Timer

verfügt, die beide bereits anderweitig genutzt werden, wird hier der Einsatz eines D/A-Wandlers weiter verfolgt.

Im ersten Ansatz wurde ein schon etwas länger auf dem Markt verfügbarer Lowcost-Wandler eingeplant, der ursprünglich für Audio-Anwendungen wie tragbare CD-Player konzipiert wurde: Der TDA1543 von Philips ist ein zweikanaliger 16Bit-D/A-Wandler, der mit Abtastfrequenzen von 50kHz bis hinunter zu statischem Betrieb arbeitet. Er wird über einen I2C-Bus mit dem Mikrocontroller verbunden.

Die endgültige Wahl fiel jedoch auf einen 16Bit-D/A-Wandler des Herstellers MAXIM, da sich der TDA1543 als zu stromhungrig erwies (>50mA). Der MAX5541 benötigt zur Stromversorgung typisch 0,3mA und ist daher für mobile, akku- oder batteriebetriebene Geräte die bessere Wahl.

In Abbildung 4.16 ist das serielle Protokoll zu sehen, in dem der DAC die 16bittigen Datenworte erwartet. Sobald /CS Low-Potential annimmt, erwartet der DAC auf jeder steigenden Flanke von SCLK ein stabil anliegendes Datenbit auf der DIN-Leitung. Das erste Bit ist das höchstwertigste (Most Significant Bit, MSB). Nach Übertragungs des LSB's wird /CS zurückgenommen, daraufhin wird das übertragene Datenwort in das Ausgangslatch übernommen, der analoge Ausgangspegel nimmt den neuen Wert an.

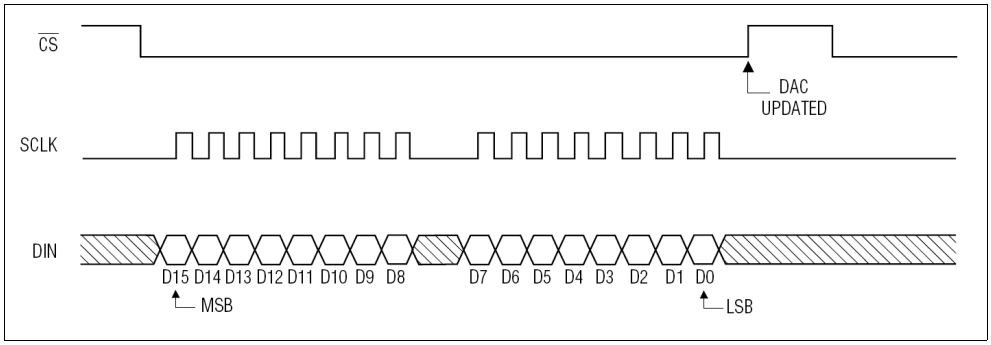

Abbildung 4.16: Serielles Protokoll, MAX5541

Leider benötigt der D/A-Wandler eine extern zugeführte Referenzspannung, die noch dazu aufgrund des verwendeten Wandler-Konzepts (R2R-Ladder, starke Schwankung des Innenwiderstands REF->AGND abhängig von den Wertänderungen am digitalen Eingang) sehr niederohmig sein muss.

Diese Referenzspannung wird von VR_1 (TL431C) erzeugt, einer sehr rauscharmen und niederohmigen 2,5V-Referenzspannungsquelle. Abbildung 4.19 zeigt diesen Schaltungsteil.

Das Datenblatt des D/A-Wandlers ist unter [MAX5541] zu finden.

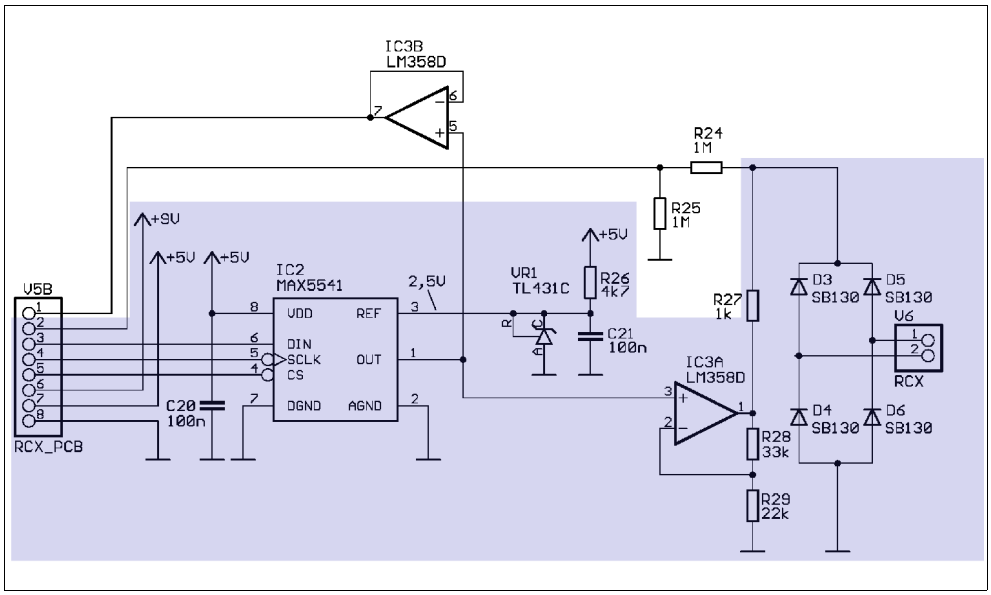

Abbildung 4.17: Schaltungsteil 'D/A-Wandler'

Der Spannungshub des Ausgangssignals reicht für eine Vollaussteuerung des im RCX-Modul befindlichen A/D-Wandlers nicht aus, daher wird es um den Faktor 2,5 verstärkt.

Die Schottkydioden D_3 - D_6 sind erforderlich, da ein RCX-Anschluss als zweiadriger, elektronisch verpolungssicherer Anschluss konzipiert wurde, der keine als solche ausgezeichneten Masse- und Signal-Leitungen hat.
Das Diodennetzwerk stellt eine Verbindung der Masse des RCX-Moduls mit der Masse des Distanzmessmoduls sicher und sorgt für einen Stromfluss vom A/D-Wandler-System des RCX-Moduls zum analogen Ausgang des Distanzmessmoduls: Über eine Spannungsquelle (Pull-Up-Widerstand) im Inneren des RCX-Moduls fliesst ein Strom durch den externen Sensor, der als Analog-Eingangswert gemessen wird.

4.5.7 Bidirektionale Kommunikation über RCX-Sensor-Port

Das RCX-Modul hat die Möglichkeit, sog. aktive oder passive Sensoren zu betreiben. Der Terminus 'Aktiv/Passiv' wird hier in einem anderen Zusammenhang, als in Kapitel 2.2 beschrieben, benutzt: Aktive Sensoren sind hier solche, die vom RCX-Modul mit Strom versorgt werden (müssen).
Hierzu teilt das RCX-Modul die Zeit in Zeitscheiben ein, während derer abwechselnd eine konstante Spannung am RCX-Anschluss angelegt wird bzw. ein analoger Wert gemessen wird.

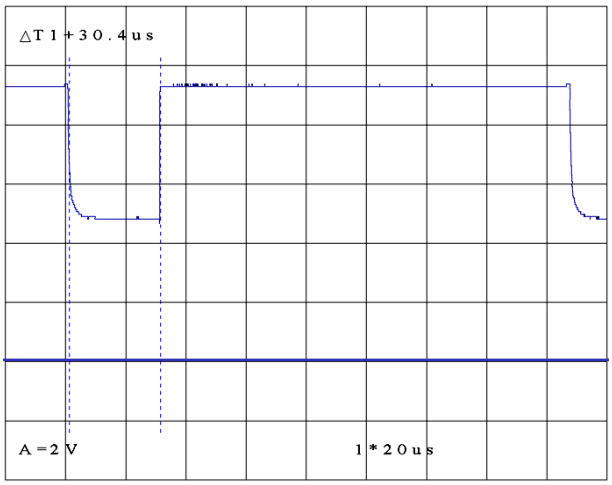

Abbildung 4.18: Signalverlauf am RCX-Eingang, Aktiver Sensor

Die horizontale blaue Linie in Abbildung 4.18 markiert 0V. Zwischen den Zeitmarkern ist das anliegende Messsignal des Sensors zu sehen. Sowohl passive (Mess-) als auch aktive (Stromversorgungs-) Phase sind starken zeitlichen Schwankungen unterworfen, gemessen wird jedoch mindestens über einen Zeitraum von etwa 30µs.

Über die Umschaltung Passiv/Aktiv lässt sich als Nebeneffekt ein einfacher Rückkanal zur Kommunikation mit dem Distanzmessmodul schaffen: Vorhandensein der Versorgungsspannung wird als ‚logisch 1' gewertet, Nichtvorhandensein als ‚logisch 0'. Da das RCX-Modul zur AD-Wandlung eine über einen Widerstand abfallende Spannung anlegt, gilt diese (~5V) als ‚logisch 0'. Im aktiven Modus steigt die Ausgangsspannung auf etwa 7V an, was als ‚logisch 1' interpretiert wird.

Wie in Kapitel 4.8 (Software, RCX-Programmierung) gezeigt wird, gibt es eine Möglichkeit, den Sensoreingang über einen beliebigen Zeitraum ohne die Unterbrechung durch einen Samplingvorgang vom RCX mit Strom versorgen zu lassen.

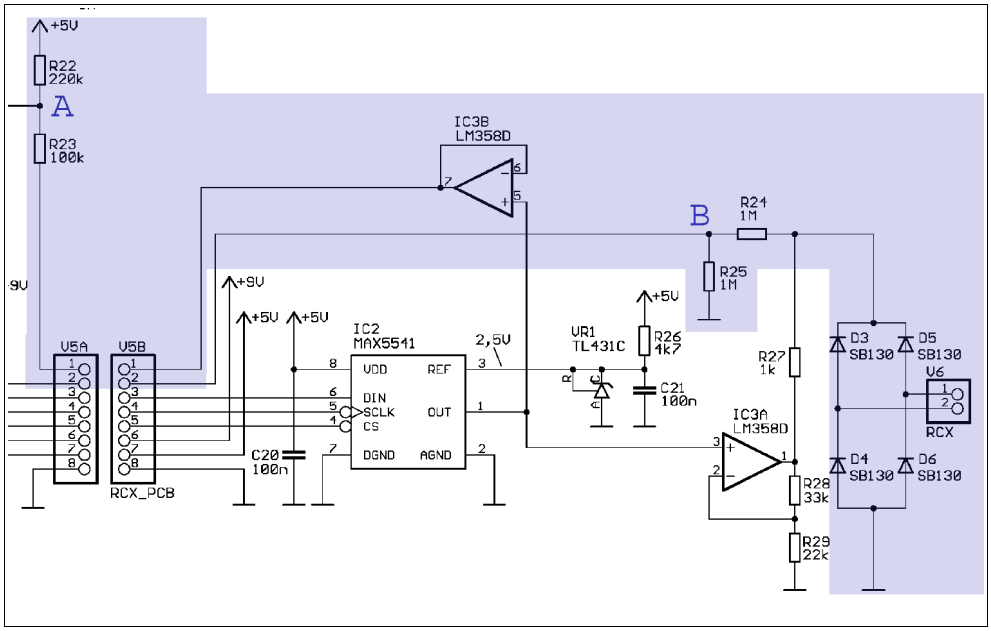

Abbildung 4.19: Schaltungsteil 'RCX-Kommunikation'

Wie in Abbildung 4.19 zu sehen, wird das RCX-Signal über den Spannungsteiler aus R_{24}, R_{25} auf die Hälfte der ursprünglichen Amplitude geteilt (als Signal 'B' bezeichnet).
Der im Mikrocontroller vorhandene Analog-Komparator vergleicht dieses Signal nun mit einer Referenzspannung, gewonnen aus dem Spannungsteiler R_{22}/R_{23} (Signal 'A').
Da das vom RCX empfangene Signal stark vom eigenen übermittelten Messwert an R_{27} beeinflusst wird, wird das Ausgangssignal des D/A-Wandlers IC2 in die Erzeugung der Referenzspannung mit einbezogen. Hierzu wird der Ausgang des D/A-Wandlers über einen Impedanzwandler (IC_{3B}) dem Spannungsteiler aus R_{22}/R_{23} zugeführt. Die Referenzspannung am Pin AIN1 des Mikrocontrollers liegt dadurch immer etwa in der Mitte zwischen analogem Ausgangssignal des Distanzmessmoduls und erzielbarem High-Pegel durch den RCX.

Über diese (unidirektionale) Kommunikation kann der RCX das Verhalten des Distanzmessmoduls beeinflussen. Näheres wird im Kapitel 4.7 (Software, Mikrocontroller-Programmierung) beschrieben.

4.5.8 Stromversorgung

Ein testweise mit verschiedenen Widerständen von 470Ω bis hinunter zu 150Ω belasteter, auf 'Aktiv' geschalteter RCX-Sensoreingang zeigte, dass

der Stromfluss zur Versorgung eines Sensors durch eine Konstantstromquelle auf etwa 14mA begrenzt wird.

Aufgrund der Erfahrungen, die mit dem hohen Strombedarf der ursprünglich eingeplanten Polaroid-Platine und dem Infrarot-Sensor gesammelt wurden, und um eine Kommunikation zwischen RCX und Distanzmesssystem in der beschriebenen Weise zu ermöglichen, bezieht die Schaltung die notwendige Energie aus einer 9V-Blockbatterie oder einem 9V-Akkumulator.

Die Spannungsstabilisierung wurde in den ersten Labormustern mit einem herkömmlichen Linearregler vom Typ 78S05 realisiert.

Dieser wurde im folgenden Design durch einen MAX8881 (IC_4, Abbildung 4.20) ersetzt, der gegenüber dem 78x05 einige Vorteile bietet:

Es handelt sich um einen LowDrop-Regler, der schon mit Eingangsspannungen, die geringfügig über der Ausgangsspannung liegen, stabil arbeitet. Ein 78x05 benötigt mindestens 7,5V um die Ausgangsspannung von 5V zu erzeugen. Dieser Vorteil ist in dieser Anwendung jedoch nicht relevant, da die Schaltung mit 9V-Versorgungsspannung arbeitet: Der Ausgangsverstärker für das RCX-Interface benötigt eine Spannung von mindestens 7V.

Hier nutzbare Vorteile sind jedoch seine geringe Grösse (SOT23-6), sein geringer Eigenstromverbrauch von nur 3,5µA, seine umfangreichen Schutzschaltungen (Schutz gegen Verpolung, Schutz gegen Kurzschluss (dauerkurzschlussfest), Temperatur-Überwachung) und die Erzeugung eines Power-OK-Signals (POK).

Der POK-Ausgang lässt sich direkt mit dem Reset-Eingang des Mikrocontrollers verbinden. Bis sich die Spannung an den ausgangsseitigen Elektrolytkondensatoren auf 90% des Nominalwertes von 5V aufgebaut hat, liegt das Potential dieser Leitung auf Low-Pegel.

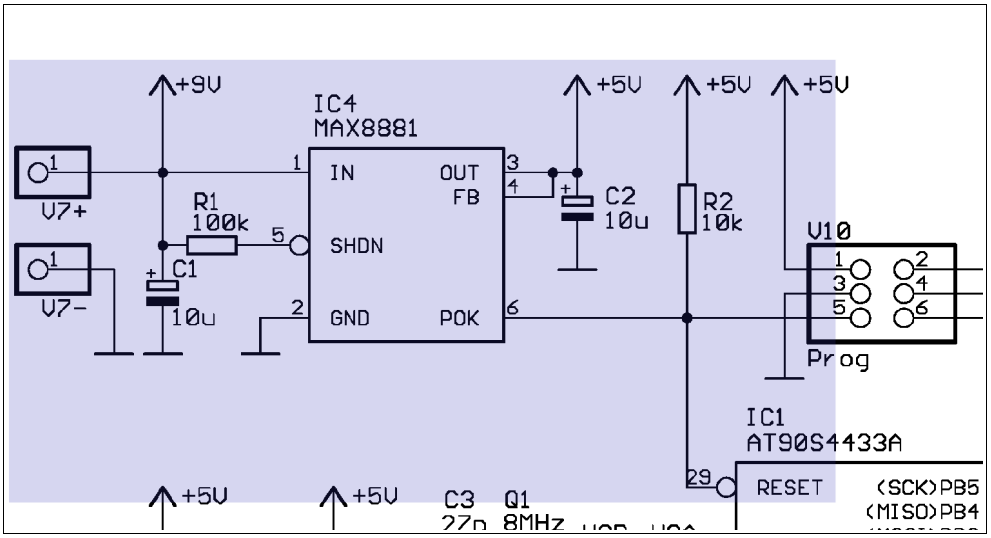

Abbildung 4.20: Schaltungsteil 'Stromversorgung und Reset'

Der Versorgungsspannungsbereich des AT90S4433 ist mit 4,0V – 6,0V spezifiziert, ein bei Erreichen von 4,5V ausklingendes Reset-Signal führt somit zu stabilen Startbedingungen.

Da es sich um einen Open-Source-Ausgang handelt, ist noch ein Pullup-Widerstand erforderlich, um einen High-Pegel zu erzeugen.

Die Shutdown-Leitung zum Abschalten des Reglers (V_{OUT} = 0V) wird hier nicht genutzt.

Ein Datenblatt des Spannungsreglers findet sich auf der Website des Herstellers [MAX8881].

4.6 Aufbau

4.6.1 Labormuster auf Steckplatinen

Beim Aufbau des Prototyps wurden in erster Linie bedrahtete Bauelemente verwendet, nur in Fällen, in denen eine Beschaffung bedrahteter Bauelemente schwer möglich war oder diese einfach nicht existierten, wurden SMD-Bauteile verwendet. Zu diesem Zweck wurden aus Lochrasterplatinen Adapter hergestellt. Dies betrifft den D/A-Wandler, die MOSFETs und den Spannungsregler.

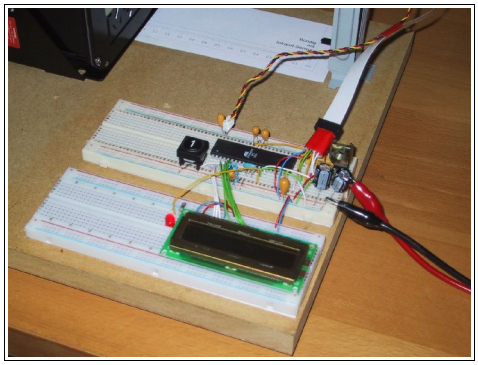

Abbildung 4.21: erstes Labormuster mit AT90S8535, LCD 1x16

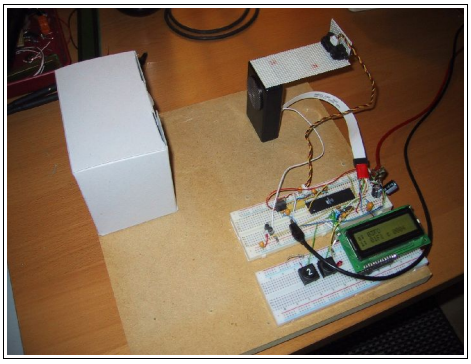

Abbildung 4.22: zweites Labormuster mit AT90S8535, LCD 2x16

4.6.2 Erste Prototypen

Im ersten Entwurf wurde ein Mikrocontroller vom Typ Atmel AT90S8535 verwendet, der in seinen Möglichkeiten mit Ausnahme der Grösse des internen RAM-Speichers, der Grösse des EEPROMs und der Anzahl der I/O-Leitungen weitgehend mit dem AT90S4433 identisch ist.

Zur Ausgabe von Text und Werten kam zunächst ein alphanumerisches LC-Display, zunächst einzeilig (siehe Abbildung 4.21), im nächsten Entwurf zweizeilig (Abbildung 4.22), zum Einsatz. Später wurde das LC-Display durch ein am seriellen Port angeschlossenes Terminal (Windows98-PC mit der Software Hyperterm) ersetzt.

Das im Bild zu sehende Flachbandkabel diente zum Anschluss der Polaroid-Ultraschall-Elektronik, die mittlerweile wie beschrieben durch einen eigenen Schaltungsentwurf ersetzt wurde. Auf Abbildung 4.22, links oberhalb des Mikrocontrollers, ist auch eine eigene Spannungsstabilisierung für den A/D-Wandler sowie der Spindeltrimmer zur Einstellung der Referenzspannung zu sehen.

4.6.3 Labormuster auf Lochrasterplatine

Um den endgültigen Schaltungsentwurf vor der Platinenfertigung zuverlässig testen zu können, wurde ein Labormuster auf einer Lochrasterplatine aufgebaut. Die verwendeten SMD-IC's wurden hierzu auf kleine Adapterplatinen gelötet. Die Bauteile wurden auf der Rückseite der Platine wenn möglich direkt (Lötzinnbrücken) oder mit Kupferlackdraht miteinander verbunden. In der Abbildung 4.23 sind die einzelnen Schaltungsteile zu erkennen:

- Links oben die Ultraschall-Impuls- und Hochspannungserzeugung, der MOSFET wird verdeckt durch einen Tantal-Elektrolyt-Kondensator.
- Darunter der Ultraschall-Verstärker mit dem digital steuerbaren Potentiometer (X9C104).
- In der Mitte der Microcontroller mit Takterzeugung.
- Darüber der auf eine Adapterplatine gelötete Spannungsregler.
- Auf der rechten Seite der D/A-Wandler und der für die Anpassung an das RCX-System nötige Operationsverstärker (hier LM324, ersetzt durch LM358D).
- Die schwebende Platine auf der rechten Seite trägt die Schottky-Dioden für das RCX-Interface, die die auf der Hauptplatine befindlichen, zuerst eingesetzten 1N4148-Dioden wegen des geringeren Spannungsabfalls ersetzen.
- Die vier Schrauben halten das auf der Lötseite angebrachte Gehäuse mit dem Ultraschall-Transducer und dem Infrarot-Sensor.

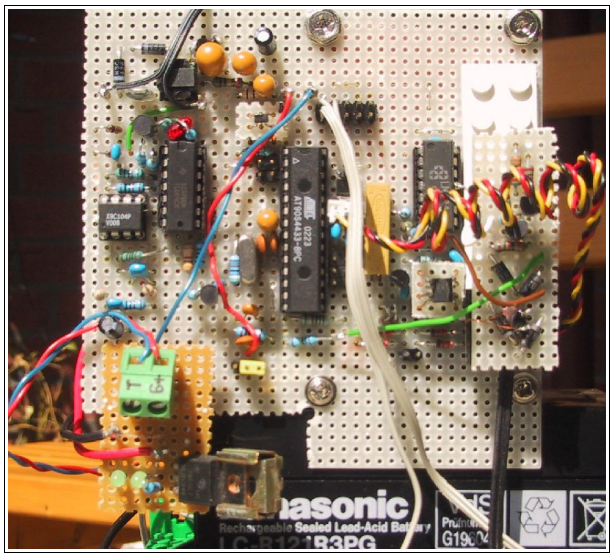

Abbildung 4.23: Labormuster (Lochrasterplatine)

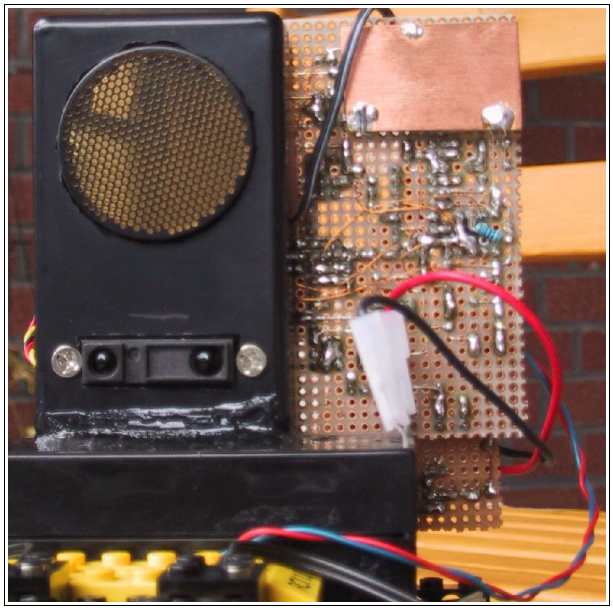

Abbildung 4.24: Labormuster (Platinen-Rückseite, Sensoren)

4.6.4 Layout

Schaltplan und Platinenlayout wurden mit dem Programm EAGLE (**E**asily **A**pplicable **G**raphical **L**ayout **E**ditor) in der Version 4.09r2 erstellt.

Für den nichtkommerziellen Einsatz ist EAGLE in der Light-Version (mit der Beschränkung auf max. 100 mm x 80 mm-Platinenfläche, 2 Layer und nur ein Schaltplanblatt) kostenlos und kann von der Webseite des Herstellers CadSoft heruntergeladen werden [EAGLE].

Mit diesem CAD-Programm ist die Erstellung von Schaltplänen und Platinenlayouts möglich. In den mitgelieferten Bauteil-Bibliotheken sind die meisten Bauteile bereits vorhanden, neue Bibliotheken lassen sich mit einem Editor recht komfortabel erstellen. Die Generierung des Platinenlayouts direkt aus dem Schaltplan ist möglich, gesonderte Netzlisten müssen nicht erstellt werden. Die Bauteile, die manuell auf der Platinenfläche positioniert werden, sind zunächst durch sogenannte Airwires miteinander verbunden (gerade Linien zwischen miteinander verbundenen Pins), ein Autorouter erledigt auf Wunsch einen Teil des Entflechtens der Leiterbahnen. Die Ergebnisse des Autorouters zeigen auf, ob die Plazierung der Bauteile zu routbaren Ergebnissen führen kann: Kann der Autorouter einen hohen Prozentsatz der Leiterbahnen nicht verlegen oder geht er sehr verschwenderisch mit Vias (Durchkontaktierungen) um, ist die Wahrscheinlichkeit hoch, das die Bauteile noch nicht optimal plaziert sind.

Um zu guten Ergebnissen, also kurzen, direkten Leiterbahnen und wenigen Durchkontaktierungen zu gelangen, ist jedoch manuelles Routen der Leiterbahnen nötig. Die Vorstellungskraft des menschlichen Layouters ist hier dem Algorithmus des Autorouters klar überlegen.

Die Masseverbindungen werden über Masseflächen realisiert. Hierdurch werden die EMV-Eigenschaften der Schaltung verbessert, was sich auf die Leistung speziell der analogen Schaltungsteile wie dem Ultraschallverstärker positiv auswirkt.

Das Platinenlayout wurde hinblickend auf SMD-Bestückung erstellt, um die Grösse des Moduls möglichst gering zu halten. Die Bestückung der Bauteile mit einem handelsüblichen SMD-Lötkolben (verwendet wurde ein ERSA Minityp 12V, 6W) sowie einer Pinzette bereitet keine grossen Probleme, lediglich der Mikrocontroller im TQFP32-Gehäuse und der Spannungsregler im SOT23-Gehäuse weisen einen sehr kleinen Pinabstand von 0,80 mm bzw. 0,95 mm auf und sind entsprechend schwieriger zu bestücken.

4.6.5 Gehäuseeinbau

Um das Modul problemlos mit dem LEGO Mindstorms RIS verwenden zu können, ist neben elektrischer auch mechanische Kompatibilität nötig. Denkbar war der Bau eines Gehäuses mit den LEGO-üblichen Noppen oder die Zweckentfremdung eines existierenden LEGO-Gehäusetyps.

Die Wahl fiel auf ein LEGO-Batterie-Gehäuse, in dem normalerweise eine 9V-Blockbatterie Platz findet. Damit die Schaltung in dieses Gehäuse passt, musste sie auf drei Platinen aufgeteilt werden, die sandwichartig übereinandermontiert nicht grösser als eine 9V-Batterie sein durften (Abbildung 4.25).

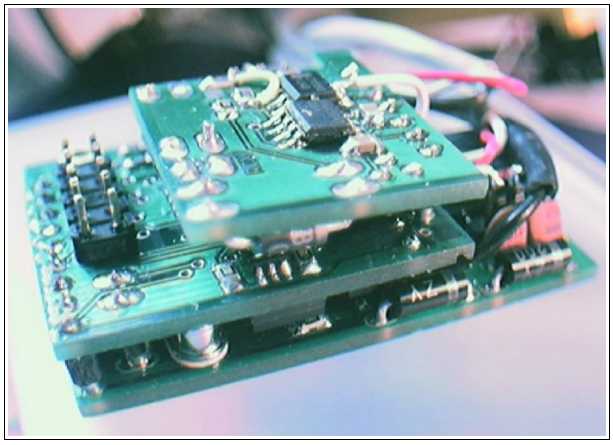

Abbildung 4.25: Die fertig bestückten Platinen

Die in das Batterie-Gehäuse eingelassenen Kontaktreihen zu je 6 Noppen wurden mit einem feinen Bohrer in drei unabhängige Steckfelder getrennt. Eines der Felder dient zum Anschluss der Versorgungsspannung, ein anderes zum Anschluss des RCX-Sensor-Eingangs. Um eine Zerstörung des Moduls durch versehentliche Verpolung zu vermeiden, wurde die Versorgungsspannung über eine Schottky-Diode (SB130) mit dem Modul verbunden.

Das fertig eingebaute Modul ist in Abbildung 4.26 zu sehen. Der freie Platz über dem Transformator wird für die Durchführung der Kabel genutzt. Der Programmieranschluss, der serielle Anschluss und das Potentiometer zur Einstellung des A/D-Wandlers sind ohne Ausbau des Modules erreichbar.
Die in Abbildung 4.25, 4.26 zu sehende Platine entspricht nicht dem endgültigen Platinenlayout im Anhang dieser Arbeit. Die zwei zusätzlichen Brücken sind im endgültigen Layout nicht erforderlich.

Der Ultraschall- und der Infrarotsensor wurden in ein kleines Plastikgehäuse (Abbildung 4.45) eingebaut, das auf einen Legostein mit 2x4 Noppen geklebt wurde. Die Anschlusskabel wurden mit etwa 15 cm Länge bewußt etwas länger ausgelegt, um bei der Montage des Sensors einige Freiheiten zu haben.

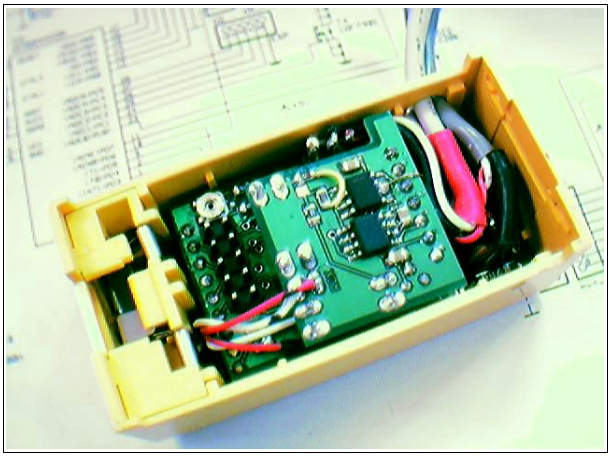

Abbildung 4.26: Gehäuseeinbau

4.7 Software (Mikrocontroller-Programmierung)

Im Hinblick auf die Wiederverwendbarkeit des gesamten Konzeptes wurde die Software in einzelne Funktionsblöcke unterteilt. Diese Funktionsblöcke wurden in Form von Libraries (Bibliotheken) realisiert, die somit auch problemlos in späteren Entwicklungen weiterverwendet werden können.

Als Programmiersprache für das Projekt wurde aus mehreren Gründen Assembler ausgewählt: Es ergeben sich bei geschickter Programmierung Vorteile wie geringe Code-Grösse und hohe Geschwindigkeit, hardwarenahe Programmierung ist in Assembler einfacher und direkter möglich als in für den Controller ebenfalls verfügbaren Hochsprachen wie C oder Basic.

Das Projekt ‚distanz.aps' besteht aus dem Hauptprogramm ‚distanz_main.asm' und etlichen Libraries, die hinzugelinkt werden. Sämtliche trennbaren Programmteile wie Ultraschall-Messung, Infrarot-Messung, RCX-Ausgabe, RCX-Eingabe, etc. wurden in einzelne Libraries verpackt.

Nach dem Reset führt der Mikrocontroller ohne weiteres Zutun etwa 8 Ultraschall- und Infrarot-Messungen pro Sekunde durch. Über ein Terminal-Programm kann über die serielle Schnittstelle (38.400Baud, 8N1) mit dem Modul kommuniziert werden. Das erste empfangene Zeichen, dem kein Befehl zugeordnet ist, führt zur Ausgabe eines Menüs (siehe Abbildung 4.27). Bei Übermittlung eines der angegebenen Zeichen wechselt die entsprechende Einstellung.

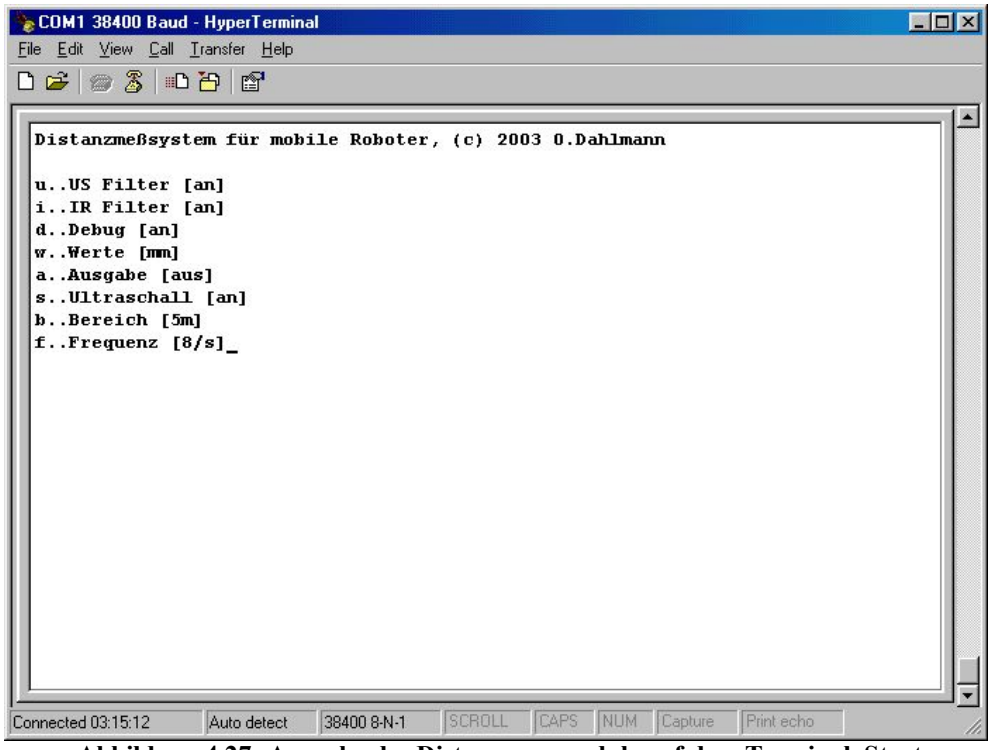

Abbildung 4.27: Ausgabe des Distanzmessmoduls auf dem Terminal, Start

Druck auf die Taste ‚d' gibt pro Messung umfangreiche Informationen aus: Die einzelnen Messwerte (in mm), deren Differenz, ein Ergebniswert in mm und eine Information darüber, wie der Ergebniswert zustande gekommen ist (siehe Abbildung 4.28).

Das Zeichen 'w' toggelt zwischen Ausgabe der Rohdaten und der Ausgabe der in Entfernungswerte umgerechneten Werte. 's' schaltet die Ultraschall-Messung bei Bedarf aus. 'b' dient zur Bereichsumschaltung des analogen Ausgangs, entweder entsprechen 5m oder 1m Maximal-Entfernung dem Aussteuerungsbereich des D/A-Wandlers. Mit 'f' kann die Anzahl der Messungen pro Sekunde von etwa 8/s auf ungefähr 2/s umgeschaltet werden. Die genaue Anzahl der Messungen variert mit der vom Ultraschall-Sensor erkannten Entfernung.

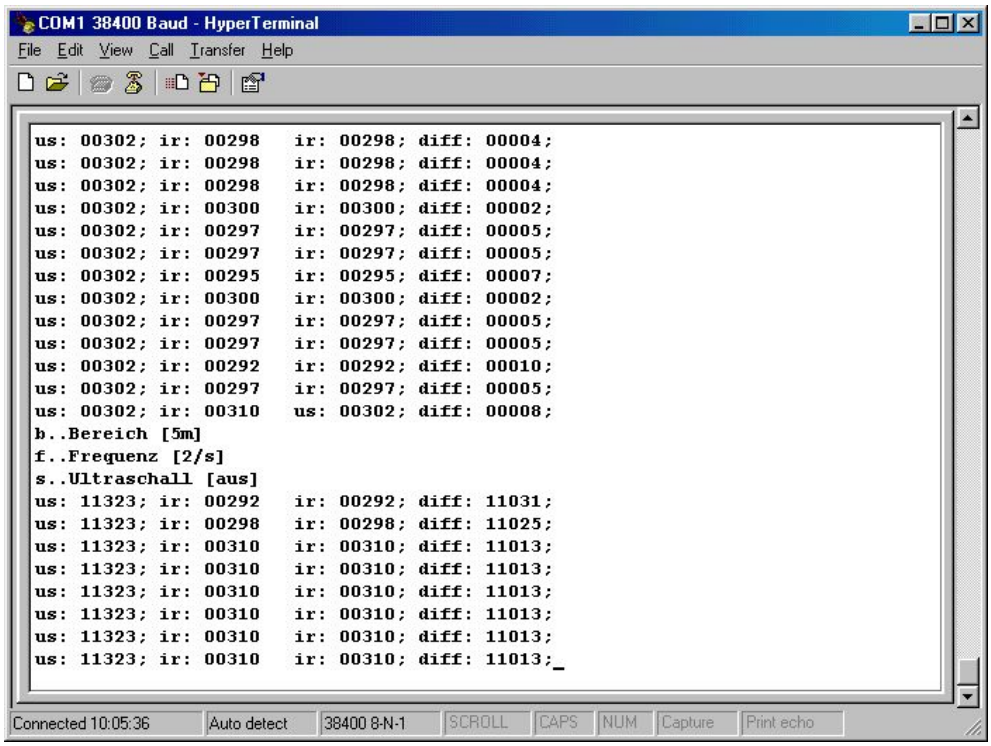

Abbildung 4.28: Ausgabe des Distanzmessmoduls im Debug-Modus

Während der zu sehenden Messdaten-Ausgabe des Distanzmessmoduls hat der RCX-Controller über die in Kapitel 4.8 beschriebene Kommunikationsmethode den Modus des Distanzmessmoduls umgeschaltet. Der Ultraschallsensor wurde deaktiviert, die Messfrequenz auf 2 Messungen pro Sekunde verringert.

In Abbildung 4.29 ist der Ablauf des auf dem Atmel-Controller ablaufenden Programmes zu sehen. Die Implementierung der Ultraschall- und Infrarot-Messung sowie die Filterung der Messwerte werden in den nachfolgenden Kapiteln erläutert.

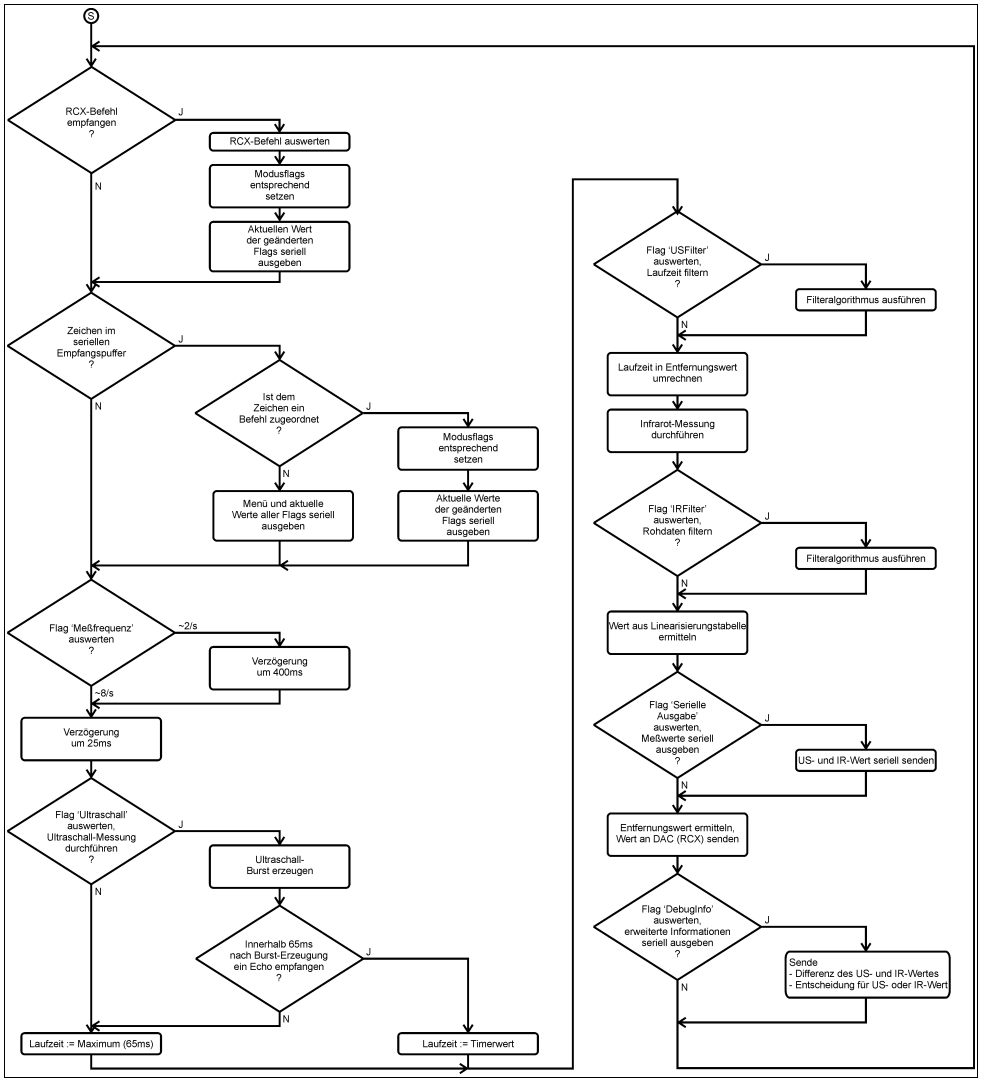

Abbildung 4.29: Ablaufdiagramm der Hauptprogrammschleife

Nachfolgend ein Diagramm (Abbildung 4.30), in dem der vereinfachte Programmablauf mit den quasi-parallel ablaufenden Timer- und UART-Interrupts in einer einem Sequenz-Diagramm ähnlichen Notation dargestellt wird.

Der Timer-Interrupt wird alle 2ms aufgerufen, aktualisiert den Ausgabewert des D/A-Wandlers und fragt den am RCX-Sensor-Eingang anliegenden Pegel ab, um die Kommunikation mit dem RCX wie in Kapitel 4.8 beschrieben zu ermöglichen.

Der UART-Interrupt-Handler sendet eigenständig die im Sendepuffer anstehenden Bytes und schreibt die empfangenen Bytes seinerseits in einen Empfangspuffer. Beide Puffer sind als Ringpuffer ausgelegt.

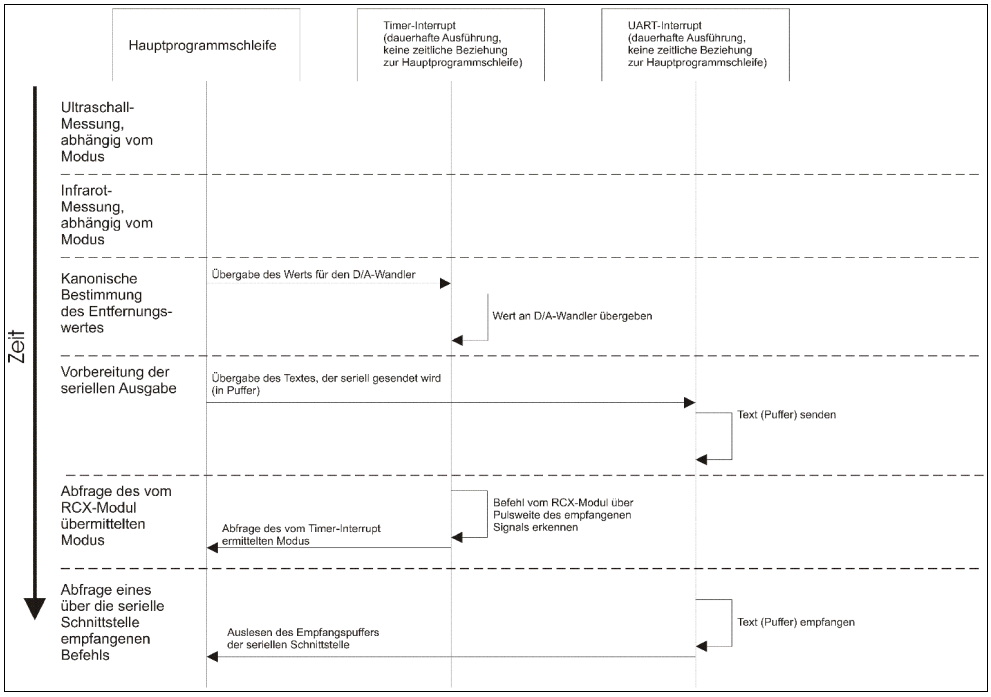

Abbildung 4.30: Programmablauf, parallele Prozesse

4.7.1 Ultraschall-Messung

Direkt vor der Erzeugung von 8 Perioden eines 50kHz-Bursts an einer der I/O-Port-Leitungen des Controllers wird ein Timer gestartet, der beginnend von Null in Abständen einer Mikrosekunde Impulse zählt. Diese Impulse werden aus der Taktfrequenz des Mikrocontrollers von 8MHz und einem in den Timer integrierten Vorteiler (programmiert auf CK/8) erzeugt. Nach einer kurzen Zeitspanne werden die zuvor gesperrten Interrupts freigegeben. Die Interrupt-Sperre ist nötig, damit nicht bereits der Burst selbst oder das Nachschwingen des Transducers als Echo erkannt werden.

Ein empfangenes Ultraschall-Echo führt zu einem Low-Pegel am ICP(Input Capture)-Eingang des Controllers. Hierauf speichert der Controller den Zählerstand des Timers und löst einen InputCapture-Interrupt aus. Die Laufzeit des Echos in Mikrosekunden ist hiermit bekannt und kann auf einfache Weise in einen Entfernungswert, gemessen in mm, umgewandelt werden. Die Messung ist hiermit beendet.

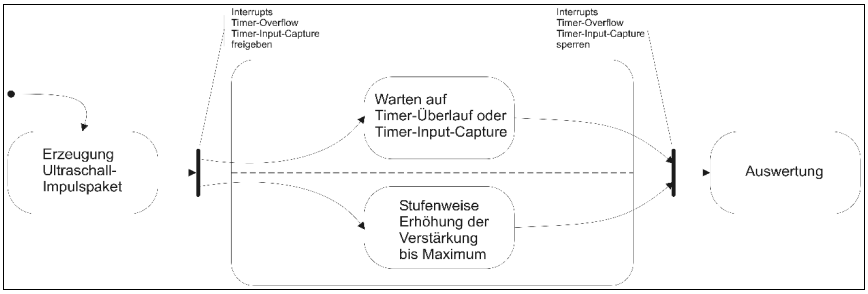

Abbildung 4.31: Ablauf Ultraschall-Messung

Bleibt das Ultraschall-Echo aus, läuft der 16Bit-Zähler nach etwa 65,5 Millisekunden über (65.535 -> 0). Daraufhin wird ein Overflow-Interrupt ausgelöst, der die Messung beendet. Da keine Entfernung gemessen werden konnte, muss davon ausgegangen werden, dass sich kein für das Ultraschall-Signal erkennbares Objekt vor dem Sensor-Modul befindet. Die Software begrenzt durch den nach 65ms überlaufenden Timer die maximal messbare Entfernung auf etwa 10m. In der Praxis sind die von weiter als 5m entfernten Objekten reflektierten Echos in der Regel zu klein, um noch von der Elektronik wahrgenommen zu werden, da die Amplituden der Signale dann im Rauschen und im Übersprechen der digitalen Signale auf den analogen Schaltungsteil untergehen.

Während der Controller nach Erzeugen des Bursts auf das Eintreffen eines Interrupts wartet, wird die Verstärkung des Ultraschall-Empfängers zunächst auf 20x erhöht, dann linear mittels des Digital-Potentiometers (Library: ‚X9C104_lib.asm') erhöht. Die einzelnen Erhöhungs-Schritte werden nach jeweils gleichem zeitlichen Abstand von etwa 2,1ms aus einer Tabelle (GAINTAB, in der Library 'Ultrasonic_lib.asm' zu finden) gelesen und können somit einfach verändert werden. Die Werte in der Tabelle stellen keine absoluten Werte dar, sondern relative Erhöhungen zum zuletzt eingestellten Verstärkungsfaktor: Das hier verwendete Digitalpoti kennt kein serielles Protokoll, um einen absoluten Wert zu setzen, stattdessen wird über drei Leitungen ein interner Up/Down-Counter gesteuert. Ein Reset-Eingang ist nicht vorhanden. Zu Beginn jeder Messung wird das ‚Potentiometer' daher auf Nullstellung gebracht, indem 100 Schritte in Down-Richtung ausgeführt werden. Dies ist möglich, da nach Erreichen der untersten 'Schleiferposition' diese auch mit weiteren Dekrementierungen beibehalten wird, es findet kein Unterlauf des internen Zählers statt.

4.7.2 Infrarot-Messung

Zunächst wird die Betriebsspannung des Infrarot-Sensors über eine I/O-Leitung des Controllers eingeschaltet. Nach einer Wartezeit im Bereich einiger Millisekunden, die nötig ist, damit der Sensor seine interne

Elektronik initialisiert hat und sinnvolle Werte liefert, werden insgesamt 8 Analog-Digital-Wandlungen gestartet.

Der A/D-Wandler des Controllers ist mit einer maximalen Samplingrate von etwa 15kHz hinreichend schnell, um die 8 Wandlungen in ungefähr einer halben Millisekunde auszuführen. Ein Objekt, dass sich mit 130 km/h bewegt, legt in dieser Zeit 2 cm zurück.

Der Infrarot-Sensor arbeitet jedoch sehr viel langsamer, so dass durch diese Mittelwertbildung nicht mehrere unabhängige Messwerte des Sensors verarbeitet werden, sondern vielmehr Messfehler des A/D-Wandlers und des Sensors aufgrund von Rauschen und transienten Überlagerungen auf den Versorgungsspannungen herausgefiltert werden.

Wie praktische Versuche gezeigt haben, wird die statistische Abweichung des gemittelten Messwertes vom erwarteten Wert mit 8 Messungen bereits signifikant geringer.

Wie im folgenden Kapitel gezeigt wird, wird der so erhaltene Wert vor der weiteren Verarbeitung gefiltert. An dieser Stelle wird nicht weiter auf den Prozess der Filterung eingegangen.

Da das Ausgangssignal nicht proportional zur gemessenen Entfernung ist, muss der Messwert linearisiert werden. Dies geschieht über eine Linearisierungstabelle: Beginnend beim theoretischen Maximal-Wert des A/D-Wandlungs-Vorgangs (dezimal 1023) beginnt eine Tabelle mit abnehmenden Messwerten für Entfernungswerte von 0..800 mm in äquidistanter Entfernung von 10 mm. Der erste Tabelleneintrag (W_K), der kleiner als der gemessene Wert (W) ist, wird gesucht. Der Index des Tabelleneintrags entspricht der Entfernung in cm (E_K).

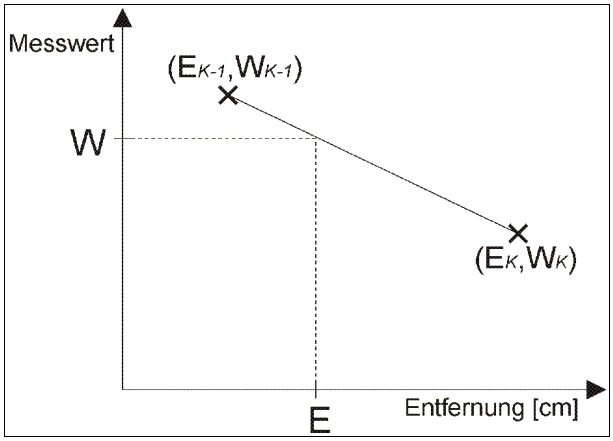

Abbildung 4.32: Lineare Interpolation (Infrarot-Sensor)

Die Schnittpunktbestimmung von W mit der durch die beiden Punkte (E_{K-1}, W_{K-1}) und (E_K, W_K) aufgespannten Geraden führt zum Ergebnis E, der gesuchten Entfernung.

$$E = \frac{W - W_K}{W_{K-1} - W_K} \cdot E_{K-1} + \frac{W_{K-1} - W}{W_{K-1} - W_K} \cdot E_K$$

Um ein 16bittiges ganzzahliges Ergebnis mit hinreichender Genauigkeit zu erhalten, wurden Divisionsroutinen mit 24Bit und Multiplikationsroutinen mit 16Bit und 32bittigem Ergebnis implementiert.

Die Aufnahme der erforderlichen Messwerte, um eine Linearisierungstabelle aufstellen zu können, geschieht über den Mitschnitt der Terminalausgaben mit der Funktion 'Linearisierung'. Die genaue Anwendung dieser Funktion wird im Anhang beschrieben.

Von den jeweils 32 Messwerten pro Entfernung ist der geometrische Mittelwert zu bilden, dies kann z.B. mit 'OpenOffice Calc' geschehen. Mit den entstandenen Mittelwerten lässt sich ein Regressionspolynom berechnen, dass für beliebige Entfernungswerte im erlaubten Bereich (0 bis 80 cm) einen Messwert annähern kann. Hierbei wird ein Polynom errechnet, dass die gemessenen Paare bestmöglich, also mit geringster Fehlerquadratsumme, annähert. Wird der Grad des Polynoms zu hoch gewählt, reproduziert das Polynom in zunehmender Weise die Messfehler, die an dieser Stelle herausgemittelt werden sollen. Ein Polynom 6. Grades führt jedoch zu einer guten Annäherung und geringen Abweichungen zu den gemessenen Paaren (x mm; y wert) (siehe Abbildung 4.33), das

Bestimmtheitsmass des dargestellten Polynoms liegt bei 0,9998 und der Korrelationskoeffizient bei 0,9999.

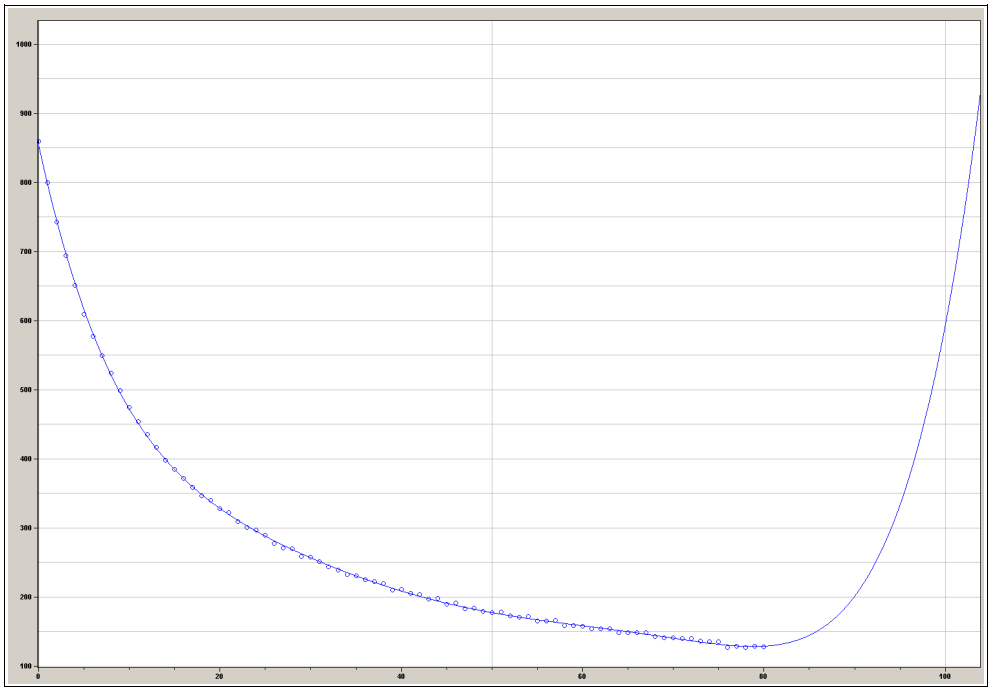

Abbildung 4.33: Polynom-Regression, Polynom und Messwerte

Während die die tatsächlichen Messwerte repräsentierenden Kreise noch für grössere Entfernungen alternierende Vorzeichen der Messwertänderung zeigen, ist der durch das Polynom angenäherte Verlauf für grösser werdende Entfernungen monoton fallend, ein wichtiges Kriterium für die einwandfreie Funktion des Linearisierungsalgorithmus. Der Bereich, in dem das Polynom streng monoton fällt, also jeder Messwert M_x eindeutig einer Entfernung x zugeordnet werden kann, lässt sich zur eindeutigen Entfernungsbestimmung nutzen. Im dargestellten Fall reicht dieser Bereich von x=0 bis x=78 [cm].
Anhand dieses Polynoms werden 80 Wertepaare zum Aufbau der Linearisierungstabelle errechnet.

4.7.3 Messwert-Filterung

Die unabhängig voneinander gewonnenen Messwerte des Ultraschall- und Infrarot-Sensors sind mit einer gewissen Unsicherheit behaftet.

Da das Distanzmessmodul keine Informationen vom Roboter über die Bewegungsrichtung und Geschwindigkeit erhält, fällt es schwer, den weiteren Verlauf der Messgrössen vorherzusagen. Weder kann das Modul einschätzen, ob sich die Messgrösse im weiteren zeitlichen Verlauf erhöhen

oder verringern sollte, noch können Änderungen in den Messwerten eindeutig der Bewegung des Roboters zugeordnet werden, genauso ist es in dynamischen Szenarien möglich, dass sich die Umgebung selbst verändert hat.

Im Falle des Ultraschallsensors kann es sein, dass der Schall über mehrere Objekte reflektiert wird, bevor ein Echo zurück zum Sensor gelangt. Ebenso ist es möglich, dass sich das Ultraschallsignal im Raum verliert, obwohl sich ein Hindernis unmittelbar vor dem Sensor befindet. Aufgrund logischer Überlegungen kann die Konsistenz der einzelnen Messwerte ermittelt werden:

Der Ultraschallsensor misst die Zeit bis zur Reflektion eines ausgesandten Signals. Das Signal kann problemlos an einem Objekt reflektiert werden oder aber das Objekt streut das Signal in die Umgebung, was eine (möglicherweise massive) Verlängerung der Laufzeit zur Folge hat. Dies kann schon bei geringfügig anderen Winkeln der Objektoberfläche zum Sensor als 90° der Fall sein, tritt also in der Praxis häufig auf (siehe Abbildungen 'Ultraschall-Werte, 0°..30°'). Auf keinen Fall, elektronische Fehler einmal ausser Acht gelassen, wird jedoch eine geringere Zeit als die theoretisch resultierende zurückgegeben.

Verringerungen der Laufzeit sind also grundsätzlich glaubwürdig (Messergebnis mit hoher Wahrscheinlichkeit), starke Erhöhungen der Laufzeit in Folge zweier Messungen müssen nicht unbedingt der Realität entsprechen (Messergebnis mit geringerer Wahrscheinlichkeit). Es scheint also sinnvoll, einen neuen Messwert (direkt) zu übernehmen, wenn eine Verringerung der Entfernung im Vergleich zur vorigen Messung gemessen wurde. Ein Messwert, der eine (starke) Erhöhung der Laufzeit zeigt, könnte nach Filterung in das Ergebnis einfließen, indem die Vorgeschichte des Messverlaufs stärker berücksichtigt wird (rekursive Filterung).

Hierdurch würden kurzzeitige Aussetzer der Messung unterdrückt werden, es ist jedoch darauf zu achten, dass ein anhaltender hoher Messwert (der ja nicht zwangsläufig eine Störung sein muss) in bestimmter Zeit in das Ergebnis einfliesst.

$$D_{US(n+1)} = \begin{cases} M_{US(n)} & \forall\, M_{US(n)} \le D_{US(n)} \\ x \cdot M_{US(n)} + (1-x) \cdot D_{US(n)} & \forall\, M_{US(n)} > D_{US(n)} \quad mit \quad 0 < x \le 1 \end{cases}$$

Der neue ermittelte Entfernungswert $D_{US(n+1)}$ ist ein gewichteter Mittelwert aus dem vorigen Entfernungswert $D_{US(n)}$ und dem aktuellen Messwert $M_{US(n)}$.

Je näher der Faktor x gegen 1 läuft, desto schneller werden gestiegene Messwerte in das Ergebnis übernommen.

Der Infrarot-Sensor zeigt keine derart hohe Winkelabhängigkeit, jedoch ein Rauschen in Höhe von etwa 2% des Wertebereichs des A/D-Wandlers, wie in schneller Folge über den A/D-Wandler des AVR-Prozessors gesampelte Messungen zeigen (siehe Aufnahme der Linearisierungstabelle).
Mit zunehmenden Entfernungen wird die Änderung des Messwertes geringer, so dass ungefilterte Messwerte aufgrund des Rauschens eine höhere Ortsungenauigkeit aufweisen.

Um den Einfluss dieses Messfehlers zu verringern, werden zunächst in schnellstmöglicher Folge 8 Wandlungen vorgenommen und aus diesen der Mittelwert gebildet. Wie zu erwarten, ist das derart erzielte Ergebnis schon deutlich näher am theoretischen Wert als die gesampelten Einzelmessungen.
Aufgrund der sehr kleinen Zeitspanne, die dieser Vorgang samt Mittelwertbildung dauert, wird das Ergebnis im Folgenden als Einzelmessung behandelt.

Der beschriebene Vorgang wird abhängig vom gewählten Modus bis zu 10 mal in der Sekunde wiederholt.

Der so erzeugte Rohdatenwert wird im Folgenden abhängig von seiner Grösse gefiltert. Der Ergebniswert $D_{IR(n+1)}$ hängt sowohl vom aktuellen Messwert $M_{IR(n)}$ als auch von seiner Vorgeschichte $D_{IR(n)}$ ab:

$$D_{IR(n+1)} = g(M_{IR(n)}) \cdot M_{IR(n)} + (1 - g(M_{IR(n)})) \cdot D_{IR(n)} \quad mit \quad 0 \le g(M_{IR(n)}) \le 1$$

$g(M_{IR(n)})$ sei die lineare Gewichtungsfunktion:

$$g(M_{IR(n)}) = \frac{a \cdot M_{IR(n)}}{M_{IRmax}} + b \quad mit \quad \begin{cases} 0 \le b \le 1 \\ a + b \le 1 \\ a > 0 \end{cases}$$

M_{IRmax} sei hierbei der Wertebereich des Messwertes $M_{IR(n)}$, der im wesentlichen durch den A/D-Wandler bestimmt wird, dessen Wertebereich 0..1023 umfasst. Die Gewichtungsfunktion $g(M_{IR(n)})$ beschreibt einen linearen Zusammenhang zwischen der Glaubwürdigkeit eines Messwertes und dessen Höhe. Für den Sonderfall a=0, b=1 wird der Ergebniswert $D_{IR(n+1)}$ nur durch den aktuellen Messwert bestimmt.
Dadurch berücksichtigt der Filter, dass kleinere Messwerte, die grössere Entfernungen darstellen, das Ergebnis langsamer beeinflussen. Zu grösseren,

in der Nähe der Aussteuerungsgrenze des A/D-Wandlers befindlichen Messwerten konvergiert D_{IR} hingegen sehr schnell.

Der lineare Zusammenhang zwischen Messwerthöhe und propagierter Wahrscheinlichkeit des Messwertes trifft den wahren Verlauf zwar nicht ganz, führt aber dennoch zu guten Ergebnissen und ist, ohne die Notwendigkeit weiterer Tabellen oder komplexeren mathematischen Funktionen, problemlos auf dem AVR-Controller implementierbar.

Vor der weiteren Verarbeitung wird der Messwert linearisiert, hierbei wird auf eine Tabelle aus 80 Messwerten zurückgegriffen, die sich im EEPROM des Atmel-Mikrocontrollers befindet. Die aus 16Bit-Worten bestehende Tabelle beginnt mit dem Wert 1023, gefolgt von 80 Messwerten und endet mit dem Wert 0.

Schließlich werden beide derart vorgefilterten Messwerte zu einem Entfernungswert kombiniert, indem Fallunterscheidungen vorgenommen werden:

Liefern beide Sensoren einen Messwert zurück, wird im Folgenden der kleinere Messwert zur Bestimmung des Rückgabewertes herangezogen.

Erkennt der Infrarot-Sensor kein Hindernis (Entfernung > 80 cm), der Ultraschall-Sensor jedoch schon, wird der Ultraschall-Wert übernommen.

Liefert keiner der Sensoren einen Messwert zurück, befindet sich kein Objekt im Messbereich. Dem Zielsystem wird der höchstmögliche Entfernungswert mitgeteilt.

Vor Übermittlung des Messwertes an den RCX werden die an der Front des Roboters angebrachten Taster abgefragt. Sollte eine Berührung mit einem Hindernis stattfinden, so wird der kleinstmögliche Wert an den RCX gesendet. Das Ergebnis der Ultraschall- bzw. Infrarotmessung wird in seinem Nullpunkt verschoben, so dass die Berührung eines Tasters vom RCX einwandfrei als solche zu erkennen ist.

4.8 Software (RCX-Programmierung)

Das Distanzmessmodul wird an einen der drei Sensoreingänge des RCX angeschlossen. Die A/D-Wandlung der analogen Eingangsspannungen der drei Anschlüsse wird von brickOS im Hintergrund durchgeführt. Die vom A/D-Wandler ausgegebenen 10Bit-Werte werden auf vorzeichenlose 16Bit-Darstellung erweitert und in den globalen Variablen AD_C (SENSOR_1),

AD_B (SENSOR_2) und AD_A (SENSOR_3) gespeichert. Die in Klammern angegebenen Variablennamen mit Sensorbezeichnung sind in [include/dsensor.h] definiert und können anstelle von AD_A - AD_C verwendet werden.

Der nutzbare Bereich umfasst etwa 0x4500 - 0xFFFF (zu kleineren Werten hin ist dieser durch den strombegrenzenden Widerstand R_{27} beschränkt). In der Praxis wurde eine Entfernung von d=0mm mit Berührung durch den Wert 0x4500 dargestellt, der Abstand d=0mm ohne Berührung führte zum Wert 0x4700, die obere Grenze des ausgewählten Messbereichs liegt bei 0xFFC0. Der vom RCX ermittelte Wert verhält sich linear zur gemessenen Entfernung. Der Zusammenhang zwischen Entfernung und vom RCX gemessenen Wert wird in Kapitel 4.9.2 verdeutlicht.

Die drei zweiadrigen Sensor-Eingänge des RCX-Moduls sind sehr vielseitig: Einerseits kann ein analoger Messwert vom RCX gelesen werden, andererseits kann ein angeschlossener Sensor über dasselbe Adernpaar mit Strom versorgt werden. Hierzu schaltet der RCX-Controller zwischen Anlegen der Versorgungsspannung und Samplen des anliegenden analogen Signals wie in Kapitel 4.5.6 beschrieben in schneller Folge um.

Das gezielte Ein- und Ausschalten dieser Versorgungsspannung wird hier zur Datenübertragung genutzt, der RCX ist hiermit in der Lage, dem Distanzmessmodul einfache Kommandos zu übermitteln.

Die Software des Atmel-Controllers unterscheidet zwischen 4 Kommandos, indem die Dauer ermittelt wird, die eine vom RCX-Modul eingeschaltete Versorgungsspannung stabil anliegt. Unterschieden wird zwischen den Zeiten 24ms, 40ms, 56ms und 72ms. Ungenauigkeiten im Bereich einiger ms werden sicher abgefangen.

Bei der Erzeugung von Impulsen genau definierter Länge mittels folgender Befehle:

```
ds_active(SENSOR_x);        // HIGH-Pegel (~8V)
delay(20);                  // 20 ms Verzögerung
ds_passive(SENSOR_x);       // LOW-Pegel (~5V)
```

stösst man auf ein Problem: Die Zeit, die der Ausgangspegel auf etwa 8V-Potential springt, scheint in grossem Maße zufällig und in jedem Fall viel länger als 20ms zu sein.

Der RCX-Controller unterbricht laut Dokumentation (siehe auch [kernel]) zu jeder Millisekunde den Programmablauf, um z.B. die Ansteuerung der

Motoren, Sensor-Abfrage, Ton-Erzeugung, Task-Wechsel, etc. durchzuführen. Dies geschieht über einen Timer-Interrupt.

Laut Dokumentation ist die Zeitscheibe, die einem Task standardmäßig zur Verfügung steht, 20ms lang. Dies erklärt die hohen zeitlichen Abweichungen (Jitter) beim Versuch, Pegelwechsel mit Genauigkeiten im Millisekundenbereich herbeizuführen.

Das Verhindern der Interrupts mittels der Funktion *disable_irqs()* führte jedoch nicht zum Erfolg (Funktion wird deklariert in [sys/irq.h]).

In der Dokumentation zu [sys/irq.h] stösst man auf einen Verweis auf den NMI ('*N*on *M*askable *I*nterrupt', also 'nicht sperrbarer Interrupt'), der durch die Funktion systime_init() (siehe auch [kernel/systime.c]) initialisiert wird. Der NMI wird vom internen Watchdogtimer WDT getriggert. Der Handler des NMI sorgt u.a. in einer erneuten Freigabe der Timer-Interrupts.

Um ein Ausschalten der Timer-Interrupts über einen längeren Zeitraum zu ermöglichen, ist also entweder eine umfangreiche Änderung des brickOS-Kernels notwendig oder der WDT muss an der Generierung eines NMI gehindert werden. Ausschalten lässt sich der WDT nicht, die Art des ausgelösten Interrupts kann jedoch geändert werden:

Der WDT lässt sich auf die Auslösung eines normalen, sperrbaren, Overflow-Interrupts umschalten (Bit6 in TCSR, siehe H8 Programming Manual). Dies bewirkt dann nach Sperren der Interrupts das gewünschte Verhalten, nämlich keinerlei Code-Ausführung, keinerlei Verzögerung des laufenden Programms. In Assembler sieht das folgendermaßen aus (Assembler-Befehlsübersicht siehe [Hitachi H8]):

```
push    r0                  ; Register r0 absichern, wird verwendet

mov.b   #0XA5,r0h           ; r0 := A5XX, A5 ist das 'Codewort', um in TCSR
                            ; schreiben zu dürfen

mov.b   @0xFFA8,r0l         ; der Inhalt von TCSR wird in das Lowbyte von
                            ; r0 geschrieben

and.b   #0b10111111,r0l     ; Bit6 wird gelöscht

mov.w   r0,@0xFFA8          ; r0 wird nach TCSR zurückgeschrieben

pop     r0                  ; Registerinhalt wiederherstellen
```

Die Routine, die den WDT wieder auf NMI-Auslösung zurückstellt, sieht bis auf einen kleinen Unterschied genauso aus: Aus der *and.b*-Zeile wird folgende Zeile:

```
or.b    #0b01000000,r0l          ; Bit6 wird gesetzt
```

Mit einer Verzögerungsschleife, bestehend aus der wiederholten Ausführung eines NOP (No OPeration, einen Taktzyklus lang nichts tun)-Befehls, wird die entsprechende Impulslänge erzeugt. Normalerweise sollte man diese Art der Verzögerungsschleife in Multitasking-Umgebungen tunlichst vermeiden, aber da das Taskswitching sowieso deaktiviert wurde, macht es natürlich keinen Unterschied.

Geschrieben in C, vermischt mit Inline-Assembler, sieht die vollständige Funktion zum Senden eines Kommandos folgendermaßen aus:

```
extern inline void WatchDog_Sperren()
{
  __asm__ __volatile__("\tpush r0\n":::"cc");
  __asm__ __volatile__("\tmov.w #0,r0\n":::"cc");
  __asm__ __volatile__("\tmov.b #0xA5,r0h\n":::"cc");
  __asm__ __volatile__("\tmov.b @0xFFA8,r0l\n":::"cc");
  __asm__ __volatile__("\tand.b #0b10111111,r0l\n":::"cc");
  __asm__ __volatile__("\tmov.w r0,@0xFFA8\n":::"cc");
  __asm__ __volatile__("\tpop r0\n":::"cc");
}

extern inline void WatchDog_Aktivieren()
{
  __asm__ __volatile__("\tpush r0\n":::"cc");
  __asm__ __volatile__("\tmov.w #0,r0\n":::"cc");
  __asm__ __volatile__("\tmov.b #0xA5,r0h\n":::"cc");
  __asm__ __volatile__("\tmov.b @0xFFA8,r0l\n":::"cc");
  __asm__ __volatile__("\tor.b #0b01000000,r0l\n":::"cc");
  __asm__ __volatile__("\tmov.w r0,@0xFFA8\n":::"cc");
  __asm__ __volatile__("\tpop r0\n":::"cc");
}

static void SendCommand(int COMMAND)
{
  int verz;
  disable_irqs();                          //Interrupts sperren
  WatchDog_Sperren();                      //Watchdog -> Intervall-Timer
  bit_set(&PORT6, 2);                      //aktiv, High-Pegel RCX-Sensor
  for (verz=0;verz<(1970+1310*(COMMAND-1));verz++)
    {
    __asm__ __volatile__("\tnop\n":::"cc"); //Verzögerungsschleife mit NOP's
    }
```

```
  bit_clear(&PORT6, 2);                 //passiv, Low-Pegel RCX-Sensor
  WatchDog_Aktivieren();                //Watchdog -> NMI
  enable_irqs();                        //Interrupts erlauben
}
```

Bei der Implementierung der Verzögerungsschleife trat ein Problem auf: Um die verschiedenen Verzögerungszeiten zu erzielen, wird die Nummer des zu übermittelnden Kommandos mit einem Faktor multipliziert. Bei einem Faktor von 1310 funktioniert die Verzögerungsschleife wie erwartet, bei Veränderung auf den (mathematisch korrekten) Faktor 1313 verringert sich die erzielte Impulsdauer um den Faktor 5. Ob dies ein Fehler des GNU-C-Compilers oder ein Feature im Sinne der Code-Optimierung durch den GCC (Verhinderung von NOP-Schleifen) ist, ließ sich in der zur Verfügung stehenden Zeit nicht mehr klären.

Nach der Wiederherstellung des NMI und Reaktivieren der Interrupts führt der RCX das unterbrochene Programm fort. Ein Nachteil dieser Methode ist, dass sowohl die Motortreiber (PWM) als auch die Tonausgabe während der Impulsausgabe nicht aktualisiert werden. Kurzzeitige Änderungen der Motorgeschwindigkeit oder hörbare Auswirkungen auf die Tonerzeugung können die Folge sein. Die Systemuhr bleibt während dieser Zeit ebenfalls stehen.

4.9 Funktions- und Leistungstest

4.9.1 Messwerte in Abhängigkeit von Winkel und Entfernung

Nachfolgend werden 5 Messreihen betrachtet (Abbildung 4.35-4.44), bei denen ein Karton (Höhe: 35 cm, Breite: 21 cm) als Messobjekt verwendet wurde. Auf der X-Achse der Diagramme ist die Entfernung zum Objekt aufgetragen, die Y-Achse zeigt die Laufzeit (Ultraschall) bzw. den vom A/D-Wandler gemessenen Wert. Jede dieser Messreihen wurde mit einem anderen Winkel des Objektes durchgeführt, der Karton stand hierzu auf einem verschiebbaren Schlitten, auf dem eine Skala mit verschiedenen Winkeln aufgeklebt wurde (Abbildung 4.34). Der Sensor befand sich zunächst in einer Höhe von etwa 7 cm, musste jedoch höher montiert werden, da ansonsten die 1cm hohe Kante des Schlittens ab einer Entfernung von etwa 45 cm zum Sensor als Hindernis erkannt wurde. Der Winkel α (siehe nachfolgende Grafik) des Objekts nahm 0°, 5°, 10°, 20° und 30° ein. Der sich dabei ergebende Offset wurde in der Weiterverarbeitung der Messreihen berücksichtigt.

Mit zunehmendem Winkel wird deutlich, dass die Qualität der Ultraschallmessung abnimmt, während die Infrarot-Messung deutlich unabhängiger vom Winkel ist.

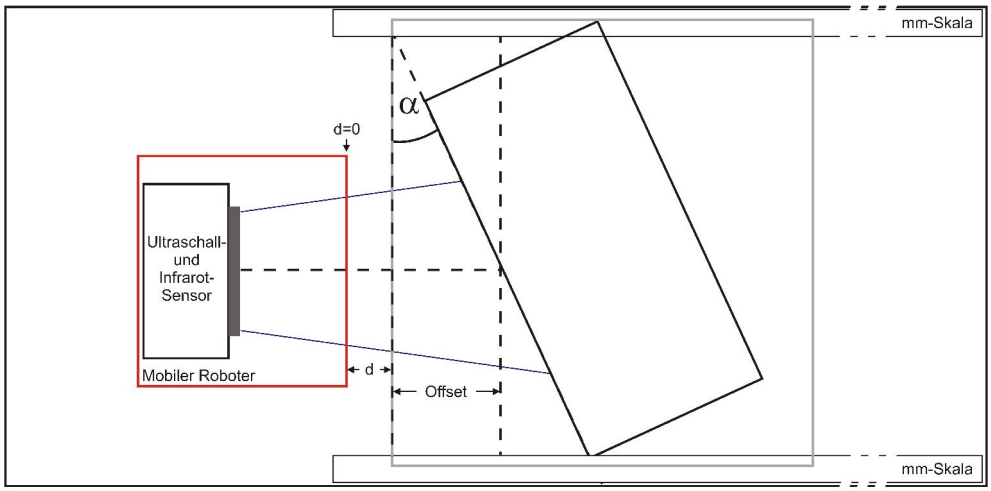

Abbildung 4.34: Aufnahme der Linearisierungstabellen, Draufsicht

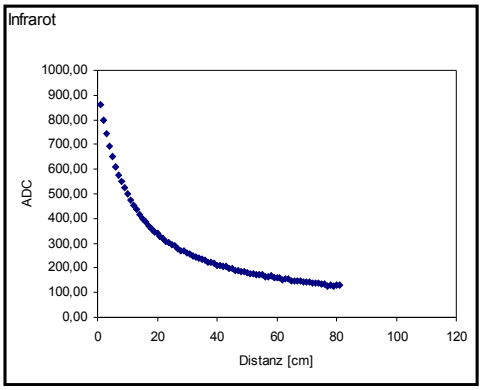

Abbildung 4.35: Infrarot-Werte, 0°

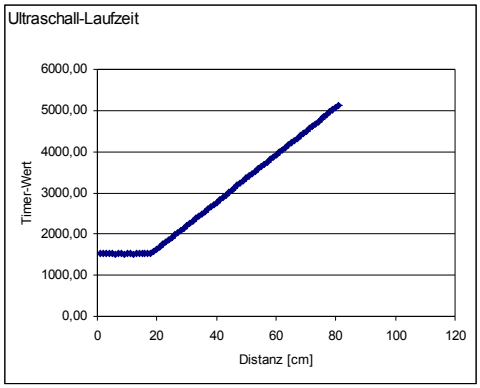

Abbildung 4.36: Ultraschall-Werte, 0°

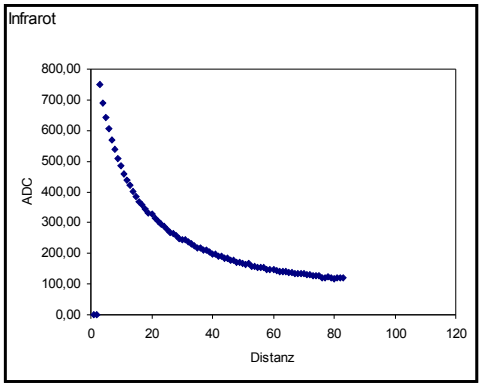

Abbildung 4.37: Infrarot-Werte, 5°

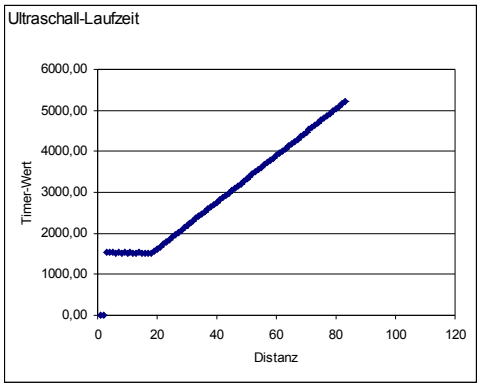

Abbildung 4.38: Ultraschall-Werte, 5°

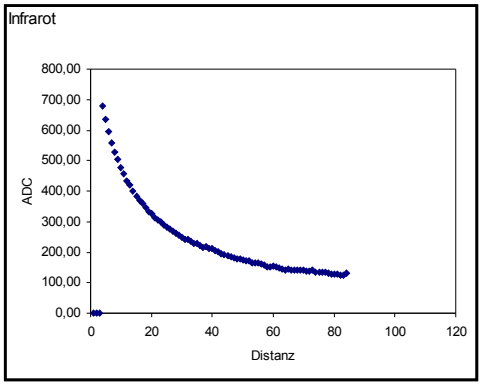

Abbildung 4.39: Infrarot-Werte, 10°

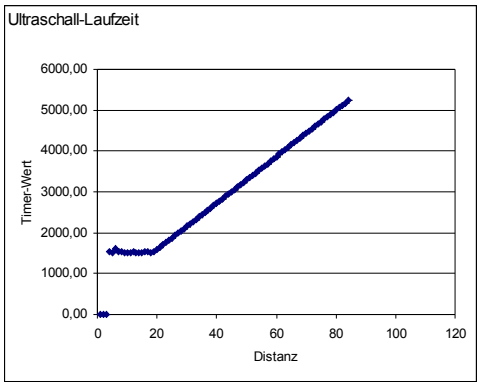

Abbildung 4.40: Ultraschall-Werte, 10°

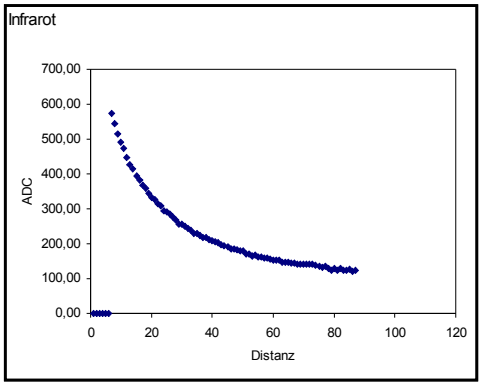

Abbildung 4.41: Infrarot-Werte, 20°

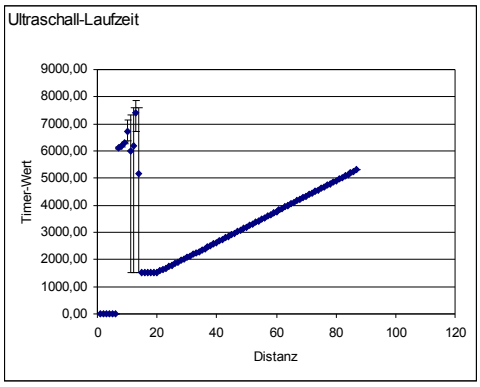

Abbildung 4.42: Ultraschall-Werte, 20°

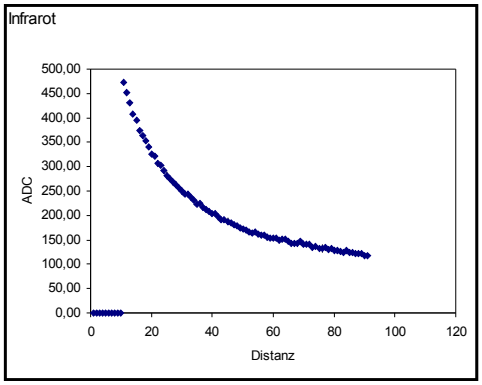

Abbildung 4.43: Infrarot-Werte, 30°

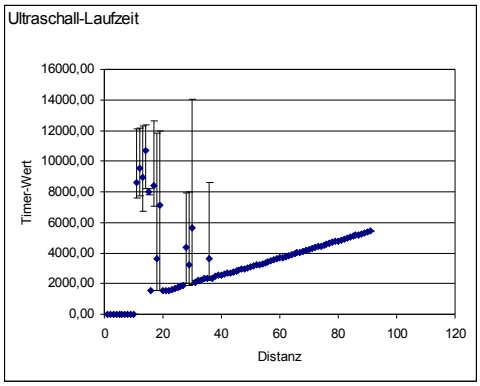

Abbildung 4.44: Ultraschall-Werte, 30°

In den Diagrammen sind neben den Mittelwerten von jeweils 5 Ultraschallmessungen pro Messpunkt auch die Abweichungen in Form von Minima und Maxima angegeben. Bei den Winkeln 20° und 30° ist zu sehen, das im Nahbereich bis etwa 40 cm Distanz das Ultraschallsignal in die Umgebung gestreut und über weitere Objekte der Umgebung schließlich wieder zurückgeworfen wird. Das über mehrere Objekte reflektierte Signal ist jedoch so schwach, dass es nicht mehr stabil als Echo erkannt wird, daher ergeben sich starke Streuungen des Messwertes.

4.9.2 Linearität

In einem weiteren Test wurde die Entfernung zu 4 verschiedenenen Hindernissen bestimmt und danach mit dem Distanzmessmodul gemessen. Der auf dem LCD-Display des RCX hexadezimal angezeigte Wert wurde gegen die Entfernung in Abbildung 4.45 aufgetragen.

Der erfasste Messwert wird im Distanzmessmodul durch den D/A-Wandler in einen analogen Wert gewandelt, im RCX geschieht der umgekehrte Vorgang. Trotz der doppelten Wandlung ist der angezeigte Wert linear und nahezu störungsfrei.

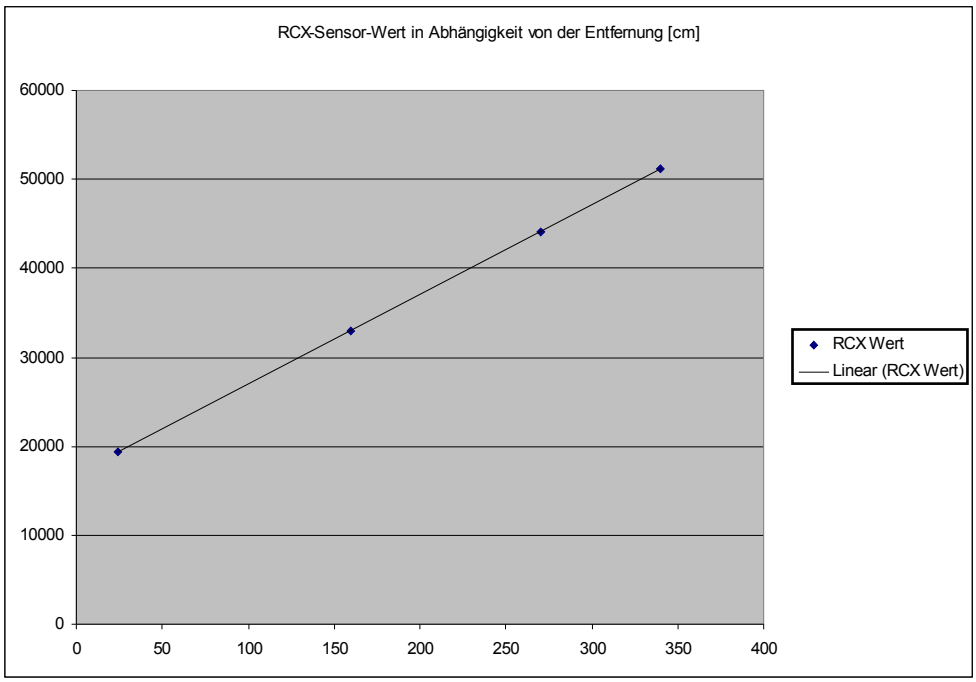

Abbildung 4.45: RCX-Sensor-Wert in Abhängigkeit von der Entfernung

4.9.3 Objekte geringer Grösse

Es wurde ein Test durchgeführt, ob Objekte sicher erkannt werden, wenn sich diese nicht auf der gedachten Ideallinie befinden. Als Objekt wurde eine 0,7l-PET-Flasche, die in 10cm (Abbildung 4.47) bzw. 30cm (Abbildung 4.48) Entfernung vor dem Roboter (d=100mm bzw. d=300mm) aufgestellt wurde, gewählt. Das Objekt wurde nun bzgl. der Ideallinie seitlich verschoben (siehe Abbildung 4.46). In 1,5m Entfernung vor dem Roboter befand sich eine Wand.

Wie zu erwarten war, sind Objekte ab einer gewissen Verschiebung von der Ideallinie im Bereich einiger cm nicht mehr sicher detektierbar. Eine Berührung dieser Objekte ist möglich, daher ist ein zusätzlicher, berührungsempfindlicher Sensor (Taste) als Kollisionserkennung erforderlich. Je näher sich ein Objekt vor dem Roboter befindet, desto stärker tritt dieser Effekt auf.

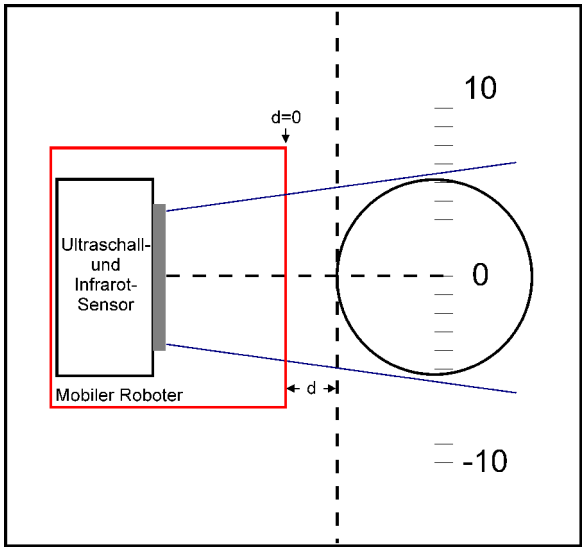

Abbildung 4.46: Messaufbau 'Seitliche Verschiebung'

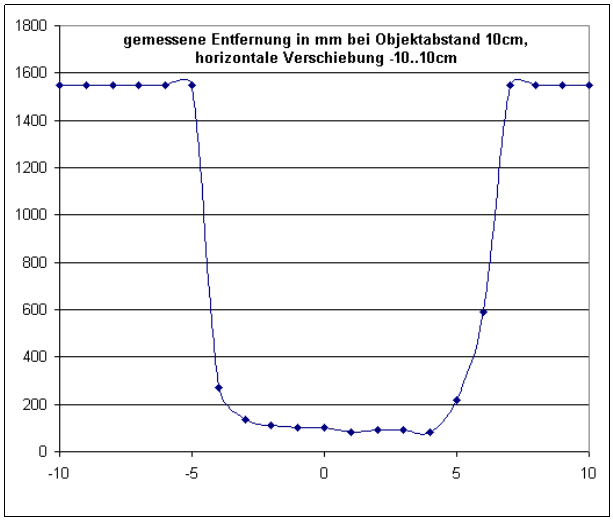

Abbildung 4.47: Objekt in 10cm Entfernung, seitlich verschoben

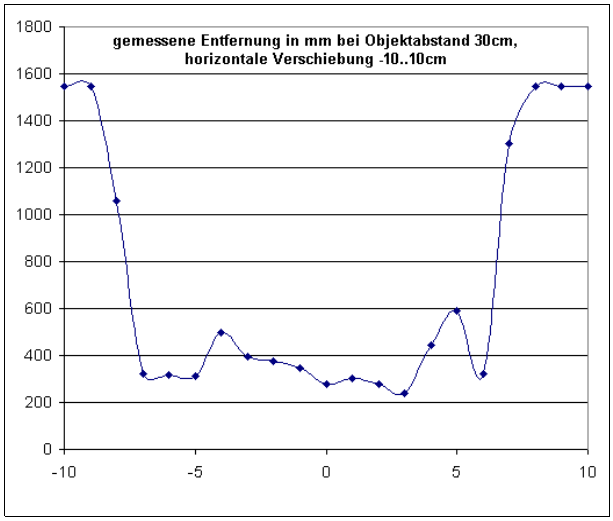

Abbildung 4.48: Objekt in 10cm Entfernung, seitlich verschoben

4.9.4 Praxistest Mobiler Roboter

Ein Praxistest mit einem einfachen Roboter (Abbildung 4.49), der mit zwei Motoren, dem Distanzmessmodul samt Ultraschall-, Infrarotsensor und zwei Tastern in der 'Stossstange' ausgerüstet war, demonstrierte die Funktion des Moduls unter realen Bedingungen. Ein einfaches C-Programm steuerte das simple Verhalten des Roboters:

- Geradeausbewegung bis Hindernis weniger als 20 cm entfernt
- Weiterfahrt mit verringerter Geschwindigkeit bis Hindernis weniger als 5 cm entfernt
- auf der Stelle nach links drehen, Entfernung messen, nach rechts drehen, Entfernung messen
- für grössere Entfernung entscheiden, in entsprechende Richtung drehen
- Lärmschutz: Wenn erkannte Entfernung <70 cm, auf Ultraschall verzichten (Kommando: Nur Infrarot an Distanzmessmodul)
- wiederholen

Der vom Distanzmessmodul gelieferte Entfernungswert beeinflusste das Verhalten des Roboters und wurde gleichzeitig auf dem LCD-Display des RCX in hexadezimaler Form ausgegeben.

Möbelstücke und Wände wurden bei diesem Test recht sicher erkannt.

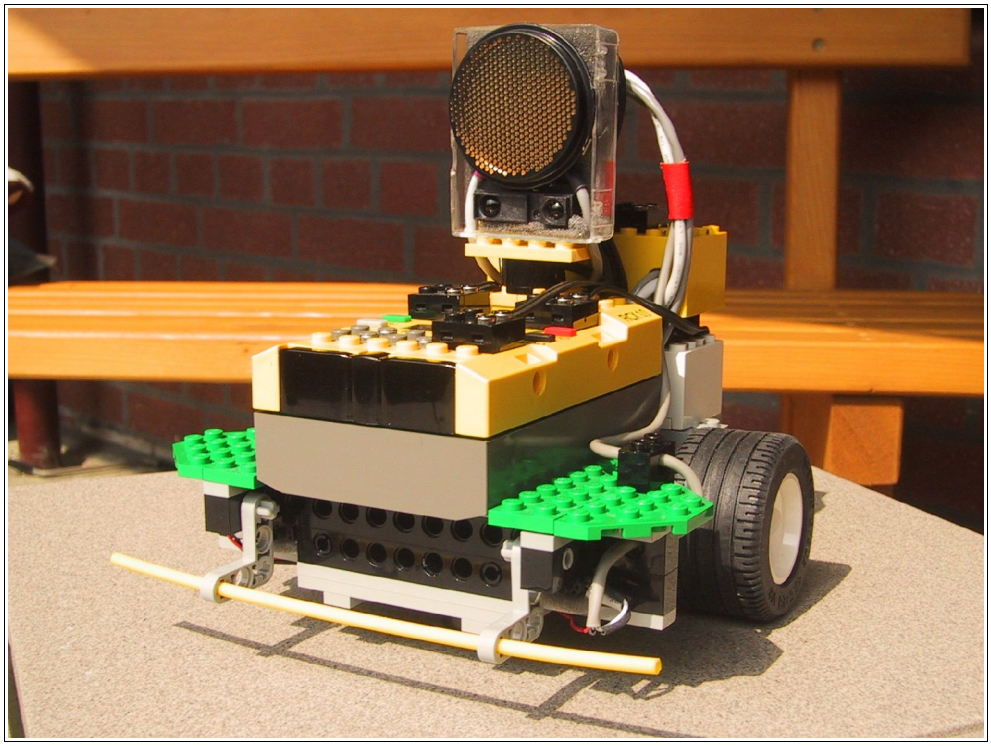

Abbildung 4.49: LEGO-Roboter, ausgerüstet mit Distanzmessmodul

In wenigen Fällen berührte der Roboter das Hindernis, hier signalisierten die Taster die Berührung. In den meisten Fällen fand jedoch eine berührungslose Erkennung statt.

4.9.5 Stromverbrauch

Der Stromverbrauch des Moduls wurde mit der Beschaltung nach Abbildung 4.50 ermittelt: Die an dem 10Ω-Widerstand abfallende Spannung wird über ein RC-Glied tiefpassgefiltert, die Spannung am Elektrolytkondensator mit einem Digitalvoltmeter nach einigen Sekunden gemessen (Zeitkonstante τ_{RC}=0,47s).

Die mittlere Stromaufnahme des Gerätes errechnet sich somit aus folgender Formel:

$$\bar{I} = \frac{\bar{U}(i(t))}{10}$$

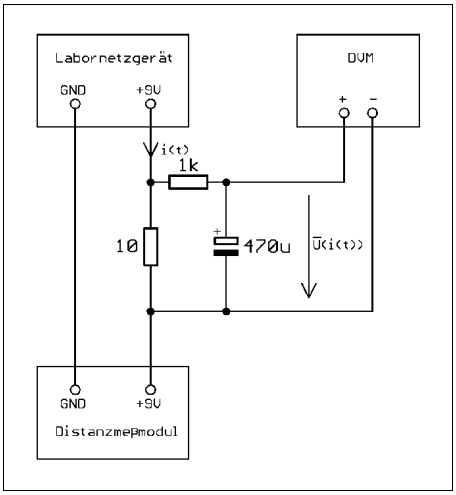

Abbildung 4.50: Ermittlung der mittleren Stromaufnahme

Bei einer Messfrequenz von etwa 8 Messungen pro Sekunde liegt die ermittelte mittlere Stromaufnahme bei 34mA. Mit dem Oszilloskop direkt am 10Ω-Widerstand gemessen ergaben sich Spitzenströme von etwa 200mA (hervorgerufen durch die Ultraschall-Impulserzeugung, <10ms).

5 Resümee

5.1 Ergebnis der Arbeit

Diese Arbeit hat gezeigt, dass es mit recht geringem Aufwand möglich ist, ein sehr kleines, für mobile Roboter konzipiertes Modul für die Distanzmessung zu bauen. Durch die Verwendung zweier nach unterschiedlichen Funktionsprinzipien arbeitender Sensoren und einer Mikrocontroller-basierten Messwert-Verarbeitung ergibt sich eine gute Genauigkeit in einem für die Anwendung in geschlossenen Räumen ausreichenden Distanzbereich.

5.2 Ausblick

Interessant wäre sicher die Verbindung mehrerer Module über die im normalen Betrieb am Lego-RCX Computer ungenutzte serielle Schnittstelle des Atmel-Microcontrollers. Denkbar wäre eine Art Token-Ring-Implementierung auf der seriellen Schnittstelle. Eines der Module müsste die Messwerte aller anderen sammeln und ggf. miteinander verknüpfen. Auf diese Weise könnte ein grösserer Bereich, unterteilt in tortenstückartige Sektoren, überwacht werden. Denkbar wäre auch, mehrere Sensoren in leicht unterschiedlichen Winkeln auf denselben Raumbereich zeigen zu lassen, um die starke Winkelabhängigkeit eines einzigen Ultraschall-Sensors auszugleichen.

Da das Modul mit der jetzigen Software auch in der Lage ist, auf der seriellen Schnittstelle die gemessene Entfernung auszugeben, eignet es sich auch für den Anschluss an Host-Systeme mit RS-232C-Schnittstellen. Durch die komplexere bidirektionale Kommunikation sind dann auch getriggerte Einzelmessungen und Eingriffe in das Verhalten des Moduls (wie z.B. Messbereichsumschaltung, Anzahl der Messungen pro Sekunde, etc.) einfach möglich.

Auf dem mit EXP im Schaltplan bezeichneten Anschluss sind noch zwei I/O-Leitungen des Mikrocontrollers frei verfügbar, diese können als digitale Ein- oder Ausgänge oder als analoge Eingänge benutzt werden. Eine Erweiterung der Funktionalität des Moduls durch weitere Sensoren oder Aktoren (z.B. Servo zum Drehen des Sensors) ist damit möglich.

Die Kommunikation mit dem Distanzmessmodul beeinflusst, wie in Kapitel 4.8 beschrieben, die Ganggenauigkeit der Systemuhr. Da die hervorgerufene Zeitdifferenz jedoch genau bekannt ist, kann in einer späteren

Version der Kommunikations-Routinen dieser Fehler durch Nachstellen der Uhr ausgeglichen werden.

Abbildungsverzeichnis

Literaturverzeichnis

Stone: *The Robots Are Coming*. online. - URL http://www.msnbc.-com/modules/newsweek/talk/032103_stone.htm - Zugriffsdatum: 21.05.2003

Husqvarna: *Introducing the (silent) robotic lawnmower*. online. - URL http://masseynews.massey.ac.nz/2002/news_release/15_08_02.html - Zugriffsdatum: 31.05.2003

Arkin: Ronald C. Arkin: *Behaviour Based Robotics*, 1998 - ISBN 0262011654

Singh, West: *Cyclone: A Laser Scanner For Mobile Robot Navigation*. online. - URL http://www.ri.cmu.edu/pub_files/pub3/singh_sanjiv_1991_2/singh_sanjiv_1991_2.pdf - Zugriffsdatum: 31.05.2003

Robotstore1: *IRPD Ver2.1 Infrared Proximity Detector*. online. - URL http://www.robotstore.com/download/3-573_IRPD_specs.pdf - Zugriffsdatum: 31.05.2003

SharpIR: *Distance Measuring Sensors*. online. - URL http://www.s-harpsma.com/sma/Products/Prod-frame/opto.htm?main=http://www.sharpsma.com/sma/products/opto/laser_diode.htm - Zugriffsdatum: 29.06.2003

Robotstore2: *Ultrasonic Owl Scanner - Sonar Sensor Kit*. online. - URL http://www.robotstore.com/catalog/display.asp?pid=217 - Zugriffsdatum: 31.05.2003

SRF08: *SRF08 Ultra sonic range finder - a little History*. online. - URL http://www.robot-electronics.co.uk/htm/srf08history.shtml - Zugriffsdatum: 31.05.2003

Sick: *PLS Proximity Laser Scanner - Installation and Operation Manual.* online. - URL http://www.sickoptic.com/Publish/docroot/Manual(s)/PLS%20Operating%20Manual.pdf - Zugriffsdatum: 22.06.2003

KMZ51: *Philips KMZ51 Magnetic field sensor datasheet*. online. - URL http://www-us.semiconductors.philips.com/acrobat/datasheets/KMZ51_3.pdf - Zugriffsdatum: 31.05.2003

GPS: *GPS System Provider - Laipac Technology Inc.*. online. - URL http://www.laipac.com - Zugriffsdatum: 31.05.2003

Radar: *Was ist eigentlich Radar? - Das herkömmliche Radar*. online. - URL http://www.nv.et-inf.uni-siegen.de/pb2/school/page_1.htm - Zugriffsdatum: 31.05.2003

KMY24: *Dopplermodul KMY24: Bewegungsmelder erkennt Richtung*. online. - URL http://www.produktinfo.conrad.com/daten-

blaetter/175000-199999/182613-da-01-de-kmy24.pdf - Zugriffsdatum: 31.05.2003
Physik: Jung, Walter: *Das Abitur-Wissen PHYSIK*, 1983 - ISBN 3-596-24542-7
Schallgeschwindigkeit: *Die Schallgeschwindigkeit, die Temperatur*. online. - URL http://www.sengpielaudio.co.uk/DieSchallgeschwindigkeitLuftdruck - Zugriffsdatum: 13.08.2003
Breheny: *The Sharp GP2D02 IR Distance Measuring Sensor*. online. - URL http://www.wirz.com/info/gp2d02/ - Zugriffsdatum: 01.06.2003
Microchip: *Homepage des Hersteller Microchip*. online. - URL http://www.microchip.com - Zugriffsdatum: 13.06.2003
ATMEL: *Homepage des Herstellers Atmel*. online. - URL http://www.atmel.com - Zugriffsdatum: 29.06.2003
PIC: *Datenblatt PIC16F870*. online. - URL http://www.microchip.com/download/lit/pline/picmicro/families/16f87x/30569b.pdf - Zugriffsdatum: 11.06.2003
Acroname: *Polaroid 6500 Ranging Module*. online. - URL http://www.acroname.com/robotics/parts/R11-6500.html - Zugriffsdatum: 08.06.2003
MAX5541: *Datenblatt MAX5541*. online. - URL http://pdfserv.maxim-ic.com/arpdf/MAX5541.pdf - Zugriffsdatum: 11.06.2003
MAX8881: *Datenblatt MAX8881*. online. - URL http://pdfserv.maxim-ic.com/arpdf/MAX8880-MAX8881.pdf - Zugriffsdatum: 11.06.2003
EAGLE: *CadSoft Online*. online. - URL http://www.cadsoft.de - Zugriffsdatum: 11.06.2003
kernel: *Introduction to the legOS kernel*. online. - URL http://legos.sourceforge.net/docs/kerneldoc.pdf - Zugriffsdatum: 11.08.2003
sys/irq.h: *brickOS - include/lnp/sys/irq.h File Reference*. online. - URL http://brickos.sourceforge.net/docs/APIs/html-kern/irq_8h.html - Zugriffsdatum: 10.08.2003
kernel/systime.c: . online. - URL http://brickos.sourceforge.net/docs/APIs/html-kern/systime_8c-source.html#l00270 - Zugriffsdatum: 10.08.2003
Hitachi H8: *Hitachi Single-Chip Microcomputer H8... Hardware Manual*. online. - URL http://legolab.daimi.au.dk/DigitalControl.dir/RCX/Manual.dir/H8Hardware.pdf - Zugriffsdatum: 08.06.2003

A.Schaltplan

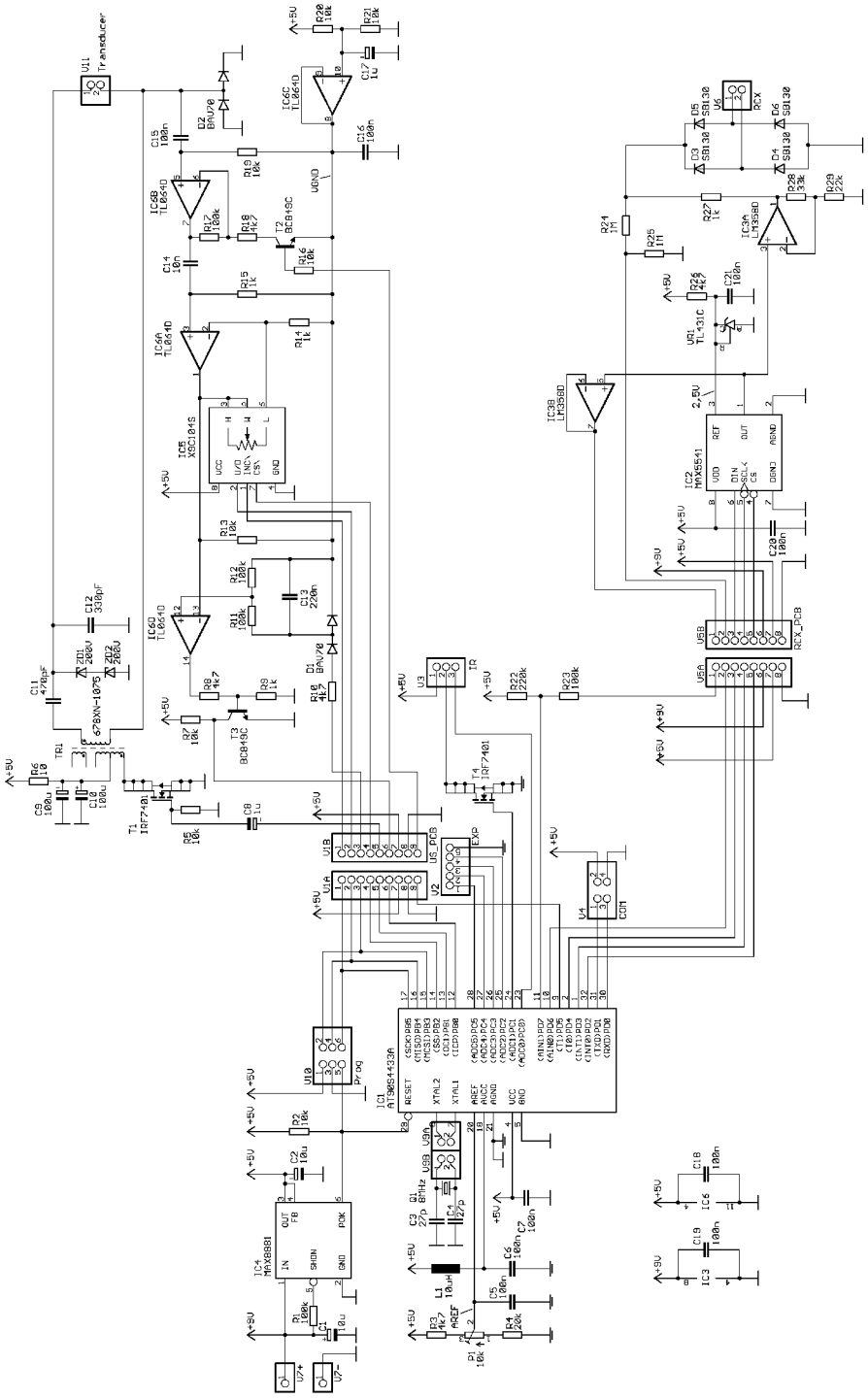

Abbildung 6.1: Schaltplan 'Distanzmessmodul'

B.Platinenlayout und Bestückungsplan

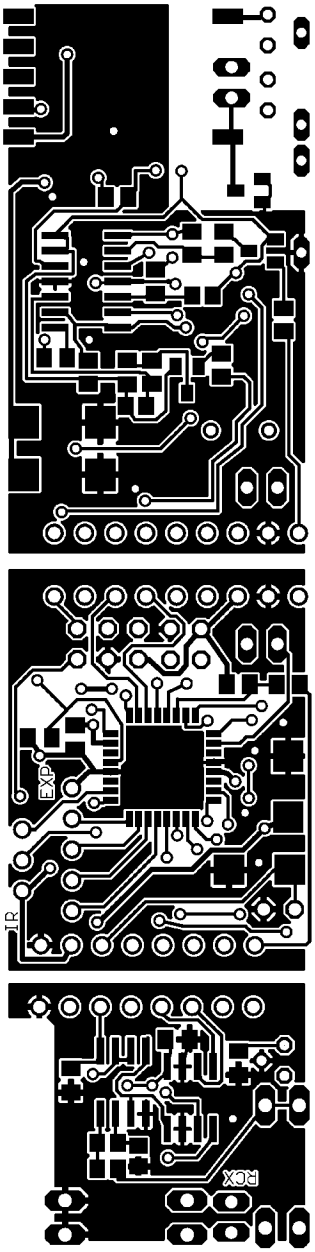

Abbildung 7.1: Layout Oberseite (Top)

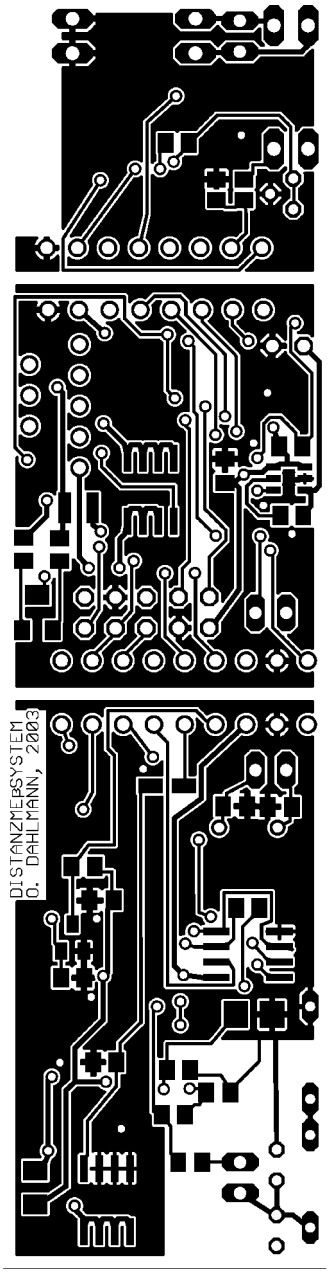

Abbildung 7.2: Layout Unterseite (Bottom)

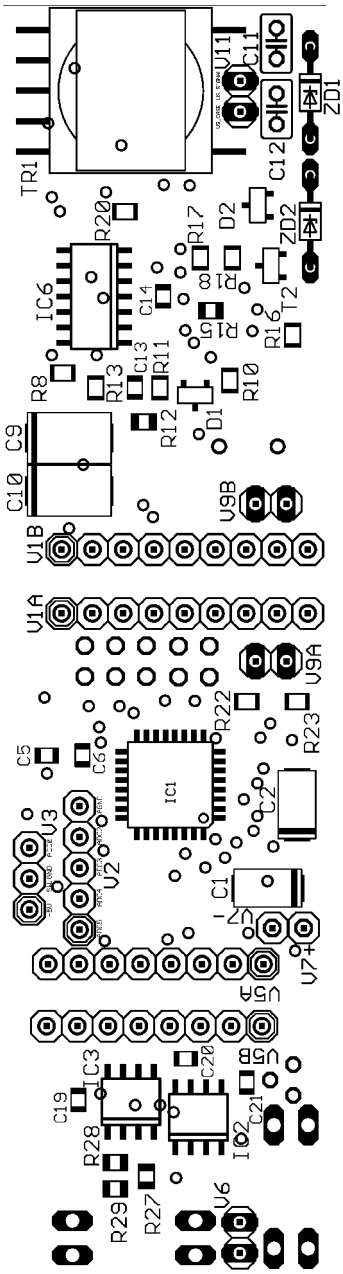

Abbildung 7.3: Bestückungsplan Oberseite (Top)

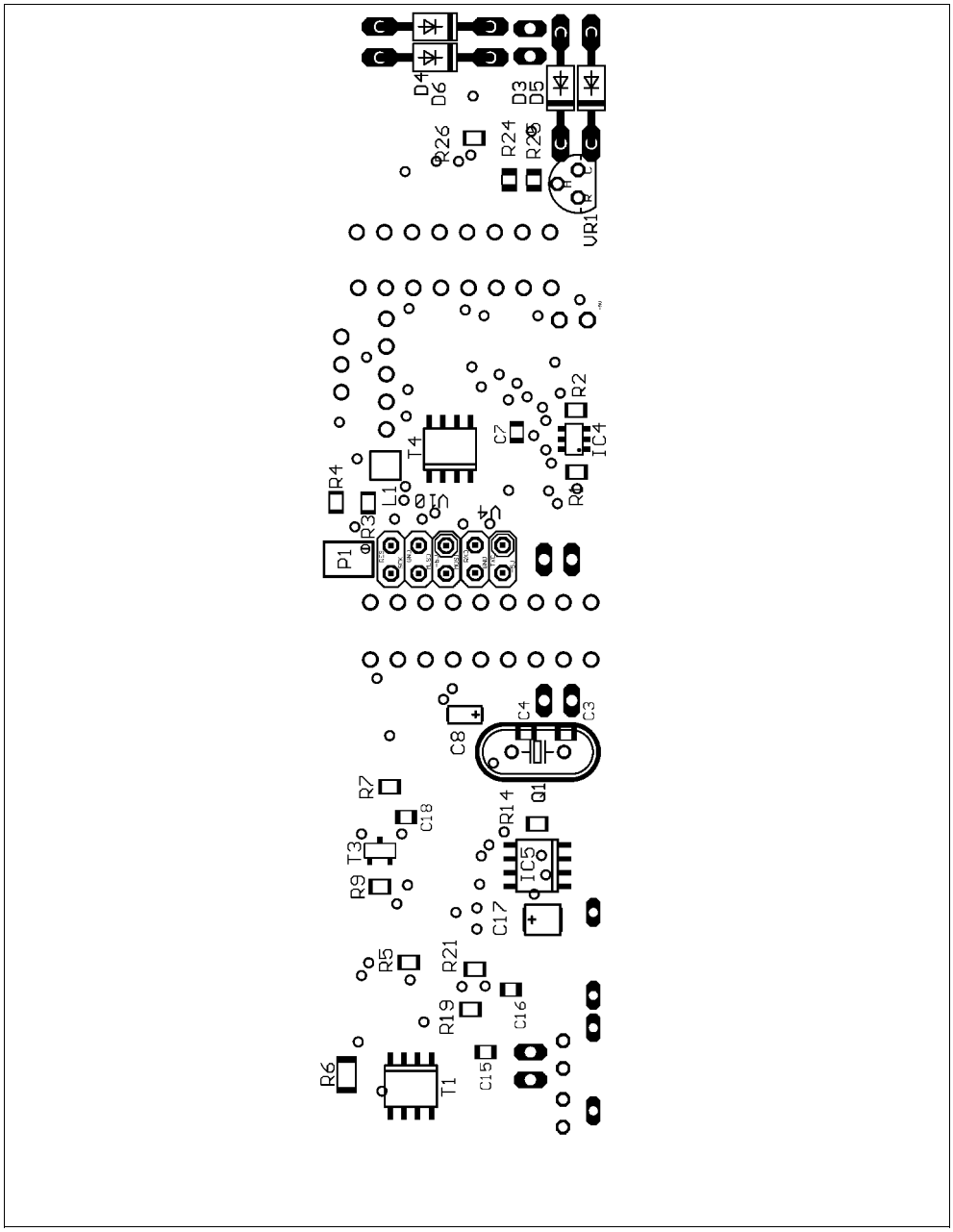

Abbildung 7.4: Bestückungsplan Unterseite (Bottom)

C.Inbetriebnahme

Der Mikrocontroller wird, nachdem das Modul fertig aufgebaut wurde, mit einem handelsüblichen ISP-Adapter programmiert. Im konkreten Fall wurde ein AVR-ISP-Adapter für den Anschluss an den Parallelport des PC's mit der frei im Internet erhältlichen Software PonyProg verwendet.

Der Infrarot-Sensor sei so am Roboter montiert, das ein direkt vor dem Roboter befindliches Hindernis (dies sei Entfernung d=0), das weiter entfernt wird, ein monoton fallendes Signal am analogen Sensor-Ausgang zur Folge hat (siehe Beschreibung des Sensors). Auf diese Weise ist ein Messwert eindeutig einer bestimmten Entfernung zuordbar.

Ein Terminalprogramm dient zur Anzeige der Messwerte, es werden Rohdaten angezeigt (Taste 'r' am Terminal drücken). Ein gut reflektierendes Objekt wird im rechten Winkel bei d=0 positioniert. Nun die Referenzspannung des A/D-Wandlers mittels des Potentiometers P1 so einstellen, das etwa 900 als Messwert angezeigt wird. Auf diese Weise bleiben etwa 10% des Wertebereichs des A/D-Wandlers (10Bit, 0..1023) als 'Headroom' übrig.

Als nächstes wird die Funktion „Linearisierung“ aufgerufen (Taste 'l' drücken). Auf dem Bildschirm des Terminals werden jetzt Entfernungen (in mm) ausgegeben.

Nun ein Hindernis im rechten Winkel in der angegebenen Entfernung positionieren und eine Taste am Terminal drücken. Es werden nun 5 Ultraschall- und 32 Infrarot-Messungen vorgenommen und auf dem Terminal-Bildschirm ausgegeben. Diese Ausgabe in eine ASCII-Datei umlenken (Hyperterminal -> Capture). Aus den Mittelwerten dieser Messreihen wird eine Linearisierungstabelle gebildet, die im Flash-Speicher des Atmel-Controllers, zusammen mit der Firmware, abgelegt wird.

Wie in Kapitel 4.7.2 beschrieben ist eine Annäherung der Messwerte durch ein Polynom den tatsächlichen Messwerten zum Aufbau der Linearisierungstabelle vorzuziehen.

Zur Ermittlung der geometrischen Mittelwerte der Messreihen bietet sich z.B. 'OpenOffice Calc' an, das Regressionspolynom kann mit der Freeware 'Matheass' erstellt werden.

Für eine korrekte Funktion des Ultraschallsensors ist die Anpassung der im Quelltext änderbaren Konstanten nötig: Anzugeben sind:

- US_Offset [mm], der Abstand des Sensors von der Nulllinie d=0,
- US_Totbereich [mm], die Entfernung von d=0 bis zu der Position, an der der Ultraschall-Sensor beginnt, sinnvolle Werte zu liefern
- in Sonderfällen US_Faktor [m/s], die halbe Schallgeschwindigkeit (änderbar, da temperaturabhängig)

D.Sourcecode (RCX)

Auf den folgenden Seiten folgt die Software für den RCX, die es ermöglicht, Kommandos über die Sensor-Schnittstelle an einen angeschlossenen Sensor zu senden.

Library 'distanz.h', Kommunikationsroutinen für den RCX

```
/*! \file   include/distanz.h
    \brief  Interface: Kommunikationsroutinen für das Distanzmessmodul
    \author Oliver Dahlmann <ollinux@gmx.net>
 */

#ifndef __distanz_h__
#define __distanz_h__

#ifdef  __cplusplus
extern "C" {
#endif

#include <config.h>
#include <sys/h8.h>
#include <sys/bitops.h>

////////////////////////////////////////////////////////////////////////
//
// Functions
//
////////////////////////////////////////////////////////////////////////

extern inline void WatchDog_Sperren()
{
  __asm__ __volatile__("\tpush r0\n":::"cc");
  __asm__ __volatile__("\tmov.b #0xA5,r0h\n":::"cc");
  __asm__ __volatile__("\tmov.b @0xFFA8,r0l\n":::"cc");
  __asm__ __volatile__("\tand.b #0b10111111,r0l\n":::"cc");
  __asm__ __volatile__("\tmov.w r0,@0xFFA8\n":::"cc");
  __asm__ __volatile__("\tpop r0\n":::"cc");
}

extern inline void WatchDog_Aktivieren()
{
  __asm__ __volatile__("\tpush r0\n":::"cc");
  __asm__ __volatile__("\tmov.b #0xA5,r0h\n":::"cc");
  __asm__ __volatile__("\tmov.b @0xFFA8,r0l\n":::"cc");
  __asm__ __volatile__("\tor.b #0b01000000,r0l\n":::"cc");
  __asm__ __volatile__("\tmov.w r0,@0xFFA8\n":::"cc");
  __asm__ __volatile__("\tpop r0\n":::"cc");
}

extern inline void SendCommand(int command, volatile unsigned *sensor)
{
  int verz;

  disable_irqs();

  WatchDog_Sperren();               //Watchdog wird auf Intervall-Timer
umgeschaltet, somit keine NMI's mehr...

  //übermittelten Sensor-Eingang auf aktiv schalten
  if (sensor == &SENSOR_3)
    bit_set(&PORT6, 0);
  else if (sensor == &SENSOR_2)
```

```
    bit_set(&PORT6, 1);
  else if (sensor == &SENSOR_1)
    bit_set(&PORT6, 2);

  // command == 1 -> 24ms HIGH
  // command == 2 -> 40ms HIGH
  // command == 3 -> 56ms HIGH
  // command == 4 -> 72ms HIGH

  //Verzögerungsschleife mit NOP's
  for (verz=0;verz<(1970+1310*(command-1));verz++)
    __asm__ __volatile__("\tnop\n":::"cc");

  //übermittelten Sensor-Eingang wieder auf passiv schalten
  if (sensor == &SENSOR_3)
    bit_clear(&PORT6, 0);
  else if (sensor == &SENSOR_2)
    bit_clear(&PORT6, 1);
  else if (sensor == &SENSOR_1)
    bit_clear(&PORT6, 2);

  WatchDog_Aktivieren();             //Watchdog wieder auf NMI umstellen

  enable_irqs();
}
```

E.Sourcecode (Atmel AT90S4433)

Auf den folgenden Seiten folgt der Sourcecode des Atmel-AT90S4433-Mikrocontrollers.

Distanz_Main.asm

```
; ***************************************************************************
; Projektname:      Distanz.aps
; Diese Datei:      Distanz_Main.asm
; Beschreibung:     Distanzmesssystem für mobile Roboter
; Autor:            Oliver Dahlmann
; Version:          1.0
; ***************************************************************************

.NOLIST
.INCLUDE "4433def.inc"
.LIST

; I/O Port D
; ** verwendete Bits
.EQU    MAX_CS          =       2       ; OUT Pin4 MAX554x /CS (Chip Select, active low)
.EQU    MAX_SCLK        =       3       ; OUT Pin5 MAX554x /SCLK (Serial Clock, steigende Flanke übernimmt Daten-Bit)
.EQU    MAX_DIN         =       4       ; OUT Pin6 MAX554x DIN (Data In)
.EQU    GAIN_20         =       5       ; Der Transistor T2 kann die Eingangsverstärkung des Ultraschallempfängers von 1x auf 20x umschalten
; ** ... an Port
.EQU    MAX_PORT        =       PORTD
.EQU    GAIN_PORT       =       PORTD

; I/O Port B
; ** verwendete Bits ...
.EQU    USN_REC         =       0       ; IN  Komparator mit Empfangssignal
.EQU    USN_XMIT        =       1       ; OUT Puls-Erzeugung (MOSFET)
.EQU    POT_CS0         =       2       ; _CS-Signal für Digital-Poti 0
.EQU    USN_TRESHOLD    =       3       ; OUT Trigger für Kondensator-Entladekurve für zeitabhängigen Treshold des US Komparators
.EQU    POT_UD          =       4       ; U/_D-Signal für Digital-Poti(s)
.EQU    POT_INC         =       5       ; _INC-Signal für Digital-Poti(s)
; ** ... an Port
.EQU    USN_PORT        =       PORTB   ; mit diesem Port ist der Polaroid-Sensor verbunden (Ausgang)
.EQU    USN_PIN         =       PINB    ; mit diesem Port ist der Polaroid-Sensor verbunden (Eingang)
.EQU    POT_PORT        =       PORTB   ; mit diesem Port ist das Digital-Poti verbunden

; I/O Port C
; ** verwendete Bits ...
.EQU    PWR_IRSENSOR    =       1       ; OUT MOSFET, wenn 1 dann Infrarot-Sensor an
.EQU    BUMPER_LEFT     =       4
.EQU    BUMPER_RIGHT    =       5
; ** ... an Port
.EQU    PWR_PORT        =       PORTC   ; mit diesem Port sind die MOSFET-Treiber für die Spannungsversorgung IR/US verbunden
.EQU    BUMPER_PORT     =       PORTC

; ** Korrektur-Offset/Quotient für Ultraschall-Umrechnung
.EQU    US_Faktor       =       174     ; ergibt sich aus Schallgeschwindigkeit ([mm/ms] / 2)
.EQU    US_Offset       =       80      ; abhängig von Sensorposition (Entfernung von d=0 in mm)
.EQU    US_Totbereich   =       210     ; auch abhängig von Sensorposition, Mindest-Entfernung von d=0, die gerade gemessen werden kann
```

```
; ** Linearisierungstabelle erzeugen
.EQU    Lin_Schrittweite    =       10      ; 10mm Schrittweite
.EQU    Lin_MaxEntfernung   =       800     ; 80cm Maximal-Entfernung

; ** Die Bits der Speicherzelle 'ModusFlags'
.EQU    MOD_Bereich             = 0     ; Bit0, 0=5m, 1=1m
.EQU    MOD_Frequenz            = 1     ; Bit1, 0=8/s, 1=2/s
.EQU    MOD_Ultraschall         = 2     ; Bit2, 0=Ultraschall aus, 1=Ultraschall an
.EQU    MOD_WerteSeriell        = 3     ; Bit3, 0=keine serielle Ausgabe, 1=serielle Ausgabe
.EQU    MOD_DebugInfo           = 4     ; Bit4, 0=wenig Informationen, 1=viele Informationen (verbose)
.EQU    MOD_Rohdaten            = 5     ; Bit5, 0=mm, 1=Rohdaten
.EQU    MOD_USFilter            = 6     ; Bit6, wenn gesetzt, wird der Ultraschall-Messwert gefiltert
.EQU    MOD_IRFilter            = 7     ; Bit7, wenn gesetzt, wird der Infrarot-Messwert gefiltert

.ORG 0000
; ** Tabelle mit Sprunganweisungen auf die Interrupt-Handler
        rjmp    RESET                   ; Reset Handler
        reti                            ; IRQ0 Handler
        reti                            ; IRQ1 Handler
        rjmp    USN_PacketRecvd         ; Timer1 Capture Handler, Laufzeit Ultraschall auswerten
        reti                            ; Timer1 Compare Match Handler
        rjmp    USN_NoPacketRecvd       ; Timer1 Overflow Handler, kein Echo empfangen
        rjmp    Sysclk                  ; Timer0 Overflow Handler
        reti                            ; SPI Transfer Complete Handler
        rjmp    UART8_RXC_Handler       ; UART RX Complete Handler
        reti                            ; UART Empty Handler
        rjmp    UART8_TXC_Handler       ; UART TX Complete Handler
        rjmp    ADCC                    ; AD Conversion Complete Handler
        reti                            ; EEPROM Ready Handler
        reti                            ; Analog Comparator Handler

RESET:
; ** Stackpointer-Initialisierung
        ldi     r16,LOW(RAMEND)
        out     SP, r16                 ; Stackpointer initialisieren

; ** Datenrichtungsregister Initialisierung
        ldi     r16,0b00111110
        out     DDRB,r16                ; PortB: PB0 ist Eingang (ECHO)
        rcall   USN_Init                ; Jetzt die Ultraschall-Elektronik initialisieren

; ** Serieller Port (UART) Initialisierung
        ldi     r16,12                  ; Baudrate auf 38.400 Baud (@8MHz)
        rcall   UART8_Init

; ** Datenrichtungsregister Initialisierung
        ldi     r16,0b00001110          ; PortC: PC0 ist Analog-In, PC4,5 Bumper Left, Right
        out     DDRC,r16

        ldi     r16,0b00111100          ; PD7+PD6 Analog-Komparator, PD2..PD4 DAC, PD0+PD1 UART
        out     DDRD,r16
```

```
        ldi     r16,(1<<CS00)+(1<<CS01)
        out     TCCR0,r16               ; Timer0 CLK/64, Interrupt für DAC Ausgabe
        ldi     r16,0                   ; Timer0 bei $00 starten
        out     TCNT0,r16

        ldi     r16,0
        out     TCCR1A,r16              ; Timer1 Output_Compare aus, PWM aus

        ldi     r16,(1<<CS11)+(1<<ICES1)    ; steigende Flanke triggert Timer1, CLK/8
        out     TCCR1B,r16              ; Timer1 Vorteiler auf CK/8

        ldi     r16,(1<<TOIE0)          ; Timer0 Overflow (DAC) aktivieren
        out     TIMSK,r16

        ldi     r16,(1<<ADEN)+(1<<ADPS2)+(1<<ADPS1)+(1<<ADIE)
        out     ADCSR,r16               ; AD-Clock = CLK/64 = 125kHz -> tconv = 112usec (14cycles), IRQ an

        ldi     r16,0                   ; Eingang PA0 wählen
        out     ADMUX,r16

        ldi     r16,(1<<SE)
        out     MCUCR,r16               ; SM = 0 -> IdleMode @Sleep Instruction, SE=1 -> Sleep Instruction enabled

        ldi     r16,0                   ; Timer1 bei $0000 starten
        out     TCNT1H,r16
        out     TCNT1L,r16

        ldi     r16,(1<<MOD_Ultraschall)
        sts     Modusflags,r16

        rcall   MAX554x_Init            ; DAC initialisieren
        rcall   RCX_Init                ; Variablen für RCX-Empfang initialisieren
        sei                             ; Bit7 SREG setzen -> Interrupts erlauben

; ************************************** MAIN ***********************************************
MAIN:
        rcall   RCX_Befehl_ausfuehren   ; wenn ein RCX-Befehl empfangen wurde, diesen ausführen
        rcall   SerialRXD               ; Wurde vom Terminal ein Befehl empfangen (RS-232)?

        rcall   Messfrequenz            ; abhängig von gewünschter Messfrequenz verzögern

        rcall   Ultraschall             ; Ultraschall-Messung abhängig vom Modus, Wert filtern
        rcall   Infrarot                ; Infrarot-Messung, Wert filtern

        rcall   Ausgabe_US_Seriell      ; abhängig von MOD_WerteSeriell Ultraschall-Messwert ausgeben
        rcall   Ausgabe_IR_Seriell      ; abhängig von MOD_WerteSeriell Infrarot-Messwert ausgeben

        rcall   MesswertVerarbeitung    ; Errechnung eines Entfernungswertes aus den gemessenen Werten
                                        ; Ergebnis auf DAC ausgeben, erweiterte Informationen seriell ausgeben
        rjmp    MAIN
; *******************************************************************************************
```

```
Ultraschall:
        lds     r16,Modusflags
        sbrs    r16,MOD_Ultraschall
        rjmp    KeineUltraschallMessung
        rcall   USN_SendPacket          ; Starte Ultraschall-Messung, Rückgabe des Entfernungs-Wertes in mm
        rcall   USN_WaitMeasure         ; Warte auf Beendigung der Messung
Ultraschall_Filtern:
        rcall   USN_Filter              ; abhängig von MOD_USFilter wird der Timerwert gefiltert
        ret
KeineUltraschallMessung:
; ** Da keine Messung durchgeführt werden soll, maximale Entfernung als Ultraschallwert zurückgeben
        ser     r16
        sts     USN_RawLo,r16
        sts     USN_RawHi,r16
        rjmp    Ultraschall_Filtern

Infrarot:
        sbi     PWR_PORT,PWR_IRSENSOR ; Infrarot-Sensor einschalten
        rcall   Delay25ms
        rcall   Delay25ms               ; warte bis SHARP-IR-Sensor stabil
        rcall   Delay25ms               ; (min. 42ms bis zur Ausgabe eines analogen Wertes)
        rcall   SIR_Messung8            ; 8 Wandlungen durchführen und Mittelwert bilden
        rcall   SIR_Filter              ; gewichteter Mittelwert abhängig von Wahrscheinlichkeit
        cbi     PWR_PORT,PWR_IRSENSOR ; Infrarot-Sensor ausschalten
        ret

Messfrequenz:
        lds     r16,Modusflags
        sbrs    r16,MOD_Frequenz
        rjmp    Messfrequenz8
Messfrequenz2:
        rcall   Delay100ms
        rcall   Delay100ms
        rcall   Delay100ms
        rcall   Delay100ms
Messfrequenz8:
        rcall   Delay25ms               ; etwa 10 Messwerte pro Sekunde
        ret

Ausgabe_US_Seriell:
        lds     r16,Modusflags
        sbrs    r16,MOD_Rohdaten
        rcall   USN_mm                  ; Den Timer-Wert in mm umrechnen, wenn nicht Rohdaten gefordert
; ** und hier wird der US-Wert seriell ausgegeben, wenn MOD_WerteSeriell=1
        lds     r16,Modusflags
        sbrs    r16,MOD_WerteSeriell    ; sollen der US- und der IR-Wert seriell ausgegeben werden?
        rjmp    US_WertAusgegeben

        ldi     zl,LOW(NewLine_String<<1)
        ldi     zh,HIGH(NewLine_String<<1)
        rcall   UART8_SendPGMString     ; 'Neue Zeile' senden (CR+LF)
```

```
            ldi     zl,LOW(UltraschallString<<1)
            ldi     zh,HIGH(UltraschallString<<1)
            rcall   UART8_SendPGMString     ; "Ultra:" senden

            ldi     zl,LOW(SerString)
            ldi     zh,HIGH(SerString)
            push    zl
            push    zh
            lds     r18,USN_ValLo
            lds     r19,USN_ValHi
            rcall   uWordToDecNull          ; in String umwandeln (Dezimal, 5 stellig, null-terminiert)
            pop     zh
            pop     zl
            rcall   UART8_SendRAMString     ; Ultraschall-Wert (Dezimal, 5stellig) senden

            ldi     zl,LOW(Divider_String<<1)
            ldi     zh,HIGH(Divider_String<<1)
            rcall   UART8_SendPGMString     ; Trennzeichen senden
US_WertAusgegeben:
            ret

Ausgabe_IR_Seriell:
            lds     r16,Modusflags
            sbrs    r16,MOD_Rohdaten
            rcall   SIR_mm                  ; Den Infrarot-AD-Wert in mm umrechnen
; ** und hier wird der IR-Wert seriell ausgegeben
            lds     r16,Modusflags
            sbrs    r16,MOD_WerteSeriell
            rjmp    IR_WertAusgegeben
            ldi     zl,LOW(InfrarotString<<1)
            ldi     zH,HIGH(InfrarotString<<1)
            rcall   UART8_SendPGMString     ; "Infra:" senden
            ldi     zl,LOW(SerString)
            ldi     zH,HIGH(SerString)
            push    zl
            push    zh

            lds     r19,SIR_RawHi
            lds     r18,SIR_RawLo           ; hierin steht der Wert in mm (in SIR_ValHi,Lo steht der nicht linearisierte Wert)
            rcall   uWordToDecNull

            pop     zh
            pop     zl
            rcall   UART8_SendRAMString     ; Infrarot-Wert (Dezimal, 5stellig) senden
IR_WertAusgegeben:
            ret

MesswertVerarbeitung:
; ** Die in mm umgerechneten US- und IR-Messwerte auf Konsistenz prüfen und Entfernungswert ermitteln
            lds     r19,SIR_RawHi
            lds     r18,SIR_RawLo           ; hierin steht der Wert in mm (in SIR_ValHi,Lo steht der nicht linearisierte Wert)
```

```
		lds	r21,USN_ValHi
		lds	r20,USN_ValLo

		rcall	SensorVerarbeitung		; danach Rückgabewert in mm in r23:r22, Differenz in r17:r16, Status in r24

		push	r17
		push	r16				; Differenz auf Stack sichern, wird von DebugInfo ausgegeben

		mov	r17,r23
		mov	r16,r22

		lds	r18,Modusflags
		sbrs	r18,MOD_Bereich
		rjmp	MV_Bereich_5m
;** Bereich 1m
		ldi	r19,HIGH(1000)
		ldi	r18,LOW(1000)			; Skalierung des DAC-Wertebereichs auf 1m
		rjmp	MV_RCX
MV_Bereich_5m:
		ldi	r19,HIGH(5000)
		ldi	r18,LOW(5000)			; Skalierung des DAC-Wertebereichs auf 5m
MV_RCX:
		rcall	RCX_Output			; skalieren und verschieben des mm-Wertes für den D/A-Wandler
		rcall	MAX554x_Set_RAM			; schreibe die Werte ins RAM, den Rest erledigt der Timer0 Interrupt

; ** Und hier wird das Ergebnis der Filterung mit Status-Informationen ausgegeben
		lds	r16,Modusflags
		sbrs	r16,MOD_DebugInfo
		rjmp	DebugInfo_KeineAusgabe
		sbrc	r16,MOD_WerteSeriell
		rjmp	DebugInfo_Ausgabe
DebugInfo_KeineAusgabe:
		pop	r16
		pop	r16				; Die Differenz vom Stack entfernen
		ret
DebugInfo_Ausgabe:
		cpi	r24,VVA_STATUS_IR
		breq	DebugInfo_Ausgabe_Infrarot
		cpi	r24,VVA_STATUS_US
		breq	DebugInfo_Ausgabe_Ultraschall
		cpi	r24,VVA_STATUS_BUMPER
		breq	DebugInfo_Ausgabe_Bumper
DebugInfo_Ausgabe_Fehler:
; ** Als Status kam keine der drei erlaubten Meldungen zurück, daher Fehler ausgeben, resetten
		ldi	zl,LOW(InfoStringFehler<<1)
		ldi	zh,HIGH(InfoStringFehler<<1)
		rcall	UART8_SendPGMString		; "Fehler" senden
		rcall	Delay100ms
		rjmp	RESET
DebugInfo_Ausgabe_Ultraschall:
		ldi	zl,LOW(InfoStringUS<<1)
		ldi	zh,HIGH(InfoStringUS<<1)
```

```
        rjmp    DebugInfo_Ausgabe_Wert
DebugInfo_Ausgabe_Bumper:
        ldi     zl,LOW(InfoStringBumper<<1)
        ldi     zh,HIGH(InfoStringBumper<<1)
        rjmp    DebugInfo_Ausgabe_Wert
DebugInfo_Ausgabe_Infrarot:
        ldi     zl,LOW(InfoStringIR<<1)
        ldi     zh,HIGH(InfoStringIR<<1)
DebugInfo_Ausgabe_Wert:
        rcall   UART8_SendPGMString     ; "IR" senden
        ldi     zl,LOW(SerString)
        ldi     zh,HIGH(SerString)
        mov     r18,r22
        mov     r19,r23                 ; Wert in mm
        rcall   uWordToDecNull          ; in String umwandeln (Dezimal, 5 stellig, null-terminiert)

        ldi     zl,LOW(SerString)
        ldi     zh,HIGH(SerString)
        rcall   UART8_SendRAMString     ; Wert seriell senden
        ldi     zl,LOW(Divider_String<<1)
        ldi     zh,HIGH(Divider_String<<1)
        rcall   UART8_SendPGMString     ; Trennzeichen senden
        ldi     zl,LOW(InfoStringDiff<<1)
        ldi     zh,HIGH(InfoStringDiff<<1)
        rcall   UART8_SendPGMString     ; "Diff" senden

        ldi     zl,LOW(SerString)
        ldi     zh,HIGH(SerString)
        pop     r18
        pop     r19                     ; auf dem Stack steht die Differenz der beiden Messwerte
        rcall   uWordToDecNull          ; in String umwandeln (Dezimal, 5 stellig, null-terminiert)

        ldi     zl,LOW(SerString)
        ldi     zh,HIGH(SerString)
        rcall   UART8_SendRAMString     ; Wert seriell senden
        ldi     zl,LOW(Divider_String<<1)
        ldi     zh,HIGH(Divider_String<<1)
        rcall   UART8_SendPGMString     ; Trennzeichen senden
        rjmp    DebugInfo_Ausgabe_Ende
DebugInfo_Ausgabe_Ende:
        ret

SerialRXD:
; ** Wurde was auf dem seriellen Port empfangen?
        ldi     zl,LOW(RS232)           ; die Adresse der Funktion, die das empfangene Zeichen
        ldi     zh,HIGH(RS232)          ; verarbeitet
        rcall   UART8_GetByte
        ret

Menue_UltraFilter:
        ldi     zl,LOW(UltraFilterString <<1)
        ldi     zh,HIGH(UltraFilterString <<1)
```

```
        rcall   Menue_Ausgabe
        lds     r16,ModusFlags
        sbrs    r16,MOD_USFilter
        rjmp    Menue_FlagAus
Menue_FlagAn:
        ldi     zl,LOW(AnString<<1)
        ldi     zh,HIGH(AnString<<1)
        rjmp    Menue_FlagAusgabe
Menue_FlagAus:
        ldi     zl,LOW(AusString<<1)
        ldi     zh,HIGH(AusString<<1)
        rjmp    Menue_FlagAusgabe

Menue_Ausgabe:
        ldi     r17,13
        rcall   UART8_SendByte
        ldi     r17,10
        rcall   UART8_SendByte
        rcall   UART8_SendPGMString
        ret

Menue_InfraFilter:
        ldi     zl,LOW(InfraFilterString <<1)
        ldi     zh,HIGH(InfraFilterString <<1)
        rcall   Menue_Ausgabe
        lds     r16,ModusFlags
        sbrs    r16,MOD_IRFilter
        rjmp    Menue_FlagAus
        rjmp    Menue_FlagAn

Menue_Debug:
        ldi     zl,LOW(DebugString <<1)
        ldi     zh,HIGH(DebugString <<1)
        rcall   Menue_Ausgabe
        lds     r16,ModusFlags
        sbrs    r16,MOD_DebugInfo
        rjmp    Menue_FlagAus
        rjmp    Menue_FlagAn

Menue_Value:
        ldi     zl,LOW(ValueString <<1)
        ldi     zh,HIGH(ValueString <<1)
        rcall   Menue_Ausgabe
        lds     r16,ModusFlags
        sbrs    r16,MOD_Rohdaten
        rjmp    Menue_ValueMm
Menue_ValueRaw:
        ldi     zl,LOW(ValueRawString<<1)
        ldi     zh,HIGH(ValueRawString<<1)
        rjmp    Menue_FlagAusgabe
Menue_ValueMm:
        ldi     zl,LOW(ValueMmString<<1)
```

```
        ldi     zh,HIGH(ValueMmString<<1)
        rjmp    Menue_FlagAusgabe

Menue_SerielleAusgabe:
        ldi     zl,LOW(SeriellString <<1)
        ldi     zh,HIGH(SeriellString <<1)
        rcall   Menue_Ausgabe
        lds     r16,ModusFlags
        sbrs    r16,MOD_WerteSeriell
        rjmp    Menue_FlagAus
        rjmp    Menue_FlagAn

Menue_UltraAktivAusgabe:
        ldi     zl,LOW(UltraAktivString <<1)
        ldi     zh,HIGH(UltraAktivString <<1)
        rcall   Menue_Ausgabe
        lds     r16,ModusFlags
        sbrs    r16,MOD_Ultraschall
        rjmp    Menue_FlagAus
        rjmp    Menue_FlagAn

Menue_BereichAusgabe:
        ldi     zl,LOW(BereichString <<1)
        ldi     zh,HIGH(BereichString<<1)
        rcall   Menue_Ausgabe
        lds     r16,ModusFlags
        sbrs    r16,MOD_Bereich
        rjmp    Menue_Bereich5
Menue_Bereich1:
        ldi     zl,LOW(Bereich1String<<1)
        ldi     zh,HIGH(Bereich1String<<1)
        rjmp    Menue_FlagAusgabe
Menue_Bereich5:
        ldi     zl,LOW(Bereich5String<<1)
        ldi     zh,HIGH(Bereich5String<<1)
        rjmp    Menue_FlagAusgabe

Menue_FrequenzAusgabe:
        ldi     zl,LOW(FrequenzString <<1)
        ldi     zh,HIGH(FrequenzString <<1)
        rcall   Menue_Ausgabe
        lds     r16,ModusFlags
        sbrc    r16,MOD_Frequenz
        rjmp    Menue_Frequenz2
Menue_Frequenz8:
        ldi     zl,LOW(Frequenz8String<<1)
        ldi     zh,HIGH(Frequenz8String<<1)
        rjmp    Menue_FlagAusgabe
Menue_Frequenz2:
        ldi     zl,LOW(Frequenz2String<<1)
        ldi     zh,HIGH(Frequenz2String<<1)
        rjmp    Menue_FlagAusgabe
```

```
Menue_FlagAusgabe:
        ldi     r17,' '
        rcall   UART8_SendByte
        ldi     r17,'['
        rcall   UART8_SendByte
        rcall   UART8_SendPGMString
        ldi     r17,']'
        rcall   UART8_SendByte
        ret

RS232:                                          ; ein Zeichen wurde über die serielle Schnittstelle empfangen
;** das seriell empfangene Byte steht jetzt in r17
        cpi     r17,'l'
        brne    NaechsteAbfrage
        rjmp    Erzeuge_Linearisierungstabelle
NaechsteAbfrage:
        cpi     r17,'s'
        breq    Toggle_Ultraschall
        cpi     r17,'w'
        breq    Toggle_mm
        cpi     r17,'a'
        breq    Toggle_Seriell
        cpi     r17,'d'
        breq    Toggle_DebugModus               ; erweiterte Ausgaben
        cpi     r17,'b'
        breq    Toggle_Bereich
        cpi     r17,'f'
        breq    Toggle_Frequenz
        cpi     r17,'u'
        breq    Toggle_FilterUltraschall
        cpi     r17,'i'
        brne    SkipInstruction1
        rjmp    Toggle_FilterInfrarot
SkipInstruction1:
;** ungueltiges Zeichen empfangen, daher Menü ausgeben
        ldi     zl,LOW(WillkommenString <<1)
        ldi     zh,HIGH(WillkommenString <<1)
        rcall   UART8_SendPGMString
        rcall   Menue_UltraFilter
        rcall   Menue_InfraFilter
        rcall   Menue_Debug
        rcall   Menue_Value
        rcall   Menue_SerielleAusgabe
        rcall   Menue_UltraAktivAusgabe
        rcall   Menue_BereichAusgabe
        rcall   Menue_FrequenzAusgabe
        ret

Toggle_Ultraschall:
        lds     r16,Modusflags
        ldi     r17,(1<<MOD_Ultraschall)
```

```
        eor     r16,r17
Set_Ultraschall:
        sts     Modusflags,r16
        rcall   Menue_UltraAktivAusgabe
        ret
Toggle_Bereich:
        lds     r16,Modusflags
        ldi     r17,(1<<MOD_Bereich)
        eor     r16,r17
Set_Bereich:
        sts     Modusflags,r16
        rcall   Menue_BereichAusgabe
        ret
Toggle_Frequenz:
        lds     r16,Modusflags
        ldi     r17,(1<<MOD_Frequenz)
        eor     r16,r17
Set_Frequenz:
        sts     Modusflags,r16
        rcall   Menue_FrequenzAusgabe
        ret
Toggle_mm:
        lds     r16,Modusflags
        ldi     r17,(1<<MOD_Rohdaten)
        eor     r16,r17
Set_mm:
        sts     Modusflags,r16
        rcall   Menue_Value
        ret
Toggle_Seriell:
        lds     r16,Modusflags
        ldi     r17,(1<<MOD_WerteSeriell)
        eor     r16,r17
Set_Seriell:
        sts     Modusflags,r16
        rcall   Menue_SerielleAusgabe
        ret
Toggle_DebugModus:
        lds     r16,Modusflags
        ldi     r17,(1<<MOD_DebugInfo)
        eor     r16,r17
Set_DebugModus:
        sts     Modusflags,r16
        rcall   Menue_Debug
        ret
Toggle_FilterUltraschall:
        lds     r16,Modusflags
        ldi     r17,(1<<MOD_USFilter)
        eor     r16,r17
Set_FilterUltraschall:
        sts     Modusflags,r16
        rcall   Menue_UltraFilter
```

```
        ret
Toggle_FilterInfrarot:
        lds     r16,Modusflags
        ldi     r17,(1<<MOD_IRFilter)
        eor     r16,r17
Set_FilterInfrarot:
        sts     Modusflags,r16
        rcall   Menue_InfraFilter
        ret

Erzeuge_Linearisierungstabelle:
;** Das Terminal hat 'l' gesendet, die Linearisierungstabelle wird aufgenommen
;** Auf dem Terminal wird eine Distanz in mm vorgegeben (Ausgabe '00###mm;')
;** Der Benutzer stellt mit einem Hindernis die geforderte Entfernung ein und sendet
;** über das Terminal zur Bestätigung ein beliebiges Zeichen (ausser 'a', Abbruch)
;** Daraufhin werden 5 Ultraschall und 32 Infrarot-Messungen durchgeführt
;** deren Werte im Format u1;u2;u3;u4;u5;i1;i2;...;i32;CR+LF an das Terminal gesendet werden.
;** Die übertragenen Werte sind Rohdaten.
;** Auf dem PC kann eine Weiterverarbeitung der Werte erfolgen (z.B. grafische Aufbereitung,
;** geometrische Mittelwertbildung, Polynom-Regeression)
        ldi     zl,LOW(LinearString <<1)
        ldi     zh,HIGH(LinearString <<1)
        rcall   UART8_SendPGMString

        clr     r16                     ; Position mm
        sts     Lin_PositionH,r16       ; Highbyte
        sts     Lin_PositionL,r16       ; Lowbyte
EL_Schleife:
;** Schleifenzähler senden
        rcall   Delay100ms
        lds     r19,Lin_PositionH
        lds     r18,Lin_PositionL
        ldi     zl,LOW(SerString)
        ldi     zH,HIGH(SerString)
        rcall   uWordToDecNull          ; r19:r18 in Dezimalzahl wandeln

        ldi     zl,LOW(SerString)
        ldi     zH,HIGH(SerString)
        rcall   UART8_SendRAMString

        ldi     zl,LOW(EL_mmString<<1)
        ldi     zh,HIGH(EL_mmString<<1)
        rcall   UART8_SendPGMString     ; 'mm;' senden

;** auf Bestätigung oder Abbruch warten ('b' oder 'a')
EL_WarteAufBestaetigung:
        ldi     zl,LOW(EL_Wait4Handshake)
        ldi     zh,HIGH(EL_Wait4Handshake)
        rcall   UART8_GetByte
        rjmp    EL_WarteAufBestaetigung
EL_BestaetigungErfolgt:
        pop     r17
```

```
        pop     r17                     ; Rücksprungadresse von 'rcall UART8_GetByte' vom Stack entfernen (siehe UART8_GetByte)

        clr     r17
EL_Ultraschall:
;** zuerst 5 Ultraschall-Messungen
        push    r17
        rcall   USN_SendPacket
        rcall   USN_WaitMeasure
        rcall   USN_Filter              ; abhängig von MOD_USFilter wird der Timerwert gefiltert
;** Ausgabe des US-Wertes
        lds     r19,USN_RawHi
        lds     r18,USN_RawLo           ; ungefilterter Rohdatenwert
        ldi     zl,LOW(SerString)
        ldi     zH,HIGH(SerString)
        rcall   uWordToDecNull          ; r19:r18 in Dezimalzahl wandeln

        ldi     zl,LOW(SerString)
        ldi     zH,HIGH(SerString)
        rcall   UART8_SendRAMString     ; Dezimalwert senden

        ldi     zl,LOW(Divider_String<<1)
        ldi     zh,HIGH(Divider_String<<1)
        rcall   UART8_SendPGMString     ; ';' senden

        rcall   Delay100ms

        pop     r17
        inc     r17
        cpi     r17,5
        brne    EL_Ultraschall

        clr     r17

        sbi     PWR_PORT,PWR_IRSENSOR ; Infrarot-Sensor einschalten
        rcall   Delay100ms              ; Warten, bis Werte vom Sharp IR/ADC stabil

EL_Infrarot:
;** jetzt 32 Infrarot-Messungen
        push    r17
        rcall   SIR_StartMeasure        ; ADC starten
        rcall   SIR_Wait_Semaphore      ; Warte, bis ADC fertig

        mov     r18,SIR_ValueL
        mov     r19,SIR_ValueH          ; in r19:r18 steht der analoge IR-Wert

;** Ausgabe des IR-Wertes
        ldi     zl,LOW(SerString)
        ldi     zh,HIGH(SerString)
        rcall   uWordToDecNull          ; r19:r18 in Dezimalzahl wandeln

        ldi     zl,LOW(SerString)
        ldi     zh,HIGH(SerString)
```

```
            rcall       UART8_SendRAMString         ; Dezimalwert senden

            ldi         zl,LOW(Divider_String<<1)
            ldi         zh,HIGH(Divider_String<<1)
            rcall       UART8_SendPGMString         ; ';' senden
NaechsteMessung:
            pop         r17
            inc         r17
            cpi         r17,32
            brne        EL_Infrarot

            cbi         PWR_PORT,PWR_IRSENSOR ; Infrarot-Sensor aus

            ldi         zl,LOW(NewLine_String<<1)
            ldi         zh,HIGH(NewLine_String<<1)
            rcall       UART8_SendPGMString         ; ';' senden

;** Schleifenzähler pro Durchlauf erhöhen
            lds         r16,Lin_PositionL
            ldi         r18,LOW(Lin_Schrittweite)
            add         r16,r18
            sts         Lin_PositionL,r16
            lds         r17,Lin_PositionH
            ldi         r18,HIGH(Lin_Schrittweite)
            adc         r17,r18
            sts         Lin_PositionH,r17
EL_Weiter:
            cpi         r17,HIGH(Lin_MaxEntfernung)
            breq        EL_EndBedingung
EL_NaechsteEntfernung:
            rjmp        EL_Schleife
EL_Endbedingung:
            cpi         r16,LOW(Lin_MaxEntfernung)
            breq        EL_NaechsteEntfernung
            brsh        EL_Ende
            rjmp        EL_NaechsteEntfernung       ; nur wenn r21:r20 = 1000, dann Schleife zuende (Hundert Zentimeter)
EL_Ende:
            ret

EL_Wait4Handshake:
            cpi         r17,'a'
            breq        EL_Abbruch
            rjmp        EL_BestaetigungErfolgt
EL_Abbruch:
;** Abbruchbestätigung an Terminal senden
            ldi         zl,LOW(EL_AbbruchString <<1)
            ldi         zh,HIGH(EL_AbbruchString <<1)
            rcall       UART8_SendPGMString
EL_Break:
            pop         r16
            pop         r16                         ; Da abgebrochen werden soll, nicht zurück in EL... springen, sondern in MAIN
            ret
```

```
RCX_Befehl_ausfuehren:
        lds     r16,RCX_Status
        sbrs    r16,RCX_CommandRcvd     ; wenn RCX_CommandRcvd=1 (received), dann verarbeiten
        rjmp    RCX_Kein_Befehl         ; sonst zurück
RCX_Befehl_empfangen:
        mov     r16,RCX_Command
        cpi     r16,1
        breq    RCX_Befehl1
        cpi     r16,2
        breq    RCX_Befehl2
        cpi     r16,3
        breq    RCX_Befehl3
        cpi     r16,4
        breq    RCX_Befehl4
RCX_Befehl_Bestaetigen:                 ; Befehl abgearbeitet oder Befehl war nicht 1-4, also fehlerhafter Empfang
        lds     r16,RCX_Status
        cbr     r16,(1<<RCX_CommandRcvd)
        sts     RCX_Status,r16          ; RCX_CommandRcvd:=0, der nächste Befehl kann empfangen werden
RCX_Kein_Befehl:
        ret
RCX_Befehl1:
; 8 Messungen/s, US+IR, 5m
        lds     r16,Modusflags
        andi    r16,~((1<<MOD_Bereich)+(1<<MOD_Frequenz))
        ori     r16,(1<<MOD_Ultraschall)
        rcall   Set_Bereich             ; gibt den Bereich seriell aus, setzt ausserdem Modusflags
        rcall   Menue_FrequenzAusgabe
        rcall   Menue_UltraAktivAusgabe
        rjmp    RCX_Befehl_Bestaetigen
RCX_Befehl2:
; 2 Messungen/s, US+IR, 5m
        lds     r16,Modusflags
        andi    r16,~(1<<MOD_Bereich)
        ori     r16,((1<<MOD_Ultraschall)+(1<<MOD_Frequenz))
        rcall   Set_Bereich
        rcall   Menue_FrequenzAusgabe
        rcall   Menue_UltraAktivAusgabe
        rjmp    RCX_Befehl_Bestaetigen
RCX_Befehl3:
; 8 Messungen/s, nur IR, 1m
        lds     r16,Modusflags
        andi    r16,~((1<<MOD_Bereich)+(1<<MOD_Frequenz)+(1<<MOD_Ultraschall))
        rcall   Set_Bereich
        rcall   Menue_FrequenzAusgabe
        rcall   Menue_UltraAktivAusgabe
        rjmp    RCX_Befehl_Bestaetigen
RCX_Befehl4:
; 2 Messungen/s, nur IR, 1m
        lds     r16,Modusflags
        andi    r16,~((1<<MOD_Bereich)+(1<<MOD_Ultraschall))
        ori     r16,(1<<MOD_Frequenz)
```

```
            rcall       Set_Bereich
            rcall       Menue_FrequenzAusgabe
            rcall       Menue_UltraAktivAusgabe
            rjmp        RCX_Befehl_Bestaetigen

; ** Es folgen die Interrupt Handler
Sysclk:                                 ; Timer0 Overflow Handler, DAC-Ausgabe und RCX-Befehlsempfang
            push        r16             ; Register sichern
            in          r16,SREG
            push        r16

            ldi         r16,8           ; resultierende Sysclk-Frequenz etwa 500Hz
            out         TCNT0,r16

            rcall       MAX554x_Out_RAM ; DAC-Wert wird ausgegeben
            rcall       RCX_Receive     ; ggf. wird ein neuer Befehl vom LegoRCX-Baustein empfangen
            rcall       Delay100us

            pop         r16
            out         SREG,r16
            pop         r16             ; Register wiederherstellen
            reti

ADCC:                                   ; AD Conversion Complete Handler
            push        r16             ; Register sichern
            in          r16,SREG
            push        r16

            rcall       SIR_Measure_Complete    ; speichern des Resultats in SIR_ValueL+H
            pop         r16
            out         SREG,r16
            pop         r16             ; Register wiederherstellen
            reti

.CSEG
; ** Strings für Ausgabe der Werte
UltraschallString:      .DB "us: ",0
InfrarotString:         .DB " ir: ",0

; ** Menü
WillkommenString:   .DB 27,"[2JDistanzme",223,"system f",252,"r mobile Roboter",44," ",40,"c",41," 2003 O. Dahlmann",13,10,0 ; Distanzmesssystem für mobile Roboter, (c) 2003 O.Dahlmann
UltraFilterString:      .DB "u..US Filter",0
InfraFilterString:      .DB "i..IR Filter",0
DebugString:            .DB     "d..Debug",0
ValueString:            .DB "w..Werte",0
SeriellString:          .DB "a..Ausgabe",0
EL_LinString:           .DB "l..Lintab",0
UltraAktivString:       .DB     "s..Ultraschall",0
BereichString:          .DB     "b..Bereich",0
FrequenzString:         .DB     "f..Frequenz",0

; ** bedingte Ausgaben
```

```
Bereich1String:     .DB     "1m",0              ; für Bereich-Ausgabe
Bereich5String:     .DB     "5m",0
ValueMmString:      .DB     "mm",0              ; für Messwert-Ausgabe
ValueRawString:     .DB     "raw",0
Frequenz8String:    .DB "8/s",0                 ; für Frequenz-Ausgabe
Frequenz2String:    .DB "2/s",0
AusString:          .DB "aus",0                 ; für Ultraschall, serielle Ausgabe, Debug, Filter benutzt
AnString:           .DB "an",0

; ** Linearisierungstabelle aufnehmen
LinearString:       .DB 13,10,"Linearisierungstabelle aufnehmen",13,10
AbbrechenString:    .DB 13,10,"a .. Abbrechen",13,10,0
EL_AbbruchString:   .DB 13,10,"Abbruch",13,10,0

; ** verschiedende Formatierungsstrings, etc.
EL_mmString:        .DB " mm",59,0,0            ; 'mm;'
Divider_String:     .DB     59,0                ; ';' Trennzeichen für Einzelwerte (Serielle Ausgabe)
NewLine_String:     .DB     13,10,0,0           ; Carriage Return + LineFeed(Serielle Ausgabe)
InfoStringIR:       .DB "   ir: ",0
InfoStringUS:       .DB "   us: ",0
InfoStringBumper:   .DB     " bump: ",0
InfoStringDiff:     .DB     " diff: ",0
InfoStringFehler:   .DB     " error ",0

.DSEG
SerString:          .BYTE 6                     ; 6 Bytes im RAM reservieren, um einen String zur Laufzeit erzeugen und
                                                ; seriell senden zu können (Ausgabe von dynamischen Werten, z.B. Messwerten).
Modusflags:         .BYTE 1                     ; In welchem Modus befindet sich das Modul, Erklärung zu Beginn dieser Datei
Lin_PositionH:      .BYTE 1                     ; Position in mm, erst Highbyte (wird von der Funktion '(l) -> Linearisierungstabelle aufnehmen' benötigt)
Lin_PositionL:      .BYTE 1                     ; Position in mm, dann Lowbyte

.INCLUDE "..\libraries\numout_lib_short.asm"
.INCLUDE "..\libraries\delay_short.asm"
.INCLUDE "..\libraries\MAX554xDAC_lib.asm"
.INCLUDE "..\libraries4433\SHARP_IR_lib.asm"
.INCLUDE "..\libraries4433\Ultrasonic_lib.asm"
.INCLUDE "..\libraries\Multiplication16x16_lib.asm"
.INCLUDE "..\libraries\Multiplication16x8_lib.asm"
.INCLUDE "..\libraries\Division24Bit_lib.asm"
.INCLUDE "..\libraries4433\LegoRCX_Receive_lib.asm"
.INCLUDE "..\libraries4433\UART8_lib.asm"
.INCLUDE "..\libraries\X9C104_lib_short.asm"
.INCLUDE "..\libraries\SensorVerarbeitung_lib.asm"
.INCLUDE "..\libraries\RCX_Output_lib.asm"
.INCLUDE "..\libraries4433\EEPROM_lib.asm"
```

numout_lib_short.asm

```
; ************************************************************
; * Library:          numout_lib_short für AVR Prozessoren
; * Autor:            Oliver Dahlmann
; * Version:          1.0
; * Beschreibung:     Bibliothek mit verschiedenen Unterprogrammen
; *                   zur Ausgabe von Zahlen in verschiedenen
; *                   Formaten, ASCII-Strings sind Ziel
; *                   jede Funktion auf Wunsch nullterminiert
; ************************************************************

; (char*:Z) uByteToDec(ubyte:r18)               konvertiert das Byte in r18 in ASCII, Dezimaldarstellung, 3 Stellen
; uByteToDecNull                                nullterminiert
;
; (char*:Z) ByteToDec(ubyte:r18)                konvertiert das Byte mit Vorzeichen in r18 in ASCII, Dezimaldarstellung, 3 Stellen
; ByteToDecNull                                 nullterminiert
;
; (char*:Z) uWordToDec(uword:[r19:r18])         konvertiert das Wort in r19:r18 in ASCII, Dezimaldarstellung, 5 Stellen
; uWordToDecNull                                nullterminiert

.CSEG
uByteToDecNull:
        rcall   uByteToDec
        rjmp    NUMOUT_String_Terminieren
uByteToDec:
; Unsigned Byte in r18 to ASCII, Decimal
        push    r18
        push    r19
        push    r20
        push    r21
        push    r22
        push    r23
        push    r24
        push    r25
        push    r26

        clr     r24                     ; 100er
        clr     r25                     ; 10er
        clr     r26                     ; 1er
        clr     r23                     ; Summand für Übertrags-Addition = NULL
        ldi     r20,8                   ; Schleifen-Zähler
BTD_Horner:
BTD_Stelle1:
        clc
        rol     r26                     ;   1er =   1er*2
        rol     r25                     ;  10er =  10er*2
        rol     r24                     ; 100er = 100er*2
        rol     r18                     ; Oberstes Bit in Carry schieben
        adc     r26,r23                 ; Carry auf 1er nach Verdopplung addieren
        cpi     r26,10                  ; ist das Ergebnis >=10 ?
        brlt    BTD_Stelle10            ; Wenn nein, kein Uebertrag zu behandeln
```

```
        subi    r26,10                  ; Da r26 (die 1er Stelle) >=10 war, 10 abziehen
        inc     r25                     ; Die 1er-Stelle hatte einen Uebertrag, also 10er um 1 inkr.

BTD_Stelle10:
        cpi     r25,10
        brlt    BTD_Stelle100
        subi    r25,10                  ; Da es in den 10ern einen Übertrag gab. 10 abziehen
        inc     r24                     ; 100er um 1 erhöhen

BTD_Stelle100:
        dec     r20
        brne    BTD_Horner

        ldi     r20,'0'
        add     r24,r20
        st      Z+,r24
        add     r25,r20
        st      Z+,r25
        add     r26,r20
        st      Z+,r26

        pop     r26
        pop     r25
        pop     r24
        pop     r23
        pop     r22
        pop     r21
        pop     r20
        pop     r19
        pop     r18
        ret

uWordToDecNull:
        rcall   uWordToDec
        rjmp    NUMOUT_String_Terminieren
uWordToDec:                             ; in r19:r18 steht das zu wandelnde Wort (r19 Highbyte, r18 Lowbyte)
; Unsigned Word to Dezimal
        push    r18
        push    r19
        push    r20
        push    r21
        push    r22
        push    r23
        push    r24
        push    r25
        push    r26

        clr     r22                     ; 10.000er
        clr     r23                     ; 1.000er
        clr     r24                     ; 100er
        clr     r25                     ; 10er
```

```
        clr     r26                     ; 1er
        clr     r21                     ; Summand für Übertrags-Addition = NULL
        ldi     r20,16                  ; Schleifen-Zähler
WTD_Horner:
WTD_Stelle1:
        clc
        rol     r26                     ;      1er =   1er    * 2
        rol     r25                     ;     10er =  10er    * 2
        rol     r24                     ;    100er = 100er    * 2
        rol     r23                     ;  1.000er = 1.000er  * 2
        rol     r22                     ; 10.000er = 10.000er * 2
        rol     r18                     ; Oberstes Bit von r19 in unterstes Bit von r18 schieben
        rol     r19                     ; Oberstes Bit in Carry schieben
        adc     r26,r21                 ; Carry auf 1er nach Verdopplung addieren

        cpi     r26,10                  ; ist das Ergebnis >=10 ?
        brlt    WTD_Stelle10            ; Wenn nein, kein Uebertrag zu behandeln

        subi    r26,10                  ; Da r26 (die 1er Stelle) >=10 war, 10 abziehen
        inc     r25                     ; Die 1er-Stelle hatte einen Uebertrag, also 10er um 1 inkr.

WTD_Stelle10:
        cpi     r25,10
        brlt    WTD_Stelle100
        subi    r25,10                  ; Da es in den 10ern einen Übertrag gab. 10 abziehen
        inc     r24                     ; 100er um 1 erhöhen

WTD_Stelle100:
        cpi     r24,10
        brlt    WTD_Stelle1000
        subi    r24,10                  ; Da es in den 100ern einen Übertrag gab. 10 abziehen
        inc     r23                     ; 1.000er um 1 erhöhen

WTD_Stelle1000:
        cpi     r23,10
        brlt    WTD_Stelle10000
        subi    r23,10                  ; Da es in den 1.000ern einen Übertrag gab. 10 abziehen
        inc     r22                     ; 10.000er um 1 erhöhen

WTD_Stelle10000:
        dec     r20
        brne    WTD_Horner
        ldi     r20,'0'
        add     r22,r20
        st      Z+,r22
        add     r23,r20
        st      Z+,r23
        add     r24,r20
        st      Z+,r24
        add     r25,r20
        st      Z+,r25
        add     r26,r20
```

```
            st      Z+,r26
            pop     r26
            pop     r25
            pop     r24
            pop     r23
            pop     r22
            pop     r21
            pop     r20
            pop     r19
            pop     r18
            ret

ByteToDecNull:
            rcall   ByteToDec
            rjmp    NUMOUT_String_Terminieren
ByteToDec:
; ein vorzeichenbehaftetes Byte wird in ASCII gewandelt
; das Byte wird in r18 übergeben
            push    r18
            push    r20
            cpi     r18,$80
            brsh    BTD_Negativ
BTD_Positiv:
            ldi     r20,'+'                 ; das Vorzeichen '+', wahlweise durch ' ' zu ersetzen
            st      Z+,r20
            rcall   uByteToDec
            pop     r20
            pop     r18
            ret
BTD_Negativ:
            ldi     r20,'-'
            st      Z+,r20
            clr     r20
            sub     r20,r18
            mov     r18,r20
            rcall   uByteToDec
            pop     r20
            pop     r18
            ret

NUMOUT_String_Terminieren:
            push    r16
            clr     r16
            st      Z,r16                   ; String mit Null terminieren
            pop     r16
            ret
```

delay_short.asm

```
; ************************************************************
; * Library:          delay_short für AVR Prozessoren
; * Autor:            Oliver Dahlmann
; * Version:          1.0
; * Beschreibung:     Bibliothek mit verschiedenen Verzögerungs-
; *                   routinen, Zeitangaben bezogen auf 8MHz Clock
; ************************************************************
;

.CSEG

Delay100ms:                                     ; @8MHz
        push    r16
        ldi     r16,4
Delay100msLoop:
        rcall   Delay25ms
        dec     r16
        brne    Delay100msLoop
        pop     r16
        ret

Delay100us:                                     ; @8Mhz
        push    r16
        ldi     r16,255                         ; etwa 100usec
        rjmp    Delay100usEinsprung

Delay25ms:                                      ; @8Mhz
        push    r16
        clr     r16
Delay100usEinsprung:
        push    r17
        clr     r17
Delay25msLoop:
        inc     r17
        brne    Delay25msLoop
        inc     r16
        brne    Delay25msLoop
        pop     r17
        pop     r16
        ret
```

MAX554xDAC_lib.asm

```
; ************************************************************
; * Library:          MAX554xDAC_lib für AVR Prozessoren
; * Autor:            Oliver Dahlmann
; * Version:          1.0
; * Beschreibung:     Schnelle Bibliothek mit Unterprogrammen
; *                   zur Ansteuerung eines MAX554x Digital/Analog
; *                   Converters (16Bit).
; ************************************************************
; * Achtung:          Der MAX554x arbeitet stabil bis zu herunter
; *                   zu statischem Betrieb (DC)
; ************************************************************
;
; void MAX554x_Init(void)                       setzt die benutzten Speicherstellen auf Null und führt eine
;                                               DA-Wandlung (DAC=0) durch
;
; void MAX554x_Set_RAM(r17:r16)                 DAC_Wert := r17:r16, die Ausführung
;                                               von Interrupts wird solange gesperrt (Critical Section)
;
; void MAX554x_Out_RAM(void)                    Der Wert in MAX_DAC_Wert wird am DA-Wandler ausgegeben
;
; void MAX554x_Out_Register(r17:r16)            r17:r16 wird auf dem DA-Wandler ausgegeben
;
; ** Definition der verwendeten I/O-Leitungen ...
;.EQU    MAX_CS          =       2       ; Pin4 MAX554x /CS (Chip Select, active low)
;.EQU    MAX_SCLK        =       3       ; Pin5 MAX554x /SCLK (Serial Clock, steigende Flanke übernimmt Daten-Bit)
;.EQU    MAX_DIN         =       4       ; Pin6 MAX554x DIN (Data In)
; ** ... an Port
;.EQU    MAX_PORT        =       PORTD

.CSEG

MAX554x_Init:
; ** Dummy-Wandlung mit DAC=0
        clr     r16
        sts     MAX_DAC_Wert,r16
        sts     MAX_DAC_Wert+1,r16
        rcall   MAX554x_Out_RAM
        ret

MAX554x_Set_RAM:
        push    r18
        in      r18,TIMSK               ; Critical Section, Timer0-Interrupt nicht erlaubt
        andi    r18,~(1<<TOIE0)
        out     TIMSK,r18
        sts     MAX_DAC_Wert,r17        ; Highbyte
        sts     MAX_DAC_Wert+1,r16      ; Lowbyte
        in      r18,TIMSK
        ori     r18,(1<<TOIE0)          ; Timer0-Interrupt wieder erlauben
        out     TIMSK,r18
        pop     r18
```

```
        ret

MAX554x_Out_RAM:                        ; wird aus dem Hauptprogramm vom Timer0Overflow-IRQ aufgerufen
        push    r20
        push    r17
        push    r16
        lds     r17,MAX_DAC_Wert
        lds     r16,MAX_DAC_Wert+1
        rjmp    MAX554x_DA

MAX554x_Out_Register:
; ** Der Wandler hat drei digitale Eingänge: /SCLK, DIN und /CS
; ** Format der Daten siehe Product specification MAX554x, MAXIM
; ** in r17:r16 wird der auszugebende Wert übergeben
        push    r20
        push    r17
        push    r16
MAX554x_DA:
        ldi     r20,16
        cbi     MAX_PORT,MAX_CS
MAX554x_Loop:
        cbi     MAX_PORT,MAX_SCLK
        lsl     r16
        rol     r17                     ; C <- r17:r16 <-0, das MSB ist jetzt in C
        brcs    MAX554x_Data_1
MAX554x_Data_0:
        cbi     MAX_PORT,MAX_DIN        ; 0 ausgeben
        sbi     MAX_PORT,MAX_SCLK
        dec     r20
        brne    MAX554x_Loop
        rjmp    MAX554x_WortEnde
MAX554x_Data_1:
        sbi     MAX_PORT,MAX_DIN        ; 1 ausgeben
        sbi     MAX_PORT,MAX_SCLK
        dec     r20
        brne    MAX554x_Loop
MAX554x_WortEnde:
        cbi     MAX_PORT,MAX_SCLK
        sbi     MAX_PORT,MAX_CS         ; DAC Update
        pop     r16
        pop     r17
        pop     r20
        ret

.DSEG
MAX_DAC_Wert:   .BYTE 2                 ; 16Bit-Wert für den DA-Wandler
```

SHARP_IR_lib.asm

```
; ************************************************************
; * Library:          SHARP_IR_lib für AVR Prozessoren
; * Autor:            Oliver Dahlmann
; * Version:          1.0
; * Beschreibung:     Bibliothek mit verschiedenen Unterprogrammen
; *                   zur Nutzung eines SHARP Infrarot-Sensors vom
; *                   Typ GP2D12 am analogen Port eines AT90S4433
; *                   Dieser nach dem Triangulationsprinzip arbei-
; *                   tende Sensor hat einen Erkennungsbereich von
; *                   etwa 10-80cm.
; ************************************************************

.EQU    SIR_Distanz         = 10            ; Stützstellenabstand der Linearisierungstabelle in mm

.EQU    IR_a                = 128           ; siehe IR-Filterbeschreibung
.EQU    IR_b                = 128

.DEF    SIR_Semaphore       = r10           ; wird vom ADCC-Interrupt gesetzt. Das Hauptprogramm kann somit einen ADCC-Interrupt feststellen
.DEF    SIR_ValueL          = r11           ; Das Messergebnis
.DEF    SIR_ValueH          = r12

.CSEG

SIR_Messung8:
; ** Die folgende Funktion führt 8 Infrarot-Messungen durch und bildet dann einen Mittelwert
; ** Ausserdem werden Minimum und Maximum der Messwerte ermittelt
        push    r20
        rcall   SIR_Mittelwert_loeschen ; Mittelwert und Minimum/Maximum löschen
        ldi     r20,8
SIR_M8_Schleife:
        rcall   SIR_StartMeasure        ; ADC starten
        rcall   SIR_Wait_Semaphore      ; Warte, bis ADC fertig (ADCC-Interrupt)
        ;rcall  SIR_WaitMeasure         ; alternative Wartemethode ohne Interrupt
        rcall   SIR_Sum_Min_Max         ; Summe aller 8 Messergebnisse und Bestimmung von Minimum/Maximum
        dec     r20
        brne    SIR_M8_Schleife         ; 8mal wiederholen

        rcall   SIR_Sigma               ; Abweichung (Sigma=Maximum-Minimum) bestimmen
        rcall   SIR_Mittelwert          ; Mittelwert (=Summe/8)
        pop     r20
        ret

SIR_StartMeasure:
        push    r16
        ldi     r16,(1<<RXEN)+(1<<TXEN)
        out     UCSRB,r16               ; UART Sende- und Empfangs-Interrupts deaktivieren

        clr     r16
        mov     SIR_Semaphore,r16       ; Semaphore auf NULL, Wandlung gestartet
        sbi     ADCSR,ADSC
```

```
        sleep                                   ; und schlafen, bis IRQ kommt
        ldi     r16,(1<<RXCIE)+(1<<TXCIE)+(1<<RXEN)+(1<<TXEN)
        out     UCSRB,r16                       ; Sende- und Empfangs-Interrupts aktivieren
        pop     r16
        ret

SIR_WaitMeasure:
        push    r16
SIR_WaitMeasureLoop:
        in      r16,ADCSR
        andi    r16,(1<<ADIF)
        cpi     r16,(1<<ADIF)                   ; Warte bis ADIF-Bit gesetzt
        brne    SIR_WaitMeasureLoop
        rcall   SIR_Measure_Complete
        in      r16,ADCSR
        ori     r16,(1<<ADIF)
        out     ADCSR,r16                       ; ADIF-Bit löschen (durch Schreiben einer 1 in ADIF)
        pop     r16
        ret

SIR_Measure_Complete:                           ; wird von SIR_WaitMeasure oder vom ADCC-Interrupt aufgerufen
        in      r16,ADCL
        mov     SIR_ValueL,r16
        in      r16,ADCH
        mov     SIR_ValueH,r16                  ; Messergebnis speichern
        ser     r16
        mov     SIR_Semaphore,r16               ; Semaphore auf $FF, Wandlung komplett
        ret

SIR_Wait_Semaphore:                             ; wenn mit ADCC-Interrupt gearbeitet wird, diese Routine anstelle von SIR_WaitMeasure verwenden
        mov     r16,SIR_Semaphore
        cpi     r16,$FF
        breq    SIR_Wait_Semaphore_Set          ; falls ein anderer IRQ zum Aufwachen aus Idle(Sleep) geführt hat,
        rcall   SIR_StartMeasure                ; dann Conversion erneut durchführen
        rjmp    SIR_Wait_Semaphore              ; und erneut testen
SIR_Wait_Semaphore_Set:
        ret

SIR_Mittelwert_loeschen:
        push    r16
        clr     r16
        sts     SIR_MittelwertH,r16
        sts     SIR_MittelwertM,r16
        sts     SIR_MittelwertL,r16
        sts     SIR_MaximumH,r16
        sts     SIR_MaximumL,r16

        ser     r16
        sts     SIR_MinimumH,r16
        sts     SIR_MinimumL,r16

        pop     r16
```

```
        ret

SIR_Sum_Min_Max:
; ** Errechnung der 24-Bit-Summe aller IR-Messwerte, bekommt in r18:r17 den Analogwert übergeben
        lds     r16,SIR_MittelwertL
        add     r16,SIR_ValueL
        sts     SIR_MittelwertL,r16
        lds     r16,SIR_MittelwertM
        adc     r16,SIR_ValueH
        sts     SIR_MittelwertM,r16
        push    r17
        clr     r17
        lds     r16,SIR_MittelwertH
        adc     r16,r17
        sts     SIR_MittelwertH,r16
        pop     r17

; ** Errechnung des IR-Maximums
        lds     r16,SIR_MaximumH
        cp      SIR_ValueH,r16
        brlo    SIR_Kein_Neues_Maximum
        brne    SIR_Neues_Maximum
        lds     r16,SIR_MaximumL        ; Hier angekommen waren die MSB's des neuen Werts und des Maximums gleich
        cp      SIR_ValueL,r16          ; Also LSB abfragen
        brlo    SIR_Kein_Neues_Maximum
SIR_Neues_Maximum:
        sts     SIR_MaximumH,SIR_ValueH
        sts     SIR_MaximumL,SIR_ValueL
SIR_Kein_Neues_Maximum:

; ** Errechnung des IR-Minimums
        lds     r16,SIR_MinimumH
        cp      r16,SIR_ValueH
        brlo    SIR_Kein_Neues_Minimum
        brne    SIR_Neues_Minimum
        lds     r16,SIR_MinimumL        ; Hier angekommen waren die MSB's des neuen Werts und des Minimums gleich
        cp      r16,SIR_ValueL          ; Also LSB abfragen
        brlo    SIR_Kein_Neues_Minimum
SIR_Neues_Minimum:
        sts     SIR_MinimumH,SIR_ValueH
        sts     SIR_MinimumL,SIR_ValueL
SIR_Kein_Neues_Minimum:
        ret

SIR_Sigma:
; ** Abweichung Sigma bestimmen (Sigma=Maximum-Minimum)
        push    r16
        push    r17
        lds     r16,SIR_MaximumL
        lds     r17,SIR_MinimumL
        sub     r16,r17
        sts     SIR_SigmaL,r16
```

```
            lds     r16,SIR_MaximumH
            lds     r17,SIR_MinimumH
            sbc     r16,r17
            sts     SIR_SigmaH,r16
            pop     r17
            pop     r16
            ret

SIR_Mittelwert:
; ** Errechnung des IR-Mittelwerts aus der 24Bit-Summe
; ** Bei 32 Samples -> Summe 5 Bits nach rechts schieben, um Mittelwert zu erhalten
; ** Bei 8 Samples -> Summe 3 Bits nach rechts schieben, um Mittelwert zu erhalten
            push    r16
            push    r17
            push    r18

            lds     r18,SIR_MittelwertH
            lds     r17,SIR_MittelwertM
            lds     r16,SIR_MittelwertL

            lsr     r18
            ror     r17
            ror     r16                 ; Summe/2
            lsr     r18
            ror     r17
            ror     r16                 ; Summe/4
            lsr     r18
            ror     r17
            ror     r16                 ; Summe/8

            sts     SIR_RawHi,r17
            sts     SIR_RawLo,r16       ; für die Weiterverarbeitung durch die Filter-Routine

            pop     r18
            pop     r17
            pop     r16

            ret

SIR_Filter:
; ** wird vor der Umrechnung des Wertes in mm aufgerufen
            lds     r17,SIR_RawHi
            lds     r16,SIR_RawLo       ; in r17r16 steht der aktuelle Messwert
; ** wenn MOD_IRFilter gesetzt, dann den Wert nicht direkt zurückgeben, sondern filtern
            lds     r18,Modusflags
            sbrs    r18,MOD_IRFilter
            rjmp    SIR_FilterBeendet
; ** je weiter das Objekt entfernt ist, desto unzuverlässiger wird der Messwert
; ** im folgenden wird dafür gesorgt, das der Messwert umso schneller in das Ergebnis einfliesst,
; ** desto näher das Messobjekt am Sensor ist
; ** D(n) beschreibt die errechnete Distanz der Vergangenheit, D(n+1) wird der neue Ausgabewert
```

```
; ** D(n+1) := [(a*(Messwert/Maximum)+b) * Messwert] + [(1-(a*(Messwert/Maximum)-b)) * D(n)]
; ** Maximum bezeichnet den maximalen Wertebereich des Messwerts, also 1024 bei 10Bit-A/D
; ** mit Messwert=0..1023, Messwert in r17r16, {Maximum DIV 4} ergibt 256
        lsr     r17
        ror     r16
        lsr     r17
        ror     r16                     ; M' sei Messwert DIV 4 (aus 10Bit -> 8Bit)
        ldi     r18,IR_a
        rcall   Multi16x8               ; r22r21r20 := M' * a
; ** r22r21 := (M' * a) / Mmax'
        ldi     r16,IR_b
        add     r21,r16                 ; r21 := [(M' * a) / Mmax'] * b
; ** Messwert M zurückholen
        lds     r17,SIR_RawHi
        lds     r16,SIR_RawLo
; ** Gewichtungsfunktion speichern für weitere Verwendung
        push    r21
; ** mit M multiplizieren
        mov     r18,r21
        rcall   Multi16x8               ; (gewichteter Messwert*256) in r22r21r20
; ** 1-G (bzw. 256-G)
        pop     r18
        com     r18
        inc     r18                     ; 256-G
; ** gewichteten Messwert zwischenspeichern
        push    r22
        push    r21
        push    r20
; ** (1-G)*D(n)
        lds     r17,SIR_ValHi
        lds     r16,SIR_ValLo           ; in r17r16 steht die Vorgeschichte
        rcall   Multi16x8
; ** gewichteten Messwert addieren
        pop     r16
        add     r20,r16
        pop     r16
        adc     r21,r16
        pop     r16
        adc     r22,r16                 ; r22r21:=D(n+1)

        mov     r17,r22
        mov     r16,r21
SIR_FilterBeendet:
        sts     SIR_ValLo,r16
        sts     SIR_ValHi,r17
        sts     SIR_RawLo,r16
        sts     SIR_RawHi,r17           ; falls nicht linearisiert wird, steht in SIR_Raw der gefilterte Wert
        ret

SIR_mm:
; ** Hier wird der Messwert U linearisiert und in einen Entfernungs-Wert [mm] umgewandelt
; ** Hierzu wird eine Linearisierungstabelle verwendet
```

```
; ** Zunächst die Stelle X in der Lintab suchen, wo
;
; ** U           > U > U
; **  Lintab(x)        Lintab(x+1)
;
            lds     r21,SIR_ValHi
            lds     r20,SIR_ValLo           ; Der zu suchende Wert Uir in r21:r20

; ** Index=0
            clr     r0
SIR_WertSuchen:
            ldi     r16,SIR_Lintab          ; Startadresse der Tabelle im EEPROM
            add     r16,r0                  ; darauf den Index addieren, EEPROM-Adresse jetzt in r16
            rcall   EEPROM_Read             ; Datenbyte steht jetzt in r16
            push    r16
            inc     r0                      ; Index++
            ldi     r16,SIR_Lintab          ; Startadresse der Tabelle im EEPROM
            add     r16,r0                  ; darauf den Index addieren, EEPROM-Adresse jetzt in r16
            rcall   EEPROM_Read             ; Tabelleneintrag steht jetzt in r17r16
            inc     r0                      ; Index++
            mov     r17,r16
            pop     r16                     ; LSB wurde zuerst gelesen

            rcall   SIR_VergleichAB         ; vergleiche U und Ulintab
            breq    SIR_WerteGleich         ; Beide Werte sind gleich, eine Interpolation ist nicht nötig
            brcc    SIR_WertGefunden        ; in der Lintab wurde der erste Wert gefunden, der kleiner als der Messwert ist
            rjmp    SIR_WertSuchen          ; den nächsten Wert prüfen
SIR_WertGefunden:
; ** Jetzt den vorigen Wert aus der Tabelle holen und von diesem abziehen
            push    r17
            push    r16                     ; Wert (Tabelleneintrag) absichern
            dec     r0                      ; Index:=Index-4
            dec     r0                      ; jetzt steht r0 auf dem gerade gelesenen Wort
            dec     r0
            dec     r0                      ; jetzt steht r0 auf dem davor abgespeicherten Wort

            ldi     r16,SIR_Lintab          ; Startadresse der Tabelle im EEPROM
            add     r16,r0                  ; darauf den Index addieren, EEPROM-Adresse jetzt in r16
            rcall   EEPROM_Read             ; Datenbyte steht jetzt in r16
            mov     r18,r16
            inc     r0
            ldi     r16,SIR_Lintab          ; Startadresse der Tabelle im EEPROM
            add     r16,r0                  ; darauf den Index addieren, EEPROM-Adresse jetzt in r16
            rcall   EEPROM_Read
            mov     r19,r16                 ; Wort steht jetzt in r19r18 (r19r18:=U(x))
            inc     r0

            pop     r16
            pop     r17                     ; Wert (Tabelleneintrag) wiederherstellen

            sub     r20,r16
            sbc     r21,r17                 ; r21:r20 := U(w) - U(x+1)
```

```
            sub     r18,r16
            sbc     r19,r17                 ; r19:r18 := U(x) - U(x+1)

            lsr     r0                      ; in r0 steht jetzt die Nummer des Eintrags in der Lintab
            push    r0                      ; Index des gefundenen Tabelleneintrags sichern

            push    r18                     ; Low[U(x) - U(x+1)] absichern
            mov     r17,r21
            mov     r16,r20
            ldi     r18,SIR_Distanz
            rcall   Multi16x8               ; in r22:r21:r20 := {U(w) - U(x+1)} * Distanz

            pop     r16
            mov     r17,r19
            clr     r18                     ; r18:r17:r16 := r19:r18 = U(x) - U(x+1)

            rcall   Divide24x24             ; r2:r1:r0 := [{U(w) - U(x+1)} * Distanz] / (U(x) - U(x+1))
            mov     r2,r18
            mov     r1,r17
            mov     r0,r16                  ; Ergebnis der Division durch gerundetes Ergebnis überschreiben

            pop     r16                     ; Index des gefundenen Tabelleneintrags wiederherstellen
            clr     r17                     ; Tabelle hat max. 126 Einträge, daher r17:=0

            ldi     r18,SIR_Distanz
            rcall   Multi16x8               ; r22:r21:r20 := D(x+1) * Distanz

            sub     r20,r0
            sbc     r21,r1
            sbc     r22,r2                  ; r22:r21:r20 := {D(x+1) * Distanz} - {[{U(w) - U(x+1)} * Distanz] / (U(x) - U(x+1)}
            rjmp    SIR_Lin_Ende
SIR_WerteGleich:
            clr     r17
            lsr     r0
            mov     r16,r0                  ; der Index des Eintrags steht in r17r16

            ldi     r18,SIR_Distanz
            rcall   Multi16x8               ; r22:r21:r20 := D(x+1) * Distanz
SIR_Lin_Ende:
; ** Zum Schluß abfragen, ob SIR_Raw<0, dann SIR_Raw auf Null setzen
            sbrs    r21,7                   ; Bit7 von r21 gesetzt -> Wert negativ
            rjmp    SIR_WertSpeichern
            clr     r21
            clr     r20
SIR_WertSpeichern:
            sts             SIR_RawHi,r21
            sts             SIR_RawLo,r20
            ret

SIR_VergleichAB:                            ; A = r17:r16, B = r20:r19
            cp      r21,r17                 ; HighBytes vergleichen
```

```
        breq    SIR_BGleichAHigh
        ret
SIR_BGleichAHigh:
        cp      r20,r16                 ; LowBytes vergleichen
        ret                             ; wenn A = B -> C=0, Z=1
                                        ; wenn A < B -> C=0, Z=0
                                        ; wenn A > B -> C=1, Z=0

.ESEG
.ORG 0x00
SIR_Lintab_Anfang: .dw 1023   ; Anfangswert der Linearisierungstabelle, erspart einige Sonderbehandlungen in der Routine
; ** Jetzt kommen die ersten 800mm beginnend bei 0mm, mehr als 800mm ist beim verwendeten Sensor nicht sinnvoll
SIR_LINTAB:     .dw 855,797,745,698,655,616,581,550,521,496,472,451,432,415,399,385,371,359,348,338,328,319,311
                .dw 303,295,288,281,275,269,263,257,251,246,241,235,231,226,221,217,213,209,205,201,198,194,191
                .dw 188,185,182,180,177,175,173,171,169,167,165,163,162,160,158,157,155,153,152,150,148,146,144,142,140,139,137,135,133,132,130,129,128,127,126
SIR_Lintab_Ende:.dw 0                   ; Ende der Tabelle mit 0 kennzeichnen


.DSEG

SIR_MittelwertH:    .BYTE   1
SIR_MittelwertM:    .BYTE   1
SIR_MittelwertL:    .BYTE   1       ; in MittelwertM:MittelwertL steht der arithmetische Mittelwert über n Messungen

SIR_MinimumH:       .BYTE   1
SIR_MinimumL:       .BYTE   1       ; Das Minimum von n Messungen

SIR_MaximumH:       .BYTE   1
SIR_MaximumL:       .BYTE   1       ; Das Maximum von n Messungen

SIR_SigmaH:         .BYTE   1
SIR_SigmaL:         .BYTE   1       ; Die Abweichung über n Messungen

SIR_RawHi:          .BYTE   1       ; zunächst die von der Messroutine gelieferten Rohdaten des Sensors,
SIR_RawLo:          .BYTE   1       ; wird von SIR_mm zur Ablage des Wertes in mm verwendet

SIR_ValHi:          .BYTE   1
SIR_ValLo:          .BYTE   1       ; die möglicherweise gefilterten Daten des Sensors VOR der Linearisierung
```

Ultrasonic_lib.asm

```
; ************************************************************
; * Library:          Ultrasonic_lib für AVR Prozessoren
; * Autor:            Oliver Dahlmann
; * Version:          1.0
; * Beschreibung:     Bibliothek mit verschiedenen Unterprogrammen
; *                   zur Nutzung eines Polaroid Ultraschall-
; *                   Entfernungssensors.
; *                   Diese Library arbeitet mit einer Beschaltung
; *                   die komplett ohne Spezial-IC's auskommt
; ************************************************************
;
;
; ** verwendete Bits ...

;.EQU    USN_REC         =       0       ; Komparator mit Empfangssignal
;.EQU    USN_XMIT        =       1       ; Puls-Erzeugung (MOSFET)
;.EQU    USN_TRESHOLD    =       3       ; Trigger für Kondensator-Entladekurve für zeitabhängigen Treshold des US Komparators
;.EQU    GAIN_20         =       5       ; Der Transistor T2 kann die Eingangsverstärkung des Ultraschallempfängers von 1x auf 20x umschalten

; ** ... an Port
;.EQU    USN_PORT        =       PORTB   ; mit diesem Port ist der Polaroid-Sensor verbunden (Ausgang)
;.EQU    USN_PIN         =       PINB    ; mit diesem Port ist der Polaroid-Sensor verbunden (Eingang)

.EQU    FilterFreq      =       64      ; entspricht Ergebnis = (1/4*Messwert) + (3/4*History)

.DEF    USN_RuntimeH    =       r13     ; Die Laufzeit des zurückgelegten Impulses
.DEF    USN_RuntimeL    =       r14
.DEF    USN_Semaphore   =       r15     ; wenn die Semaphore=1, wurde die Messung beendet

.CSEG

USN_Init:
        push    r16
        cbi     USN_PORT,USN_XMIT       ; die XMIT-Leitung auf 0
        sbi     USN_PORT,USN_TRESHOLD   ; Kondensator für die nächste Messung schon mal laden
        rcall   X9C_Select0
        rcall   X9C_Reset               ; Verstärkungsfaktor bei 2 beginnen, dazu Digital-Poti zurücksetzen
        rcall   X9C_Deselect0

        clr     r16
        sts     USN_RawLo,r16
        sts     USN_RawHi,r16

        pop     r16
        ret

USN_SendPacket:
        sbi     USN_PORT,USN_TRESHOLD   ; Kondensator für die nächste Messung laden
        rcall   X9C_Select0
        rcall   X9C_Reset               ; Verstärkungsfaktor bei 2 beginnen, dazu Digital-Poti zurücksetzen
        push    r17
```

```
        push    r20

        cli                             ; Critical Section, ein Interrupt würde die Genauigkeit der Messung stören
        clr     USN_Semaphore           ; Eine NULL in der Semaphore zeigt an, dass die Messung noch nicht beendet wurde

        clr     r16                     ; Timer1 bei $0000 starten
        out     TCNT1H,r16
        out     TCNT1L,r16

; ** Schwingungen erzeugen
; ** Achtung: Resonanzfrequenz des Transducers liegt bei 54,5kHz (gemessen)
; ** Also 9,17us HIGH, 9,17us LOW, 0,375us wird durch dec/brne verzögert -> Verzögerung inkl. RCALL/RET ~8,8us
        ldi     r17,8                   ; 8 Schwingungen, halb soviele wie das Original-IC
USN_PulseGeneration:
        sbi     USN_PORT,USN_XMIT       ; XMIT-Treiber auf 1, Dauer 0,125usec
        rcall   USN_PulseDelay          ; 8,625usec warten
        nop
        nop
        nop
        nop                             ; 4*nop=0,5usec -> insgesamt 9,5+0,5+0,125=10,125usec -> 49,382kHz
        nop                             ; für 50/50 Puls/Pausen Verhältnis (dec/brne ausgleichen)
        cbi     USN_PORT,USN_XMIT       ; XMIT-Treiber wieder auf 0
        rcall   USN_PulseDelay
        nop
        nop
        dec     r17
        brne    USN_PulseGeneration     ; Gewünschte Anzahl Pulse erzeugen

; ** Die Schwingungen zu erzeugen hat bei 4 Perioden etwa 73 usec gedauert
; ** Jetzt eine gewisse Zeitspanne warten und danach die Echo-Unterdrückung aufheben
; ** Jetzt das Ausschwingen des Systems Transformator, Transducer, Kondensatoren C12, C21 abwarten (etwa 1ms)

        ldi     r16,243                 ; ungefähr 1ms
        clr     r17
        inc     r17
USN_Delay1msLoop:
        inc     r17
        brne    USN_Delay1msLoop
        inc     r16
        brne    USN_Delay1msLoop

        cbi     USN_PORT,USN_TRESHOLD ; Den Kondensator über Widerstand entladen lassen.

; ** Hier GAIN-Control für Empfangsverstärker schrittweise erhöhen
; ** Erstmal etwa 2ms warten (abhängig von vorheriger Wartezeit zur Echounterdrückung)
; ** Die Wartezeiten sind nur ungefähr einzuhalten, da sie nur die Verstärkung des Empfänger-IC's betreffen
; ** Das Messergebnis wird dadurch wahrscheinlich nicht wesentlich beeinflusst
        rcall   Delay100us
; ** Laufzeit wird ab hier gemessen
        in      r16,TIMSK
        ori     r16,(1<<TICIE1)+(1<<TOIE1)
        out     TIMSK,r16               ; Interrupt für Timer Capture und Timer Overflow freigeben
```

```
        ldi     r16,(1<<ICF1)+(1<<TOV1)     ; eventuell zwischengespeichertes Flag für Timer Capture IRQ löschen (wird durch Setzen auf Eins gelöscht)
        out     TIFR,r16                    ; sonst wird der Interrupt sofort ausgelöst wenn er freigegeben wird
        sei                                 ; Interrupts erlauben, auf Echo vom Polaroid-Sensor warten

        rcall   USN_GainDelay
        ldi     zl,LOW(GAINTAB<<1)
        ldi     zh,HIGH(GAINTAB<<1)
        push    r0
GainLoop:
        lpm
        mov     r16,r0
        adiw    zl,1
        cpi     r16,-1                      ; -1 -> Ende der Tabelle erreicht?
        breq    GainLoopEnde
        cpi     r16,99                      ; 99 -> Gain20 einschalten
        breq    Gain20
        rcall   X9C_WiperUpTo               ; Widerstand um r16 kOhm erhöhen
        rcall   USN_GainDelay               ; In 2ms den nächsten Verstärkungsfaktor holen
        rjmp    GainLoop
Gain20:
        sbi     GAIN_PORT,GAIN_20           ; Verstärkung der ersten Stufe von 1x auf 20x umschalten, sofort den nächsten Wert fürs Poti lesen und setzen
        rjmp    GainLoop
GainLoopEnde:
        pop     r0

USN_Ende:
        pop     r20
        pop     r17
        rcall   X9C_Deselect0
        sei
        ret

USN_NoPacketRecvd:
        push    r16                         ; Register sichern
        in      r16,SREG
        push    r16

        ser     r16
        sts     USN_RawLo,r16
        sts     USN_RawHi,r16
        ldi     r16,(1<<ICF1)               ; eventuell zwischengespeichertes Flag für Timer Capture IRQ löschen (wird durch Setzen auf Eins gelöscht)
        out     TIFR,r16                    ; sonst wird der Interrupt sofort ausgelöst wenn er freigegeben wird

        in      r16,TIMSK
        andi    r16,~((1<<TICIE1)+(1<<TOIE1))       ; Das TICIE1- und TOIE1-Bit löschen, Capture- und Overflow-IRQ disabled
        out     TIMSK,r16                   ; Interrupt für Zeitmessung sperren

        ldi     r16,1
        mov     USN_Semaphore,r16           ; Kennzeichnet Ende der Messung
        sbi     USN_PORT,USN_TRESHOLD       ; Kondensator für die nächste Messung schon mal laden

        pop     r16
```

```
		out	SREG,r16
		pop	r16
		reti

USN_PacketRecvd:
		push	r16
		in	r16,SREG
		push	r16

		ldi	r16,(1<<TOV1)		; eventuell zwischengespeichertes Flag für Timer Capture IRQ löschen (wird durch Setzen auf Eins gelöscht)
		out	TIFR,r16		; sonst wird der Interrupt sofort ausgelöst wenn er freigegeben wird

		in	r16,TIMSK
		andi	r16,~((1<<TICIE1)+(1<<TOIE1))	; Das TICIE1- und TOIE1-Bit löschen, Capture- und Overflow-IRQ disabled
		out	TIMSK,r16		; Interrupt für Zeitmessung sperren

		in	r16,ICR1L		; Input Capture Register auslesen (LOW)
		sts	USN_RawLo,r16
		in	r16,ICR1H		; Input Capture Register auslesen (HIGH)
		sts	USN_RawHi,r16		; und im Speicher als USN_Raw ablegen

		ldi	r16,1
		mov	USN_Semaphore,r16	; Kennzeichnet Ende der Messung

		pop	r16
		out	SREG,r16
		pop	r16
		reti

USN_Filter:
		lds	r17,USN_RawHi
		lds	r16,USN_RawLo		; in r17r16 steht der aktuelle Messwert
; ** wenn MOD_USFilter gesetzt, dann den Wert nicht direkt zurückgeben, sondern filtern
		lds	r18,Modusflags
		sbrs	r18,MOD_USFilter
		rjmp	USN_FilterBeendet
; ** kleinere Werte (als der letzte) werden sofort übernommen, grössere Werte werden tiefpassgefiltert
; ** ist r17r16 kleiner als das letzte Ergebnis? wenn ja, Wert direkt übernehmen.
		cp	r17,USN_RuntimeH
		brlo	USN_KleinererWert	; Wenn der neue Wert kleiner, dann...
		brne	USN_GroessererWert	; Wenn groesser...
		cp	r16,USN_RuntimeL	; Wenn HI gleich, LO prüfen
		brlo	USN_KleinererWert	; Neuer Wert kleiner?
		breq	USN_FilterBeendet	; Neuer Wert gleich?
USN_GroessererWert:
; ** Der neue Wert war also grösser und MOD_USFilter=1, also
; ** D(n+1) := FilterFreq * Messwert + (1-FilterFreq)*D(n)
; ** Um Rundungsfehler zu vermeiden, wird folgendermaßen gerechnet:
; ** D(n+1) := [(FilterFreq * Messwert) + (256-FilterFreq) * D(n)]/256
; ** mit FilterFreq=0..255, Messwert in r17r16
		ldi	r18,FilterFreq
		rcall	Multi16x8		; r22r21r20 := FilterFreq * Messwert
```

```
            push    r22
            push    r21
            push    r20                     ; Messergebnis auf Stack sichern

            mov     r16,USN_RuntimeL
            mov     r17,USN_RuntimeH        ; r17r16 := D(n)
            ldi     r18,FilterFreq
            com     r18
            inc     r18                     ; r18 := 256-FilterFreq
            rcall   Multi16x8               ; r22r21r20 := (256-FilterFreq) * D(n)

            pop     r16
            add     r20,r16
            pop     r16
            adc     r21,r16
            pop     r16
            adc     r22,r16                 ; Das Ergebnis (16Bit) steht in r22r21, r20 ist Rest

            mov     r17,r22
            mov     r16,r21
USN_KleinererWert:
USN_FilterBeendet:
            mov     USN_RuntimeL,r16
            mov     USN_RuntimeH,r17
            sts     USN_ValLo,r16
            sts     USN_ValHi,r17           ; Falls nicht in mm umgerechnet werden soll, steht in USN_ValLo,Hi jetzt der gefilterte Wert
            ret

USN_mm:
; ** Umwandlung des Timer-Messwertes [us] in einen Entfernungswert [mm]
;
;           Vschall*Laufzeit                    Vschall
; mm =  ---------------------- - US_Offset = ( ---------- * Laufzeit) - US_Offset
;             2 * 1000                          2000
;
; /2 wegen doppelter Schalllaufzeit (hin und zurück), /1000 wegen Laufzeit in Mikrosekunden
;
; mm = (US_Faktor * Laufzeit/1000) - US_Offset
;
; mit Vschall = 343m/s = 343mm/ms -> US_Faktor = 171,5, also 172
; und Laufzeit gemessen in Mikrosekunden, US_Offset gemessen in mm
;
;
;            US_Faktor * Laufzeit
; mm = (------------------------------) - US_Offset
;                    1000
;
; US_Faktor sei ein 16Bit-Wert (theoretisch 171.,5, experimentell ermittelt (im heißen Sommer 2003) wurde ein Wert von 174)
; Laufzeit 16Bit-Wert in ms
; US_Offset 16Bit-Wert in mm
;
; ** Erhält Timer-Wert in r17:r16, Wert in ms
```

```
            mov     r16,USN_RuntimeL        ; Timerwert (LOW)
            mov     r17,USN_RuntimeH        ; Timerwert (HIGH) r17:r16 := Timer
            ldi     r18,US_Faktor
            rcall   Multi16x8               ; r22:r21:r20 := r17:r16 * US_Faktor
; ** Jetzt durch 1000 teilen

; ** A Dividend     r22:r21:r20
; ** B Divisor      r18:r17:r16
; ** E Ergebnis     r2:r1:r0
; ** Z Zähler       r5:r4:r3 interne Verwendung

            clr     r18
            ldi     r17,HIGH(1000)
            ldi     r16,LOW(1000)
            rcall   Divide24x24
; ** Jetzt abfragen, ob der Wert kleiner als der (US_Offset+US_Totbereich) ist -> Fehler
            ldi     r17,LOW(US_Offset)
            ldi     r16,LOW(US_Totbereich)
            add     r17,r16
            push    r17
            ldi     r17,HIGH(US_Offset)
            ldi     r16,HIGH(US_Totbereich)
            adc     r17,r16
            pop     r16                     ; r17r16 := (US_Offset+US_Totbereich)

            sub     r0,r16
            sbc     r1,r17
; ** wenn das Ergebnis negativ wurde, dann Ergebnis auf US_Totbereich setzen (Messfehler)
            brpl    USN_ValueInRange
            ldi     r17,HIGH(US_Totbereich)
            sts     USN_ValHi,r17
            ldi     r16,LOW(US_Totbereich)
            sts     USN_ValLo,r16           ; Entfernung = US_Totbereich (in mm)
            rjmp    USN_mm_Ende
USN_ValueInRange:
            ldi     r17,LOW(US_Totbereich)
            mov     r16,r0
            add     r16,r17
            push    r16

            ldi     r17,HIGH(US_Totbereich) ; Totbereich wieder aufaddieren, Offset bleibt korrigiert
            mov     r16,r1
            adc     r17,r16
            pop     r16

            sts     USN_ValHi,r17
            sts     USN_ValLo,r16           ; Entfernung = Wert in mm
USN_mm_Ende:
            ret

USN_WaitMeasure:                            ; Warte auf Ende der Ultraschall-Laufzeit-Messung
USN_CheckSemaphore:
```

```
            mov     r16,USN_Semaphore
            cpi     r16,1
            brne    USN_CheckSemaphore
            ret

USN_PulseDelay:
; ** Wartedauer:   1 + r16 + 2*(r16 - 1) + 1 + 4 + 3
; **              = 6 + 3*r16
; ** bei 8MHz muss r16 mit 21 geladen werden -> 69 Taktzyklen -> 8,625usec
            ldi     r16,21                  ; 1 Takt
USN_DelayLoop:
            dec     r16                     ; 1 Takt*Anzahl
            brne    USN_DelayLoop           ; 2 Takte bei Sprung, 1 Takt bei Weiter ((2*Anzahl-1) + 1)
            ret                             ; 4 Takte für RET + 3 Takte für aufrufendes RCALL

USN_GainDelay:
; ** Gesamte Wartezeit mit rcall-Aufruf: Zyklen = 770*B + 3*A - 760
; ** für B=22 und A=207 -> 16798 Zyklen ->2099,75usec
            push    r17                     ; Zyklen: 2
            push    r16                     ; Zyklen: 2
            ldi     r17,22                  ; Zyklen: 1 (r17 ist B=22)
            ldi     r16,206                 ; Zyklen: 1 (r16 ist A=206)
USN_Warte_GAIN_InnerLoop:
            dec     r16                     ; Zyklen: A + (B-1)*256 = A + 256*B -256
            brne    USN_Warte_GAIN_InnerLoop  ; Zyklen: (A-1)*2 + 1 + (B-2)*255*2 + (B-2) = 2*A - 2 + 1 + 510*B - 510 + B - 2 = 511*B + 2*A - 513
            dec     r17                     ; Zyklen: B
            brne    USN_Warte_GAIN_InnerLoop  ; Zyklen: (B-1)*2 + 1 = 2*B - 1
            pop     r16                     ; Zyklen: 2
            pop     r17                     ; Zyklen: 2
            ret

GAINTAB:            .db 99,2,2,4,8,12,16,20,32,-1   ; Tabelle der einzelnen Gain-Werte, ACHTUNG: Keine absoluten Werte, sondern relative, beginnt bei 0

.DSEG
USN_RawLo:      .BYTE   1
USN_RawHi:      .BYTE   1               ; 16Bit-Wert USN_Raw enthält den Timerstand nach Beendigung der Messung
USN_ValLo:      .BYTE   1
USN_ValHi:      .BYTE   1               ; der in mm umgerechnete Entfernungswert
```

Multiplication16x8_lib.asm

```
; ************************************************************
; * Library:          Multiplication16x8_lib für AVR Prozessoren
; * Autor:            Oliver Dahlmann
; * Version:          1.0
; * Beschreibung:     Bibliothek mit Multiplikationsroutine
; *                   Result(24Bit) := M1(16Bit) * M2( 8Bit)
; ************************************************************
;
;
.CSEG

; Result:= M1 * M2
; M1:       r17:r16
; M2:       r18
; Result:   r22:r21:r20

Multi16x8:
            push    r23
            clr     r22
            clr     r21
            clr     r20
            ldi     r23,8
Mul16x8_1:
            lsr     r18
            brcc    Mul16x8_2
            add     r21,r16
            adc     r22,r17
Mul16x8_2:
            ror     r22
            ror     r21
            ror     r20
            dec     r23
            brne    Mul16x8_1
            pop     r23
            ret
```

Multiplication16x16_lib.asm

```
; ***************************************************************
; * Library:            Multiplication16x16_lib für AVR Prozessoren
; * Autor:              Oliver Dahlmann
; * Version:            1.0
; * Beschreibung:       Bibliothek mit Multiplikationsroutine
; *                     Result(32Bit) := M1(16Bit) * M2(16Bit)
; ***************************************************************
;
.CSEG

; Result:= M1 * M2
; M1:       r17:r16
; M2:       r19:r18
; Result:   r23:r22:r21:r20

Multi16x16:
            push    r24
            clr     r23
            clr     r22
            clr     r21
            clr     r20
            ldi     r24,16
Mul1616_1:
            lsr     r19
            ror     r18
            brcc    Mul1616_2
            add     r22,r16
            adc     r23,r17
Mul1616_2:
            ror     r23
            ror     r22
            ror     r21
            ror     r20
            dec     r24
            brne    Mul1616_1
            pop     r24
            ret
```

Division24Bit_lib.asm

```
; ************************************************************
; * Library:          Division24Bit_lib für AVR Prozessoren
; * Autor:            Oliver Dahlmann
; * Version:          1.0
; * Beschreibung:     Bibliothek mit Divisionsroutine
; *                   E(24Bit) := A(24Bit) / B(24Bit)
; ************************************************************
;

.CSEG

Divide24x24:
; PRE:
; ** A Dividend      r22:r21:r20
; ** B Divisor       r18:r17:r16
; ** E Ergebnis      r2:r1:r0
; ** Z Zähler        r5:r4:r3 interne Verwendung
; POST:
; ** E = Ergebnis, A = Rest, B = gerundetes Ergebnis
; ** Z = keine Änderung

            push    r5
            push    r4
            push    r3
            push    r19

            clr     r19
            mov     r2,r19
            mov     r1,r19
            mov     r0,r19                  ; E := 0

            cpi     r18,0
            brne    Div24_DivLoop
            cpi     r17,0
            brne    Div24_DivLoop
            cpi     r16,0
            breq    Div24_DivisionByZero    ; B=0? -> Fehler
Div24_DivLoop:
            rcall   Div24_VergleichAB
            breq    Div24_Rechnen           ; wenn A = B, wird die Rechnung einfach
            brcc    Div24_DivisionEnde      ; wenn B > A ist die Rechnung jetzt schon zu Ende, Ergebnis = 0
Div24_Rechnen:
            push    r18
            push    r17
            push    r16
            mov     r5,r19
            mov     r4,r19
            mov     r3,r19
            inc     r3                      ; Z := 1
            rjmp    Div24_Vergleichen
```

```
Div24_SchiebeLinks:
        lsl     r16
        rol     r17
        rol     r18                     ; B := B *2

        lsl     r3
        rol     r4
        rol     r5                      ; Z := Z *2

Div24_Vergleichen:
        rcall   Div24_VergleichAB
        breq    Div24_AddiereZuErgebnis ; B=A -> direkt addieren
        brcs    Div24_BPruefeGroesse    ; wenn B < A, dann Linksschieben von B, vorausgesetzt, B ist nicht zu gross

Div24_SchiebeRechts:
        lsr     r18                     ; wenn B > A, dann
        ror     r17
        ror     r16                     ; B := B / 2

        lsr     r5
        ror     r4
        ror     r3                      ; Z := Z / 2

Div24_AddiereZuErgebnis:
        add     r0,r3
        adc     r1,r4
        adc     r2,r5                   ; E := E + Z

        sub     r20,r16
        sbc     r21,r17
        sbc     r22,r18                 ; A := A - B

        pop     r16
        pop     r17
        pop     r18

        rjmp    Div24_DivLoop

Div24_BPruefeGroesse:
        sbrs    r18,7                   ; MSB von r18 (B) gesetzt? Dann maximale Verschiebung erreicht, subtrahieren
        rjmp    Div24_SchiebeLinks      ; Sprung wenn MSB nicht gesetzt
        rjmp    Div24_AddiereZuErgebnis ; Sprung wenn MSB gesetzt

Div24_DivisionByZero:
; ** Division durch Null wurde versucht
        sec                             ; Division By Zero Error -> C=1
        rjmp    Div24_Div_Ende

Div24_DivisionEnde:
; ** noch prüfen, ob aufgerundet werden muss:
; ** wenn der Rest >= (Dividend/2), dann Ergebnis++
        sbrs    r22,7                   ; oberstes Bit vom Rest gesetzt?
```

```
        rjmp    Div24_RestMal2          ; Wenn nein, für den Test den Rest*2 nehmen
Div24_DivisorDurch2:                    ; Wenn ja, für den Test den Divisor/2 nehmen
        lsr     r18
        ror     r17
        ror     r16                     ; Divisor/2
        rjmp    Div24_RestVergleichen
Div24_RestMal2:
        lsl     r20
        rol     r21
        rol     r22                     ; Rest*2
Div24_RestVergleichen:
        rcall   Div24_VergleichAB
        breq    Div24_ErgebnisAufrunden ; wenn Divisor/2=Rest -> Z=1
        brcs    Div24_ErgebnisAufrunden ; wenn Divisor/2<Rest -> C=1
        mov     r16,r0
        mov     r17,r1
        mov     r18,r2                  ; gerundetes Ergebnis = Ergebnis
        rjmp    Div24_Div_Ende          ; C=0
Div24_ErgebnisAufrunden:
        ldi     r16,1
        add     r16,r0
        clr     r17
        adc     r17,r1
        clr     r18
        adc     r18,r2                  ; gerundetes Ergebnis = Ergebnis +1
        clc                             ; C=0
Div24_Div_Ende:
; ** Division beendet
        pop     r19
        pop     r3
        pop     r4
        pop     r5
        ret                             ; Ende der Division, OK -> C=0

Div24_VergleichAB:
        cp      r18,r22                 ; HighBytes vergleichen
        breq    Div24_BGleichAHigh
        ret
Div24_BGleichAHigh:
        cp      r17,r21                 ; MidBytes vergleichen
        breq    Div24_BGleichAMid
        ret
Div24_BGleichAMid:
        cp      r16,r20                 ; LowBytes vergleichen
        ret                             ; wenn B = A -> C=0, Z=1
                                        ; wenn B > A -> C=0, Z=0
                                        ; wenn B < A -> C=1, Z=0
```

RCX_Receive_lib.asm

```
;  ************************************************************
;    Library:          RCX_Receive_lib für AVR Prozessoren
;    Autor:            Oliver Dahlmann
;    Version:          1.0
;    Beschreibung:     Bibliothek mit Empfangsroutine
;                      RCX-Brick wird über passive Bauelemente
;                      mit dem Analog-Komparator-Eingang des AVR
;                      verbunden (+). Mit dem invertierenden (-)- o +5
;                      Eingang wird ein Spannungsteiler verbunden,|
;                      so dimensioniert, das ein aktiver Sensor-  -
;                      eingang vom RCX eine 1 auf ACO zur Folge  | |
;                      hat, ein passiver Sensor-Eingang eine 0.  | |220K
;  ************************************************************ | |
;                                                               -
;                                                               |      ________ _
;                                                               |     |
;                            aktiv  3.5V _ _ ___ _ _            |     |
;                                   REF  - -/- -|- -        REF @-----|AIN1
;                            passiv       _|     \__            |     |       AVR
;                                  _____                        |     |     AT90S
;   RCX -> Diodenbrücke ->  o-----|_____|----------@---------+-----|AIN0
;                                   1M              |         |     |
;                                                   -         -     |________ _
;                                                  | |       | |       |
;                                                  | |1M     | |100K   |
;                                                  | |       | |       |
;                                                   -         -        |
;                                                   |         |        |
;                                                  ---        o       ---
;                                                               DAC
;
; Am mit DAC bezeichneten Punkt wird das Ausgangssignal des D/A-Wandlers des Distanz-
; messmoduls eingespeist (über Impedanzwandler). Hierdurch bewegt sich die Referenz-
; spannung zwischen den analogen Ausgangswert des Moduls und dem Aktiv-Signal des RCX.

.EQU    RCX_Semaphore           = 0     ; Bit in RCX_Status
.EQU    RCX_CommandRcvd         = 1

.DEF    RCX_Command             = r7    ; In diesem Byte steht die Länge des High-Pegels auf der RCX-Leitung

.CSEG

RCX_Receive:
        sbis    ACSR,ACO                ; Überspringe den nächsten Befehl wenn RCX_RECEIVE Komparator High-Pegel
        rjmp    RCX_Low
RCX_High:
        lds     r16,RCX_Status
        sbrc    r16,RCX_CommandRcvd
        ret                             ; wenn RCX_CommandRcvd=1 -> ret
        sbrc    r16,RCX_Semaphore
        rjmp    RCX_HighContinue        ; wenn RCX_Semaphore=1 -> RCX_HighContinue

        push    r16
```

```
            clr     r16
            mov     RCX_Command,r16         ; RCX_Command:=0
            pop     r16

            sbr     r16,(1<<RCX_Semaphore)
            sts     RCX_Status,r16          ; RCX_Semaphore:=1, d.h. Befehlsempfang läuft
RCX_HighContinue:
            mov     r16,RCX_Command
            inc     r16
            mov     RCX_Command,r16         ; RCX_Command++
RCX_Ende:
            ret
RCX_Low:
; ** Hier die empfangene Zeitdauer des High-Impulses quantisieren
; ** wenn 4 Befehle differenziert werden können ist dies völlig ausreichend
            lds     r16,RCX_Status
            sbrs    r16,RCX_Semaphore       ; wenn Semaphore=0->LowContinue
            rjmp    RCX_LowContinue
            cbr     r16,(1<<RCX_Semaphore)  ; RCX_Sempahore für nächsten Befehlsempfang vorbereiten
            sbr     r16,(1<<RCX_CommandRcvd) ; RCX_CommandRcvd = 1 -> Befehl wurde empfangen
            sts     RCX_Status,r16

            mov     r16,RCX_Command
            lsr     r16
            lsr     r16
            lsr     r16                     ; gemessene Dauer/8 -> Befehlsnummer 1..4
            mov     RCX_Command,r16

; ** eine 1 in RCX_Command heisst, dass ein Impuls mit einer idealen Länge von 24ms empfangen wurde
; ** eine 2 in RCX_Command heisst, 40ms
; ** eine 3 in RCX_Command heisst, 56ms
; ** eine 4 in RCX_Command heisst, 72ms
RCX_LowContinue:
            ret

RCX_Init:
            lds     r16,RCX_Status
            cbr     r16,(1<<RCX_CommandRcvd)+(1<<RCX_Semaphore)     ; CommandRcvd=0->Empfang möglich,
                                                                    ; Semaphore:=0->kein vollständiger Empfang
            sts     RCX_Status,r16
            ret

.DSEG
RCX_Status:     .BYTE   1               ; in diesem Byte steht der RCX_Status: RCX_Semaphore und RCX_CommandRcvd
```

UART8_lib.asm

```
; ************************************************************
; * Library:          UART8 für AVR Prozessoren
; * Autor:            Oliver Dahlmann
```

```
; * Version:            1.0
; * Beschreibung:      Bibliothek mit verschiedenen Unterprogrammen
; *                    zur Benutzung der seriellen Schnittstelle im
; *                    8Bit-Modus
; ************************************************************
;
; void UART8_Init(Baud:r16)              bekommt in r16 einen Wert für die Baudrate übergeben,
;                                        der UART wird auf 8Bit eingestellt, TX und RX Interrupts werden erlaubt
;                                        und die Ringpuffer werden initialisiert (Zeiger)
;
;                          r16    Baud    Abweichung%
;                          ------------------------------          f_clock
;                          207     2.400   0,2%              Baud = ------------------
;                          103     4.800   0,2%                     16 * ( r16 + 1 )
;                           51     9.600   0,2%
;                           34    14.400   0,8%
;                           25    19.200   0,2%
;                           16    28.800   2.1% !!
;                           15    31.250   0,0% (MIDI)
;                           12    38.400   0,2%
;
; void UART8_SendByte(Byte:r17)          schiebt das in r17 übergebene Byte in den Sende-Ringpuffer und startet, falls
;                                        nötig, den Sendevorgang. Befinden sich noch Bytes im Puffer, werden diese
;                                        interruptgesteuert gesendet (UART8_TXC_Handler wird automatisch aufgerufen).
;                                        Ist der Puffer n Bytes gross, können maximal n Bytes 'am Stück' versandt werden
;                                        (also direkt nacheinander).
;
; void UART8_GetByte(*Callback:Z)        prüft, ob ein neues (noch nicht verarbeitetes) Byte im Ringpuffer auf seine
; r16,r17 wird verändert                 Verarbeitung wartet. In Z ist die Adresse einer Funktion zu übergeben, die
;                                        die eigentliche Verarbeitung des Bytes sicherstellt.
;                                        UART8_GetByte ändert keine Register ausser r16/r17, immer im Hinterkopf behalten
;                                        dass die CallBack-Routine vielleicht Register verändert!
;
;                                        Beispiel für Aufruf im Hauptprogramm:
;
;                                        MAINLOOP:
;                                                ...
;                                                ldi     zl,LOW(VERARBEITUNG)
;                                                ldi     zh,HIGH(VERARBEITUNG)
;                                                rcall   UART8_GetByte
;                                                ...
;                                                rjmp    MAINLOOP
;                                        VERARBEITUNG:
;                                                ...             ; empfangenes Byte in r17
;                                                ret             ; zurück hinter den Aufruf von UART8_GetByte
;
; void UART8_SendPGMString(*Char:Z)      sendet einen im ProgramMemory befindlichen null-terminierten String
;                                        Z ist ein Zeiger auf die Anfangs-Adresse des Strings
;
; void UART8_SendRAMString(*Char:Z)      sendet einen im RAM befindlichen null-terminierten String
;                                        Z ist ein Zeiger auf die Anfangs-Adresse des Strings
```

```
.CSEG

UART8_GetByte:
; ** Diese Routine holt ein Byte aus dem als Ringpuffer implementierten Empfangspuffer der seriellen Schnittstelle
; ** und springt, falls es ein noch nicht verarbeitetes Byte im Puffer gab, an die in Z übergebene Adresse.
; ** Das zu verarbeitende Byte befindet sich zu diesem Zeitpunkt in r17 und darf verändert werden. Ebenso darf Z
; ** in der CallBack-Routine verändert werden.
; ** Das RET der in Z übergebenen Callback-Routine führt zu einem Rücksprung hinter den Aufruf von UART8_GetByte
; ** r16 wird verändert !!!
        cli

        lds     r17,UART8_RB_ReadPos
        lds     r16,UART8_RB_WritePos
        cp      r17,r16
        breq    UART8_GetByte_NoNewData

        push    zl                              ; Adresse für CallBack sichern
        push    zh
        ldi     zl,LOW(UART8_RB_START)
        ldi     zh,HIGH(UART8_RB_START)         ; Basisadresse des ReceiveBuffers
        lds     r16,UART8_RB_ReadPos            ; Der Lesezeiger innerhalb des ReceiveBuffers
        add     zl,r16
        clr     r16
        adc     zh,r16

        ld      r17,z                           ; in r17 steht das empfangene Byte

        rcall   UART8_Inc_RB_ReadPos            ; Lesezeiger erhöhen
; ** Hier irgendwas mit dem empfangenen Byte anstellen ...
; ** Also Einsprung in eine Routine, die das Byte in r17 verarbeitet
        pop     zh                              ; Adresse für CallBack wiederherstellen
        pop     zl
        sei
        ijmp                                    ; springt an die Adresse in Z (r31:r30), von dort wird direkt hinter den Aufruf UART8_GetByte zurückgesprungen
UART8_GetByte_NoNewData:
        clr     r17                             ; wenn keine neuen Daten empfangen wurden, r17=0 zurückgeben
        sei
        ret

UART8_Restart:
        push    zl
        push    zh
        cli
        lds     r16,UART8_SB_Empty
        cpi     r16,1                           ; Wenn 1 in SB_Empty, ist der SB leer, es werden keine Daten gesendet, also TX_Interrupt auslösen
        breq    UR_PufferLeer
; ** Hier angekommen war SB_Empty=0, also Zeichen im Puffer vorhanden
        ldi     zl,LOW(UART8_SB_START)
        ldi     zh,HIGH(UART8_SB_START)
        lds     r16,UART8_SB_ReadPos
        add     zl,r16
        clr     r16
```

```
        adc     zh,r16
        ld      r17,z                   ; Hole Zeichen aus Ringpuffer
        sei                             ; Interrupts wieder freigeben
        rjmp    UART8_Relaunch          ; Manuell senden, Interrupts in Gang bringen...
UR_PufferLeer:
        pop     zh
        pop     zl
        sei
        ret

UART8_SendByte:
;** Diese Routine schreibt das in r17 übergebene Byte in den Sendepuffer (Ringpuffer)
;** der über den Interrupt automatisch gesendet wird
;** r16 wird verändert !!!

        push    zl
        push    zh

        cli                             ; *** Critical Section, Interrupts sperren ***
        ldi     zl,LOW(UART8_SB_START)
        ldi     zh,HIGH(UART8_SB_START)
        lds     r16,UART8_SB_WritePos
        add     zl,r16
        clr     r16
        adc     zh,r16                  ; Z := Z + SB_WritePos
        st      z,r17                   ; Dann das Byte im Ringpuffer speichern

        rcall   UART8_Inc_SB_WritePos
        lds     r16,UART8_SB_WritePos
        push    r17
        lds     r17,UART8_SB_ReadPos
        cp      r16,r17
        pop     r17
        brne    UART8_BufferNotFull
UART8_SB_BufferFull:
        rcall   UART8_Dec_SB_WritePos
        lds     r16,UART8_SB_ReadPos
        sei                             ; *** Ende Critical Section Weg1, Interrupts freigeben ***
        push    r17
UART8_SB_WaitBufferFree:
        lds     r17,UART8_SB_ReadPos
        cp      r16,r17
        breq    UART8_SB_WaitBufferFree
        pop     r17
        rcall   UART8_Inc_SB_WritePos   ; Erhöhe Schreibzeiger um ein Byte
UART8_BufferNotFull:
        sei                             ; *** Ende Critical Section Weg2, Interrupts freigeben ***
        lds     r16,UART8_SB_Empty
        cpi     r16,1                   ; Wenn 1 in SB_Empty, ist der SB leer, es werden keine Daten gesendet, also TX_Interrupt auslösen
        brne    UART8_SendevorgangLaeuft

        clr     r16
```

```
        sts     UART8_SB_Empty,r16              ; SB_Empty auf 0 setzen
UART8_Relaunch:
; ** Hier einspringen nachdem der TXIE-Interrupt gesperrt wurde, vorher zl, zh pushen!
        out     UDR,r17                         ; Zeichen senden (Das erste Zeichen hier manuell, Rest folgt per IRQ)
        rcall   UART8_Inc_SB_ReadPos            ; Erhöhe Lesezeiger um ein Byte

UART8_SendevorgangLaeuft:
        pop     zh
        pop     zl
        ret

UART8_Inc_SB_WritePos:
        lds     r16,UART8_SB_WritePos
        inc     r16
        cpi     r16,(UART8_SB_END-UART8_SB_START)       ; Ist der Schreibzeiger auf 16, dann setze ihn auf 0 (Ringpuffer-Prinzip)
        brne    UART8_SB_NeuerRingzeigerWrite
        clr     r16
UART8_SB_NeuerRingzeigerWrite:
        sts     UART8_SB_WritePos,r16
        ret

UART8_Dec_SB_WritePos:
        lds     r16,UART8_SB_WritePos
        dec     r16
        cpi     r16,-1                          ; Ist der Schreibzeiger auf -1, dann setze ihn auf 16 (Ringpuffer-Prinzip)
        brne    UART8_Dec_SB_NeuerRingzWrite
        ldi     r16,((UART8_SB_END-UART8_SB_START)-1)
UART8_Dec_SB_NeuerRingzWrite:
        sts     UART8_SB_WritePos,r16
        ret

UART8_Inc_SB_ReadPos:
        lds     r16,UART8_SB_ReadPos
        inc     r16
        cpi     r16,(UART8_SB_END-UART8_SB_START)       ; Ist der Lesezeiger auf 16, dann setze ihn auf 0 (Ringpuffer-Prinzip)
        brne    UART8_SB_NeuerRingzeigerRead
        clr     r16
UART8_SB_NeuerRingzeigerRead:
        sts     UART8_SB_ReadPos,r16
        ret

UART8_Inc_RB_WritePos:
        lds     r16,UART8_RB_WritePos
        inc     r16
        cpi     r16,(UART8_RB_END-UART8_RB_START)       ; Ist der Schreibzeiger auf 16, dann setze ihn auf 0 (Ringpuffer-Prinzip)
        brne    UART8_RB_NeuerRingzeigerWrite
        clr     r16
UART8_RB_NeuerRingzeigerWrite:
        sts     UART8_RB_WritePos,r16
        ret

UART8_Inc_RB_ReadPos:
```

```
        lds     r16,UART8_RB_ReadPos
        inc     r16
        cpi     r16,(UART8_RB_END-UART8_RB_START)       ; Ist der Lesezeiger auf 16, dann setze ihn auf 0 (Ringpuffer-Prinzip)
        brne    UART8_RB_NeuerRingzeigerRead
        clr     r16
UART8_RB_NeuerRingzeigerRead:
        sts     UART8_RB_ReadPos,r16
        ret

UART8_SendPGMString:                                    ; Z: Zeiger auf String im ProgramMemory
        push    r0
        push    r17
        push    zl
        push    zh
UART8_SP_Kopiere:
        lpm                                             ; liest aus (Z) -> r0
        mov     r17,r0
        cpi     r17,0                                   ; Null Terminierung gefunden?
        breq    UART8_SP_NullGefunden
        rcall   UART8_SendByte                          ; Das Byte in r17 wird gesendet (UART)
        adiw    zh:zl,1                                 ; inc(Z)
        rjmp    UART8_SP_Kopiere
UART8_SP_NullGefunden:
        pop     zh
        pop     zl
        pop     r17
        pop     r0
        ret

UART8_SendRAMString:                                    ; Z:  Zeiger auf String im RAM
        push    r0
        push    r17
        push    zl
        push    zh
UART8_SR_Kopiere:
        ld      r17,z+                                  ; liest aus (Z) -> r17
        cpi     r17,0                                   ; Null Terminierung gefunden?
        breq    UART8_SR_NullGefunden
        rcall   UART8_SendByte                          ; Das Byte in r17 wird gesendet (UART)
        rjmp    UART8_SR_Kopiere
UART8_SR_NullGefunden:
        pop     zh
        pop     zl
        pop     r17
        pop     r0
        ret

UART8_Init:
        out     UBRR,r16                                ; Baudrate wird in r16 übergeben (25 ergibt 19.200 Baud bei 8 MHz)
        ldi     r16,(1<<RXCIE)+(1<<TXCIE)+(1<<RXEN)+(1<<TXEN)
        out     UCSRB,r16                               ; Sende- und Empfangs-Interrupts aktivieren
```

```
        clr     r16
        sts     UART8_SB_WritePos,r16       ; Schreibzeiger in den Sendepuffer (wird von der Sende-Routine aktualisiert)
        sts     UART8_SB_ReadPos,r16        ; Lesezeiger in den Sendepuffer (wird vom Sende-IRQ aktualisiert)
        sts     UART8_RB_WritePos,r16
        sts     UART8_RB_ReadPos,r16

        ldi     r16,1
        sts     UART8_SB_Empty,r16          ; SB_Empty=1 -> Der Sendepuffer ist leer
        ret

UART8_RXC_Handler:                          ; aufzurufen vom UART_RXC_Interrupt
; ** ein Byte wurde von der seriellen Schnittstelle empfangen und hier in den Ringpuffer geschrieben
        push    zh
        push    zl
        push    r16
        in      r16,SREG
        push    r16
        ldi     zl,LOW(UART8_RB_START)
        ldi     zh,HIGH(UART8_RB_START)     ; Basisadresse des ReceiveBuffers
        lds     r16,UART8_RB_WritePos       ; Der Schreibzeiger innerhalb des ReceiveBuffers
        add     zl,r16
        clr     r16
        adc     zh,r16
        in      r16,UDR
        st      z,r16                       ; Das Byte in den Empfangspuffer schreiben
        rcall   UART8_Inc_RB_WritePos       ; Schreibzeiger für den Empfangspuffer erhöhen
        pop     r16
        out     SREG,r16
        pop     r16
        pop     zl
        pop     zh
        reti

UART8_TXC_Handler:
;** Wenn noch Bytes im Ringpuffer sind (SB_WritePos <> SB_ReadPos), wird das nächste gesendet
        push    r16
        in      r16,SREG
        push    r16
        push    r17
        push    zl
        push    zh
        lds     r16,UART8_SB_WritePos
        lds     r17,UART8_SB_ReadPos
        cp      r16,r17
        breq    UART8_AlleBytesGesendet
        ldi     zl,LOW(UART8_SB_START)
        ldi     zh,HIGH(UART8_SB_START)
        lds     r16,UART8_SB_ReadPos
        add     zl,r16
        clr     r16
        adc     zh,r16
        ld      r16,z                       ; Hole Zeichen aus Ringpuffer
```

```
        out     UDR,r16                 ; Und senden
        rcall   UART8_Inc_SB_ReadPos
        rjmp    UART8_TXC_Ende          ; Springe zum Ende des IRQ's. Ein neuer IRQ wird ausgelöst, wenn das Byte gesendet wurde
UART8_AlleBytesGesendet:
        ldi     r16,1
        sts     UART8_SB_Empty,r16      ; SB_Empty auf 1 setzen -> Sendepuffer ist leer
UART8_TXC_Ende:
        pop     zh
        pop     zl
        pop     r17
        pop     r16
        out     SREG,r16
        pop     r16
        reti

.DSEG

UART8_SB_START:         .BYTE   16      ; Sendepuffer
UART8_SB_END:

UART8_RB_START:         .BYTE   2       ; Empfangspuffer (Grösse frei veränderbar, max. 256)
UART8_RB_END:

UART8_RB_WritePos:      .BYTE   1
UART8_RB_ReadPos:       .BYTE   1
UART8_SB_WritePos:      .BYTE   1
UART8_SB_ReadPos:       .BYTE   1
UART8_SB_Empty:         .BYTE   1
```

X9C104_lib_short.asm

```
; ************************************************************
; * Library:              X9C104_lib_short für AVR Prozessoren
```

```
; * Autor:              Oliver Dahlmann
; * Version:            1.0
; * Beschreibung:       Bibliothek mit verschiedenen Unterprogrammen
; *                     zur Nutzung eines X9C104 (Xicor) Digital-
; *                     Potentiometers (3-Draht-Bus, 100Stufen)
; *                     weniger Funktionsumfang als die X9X104_lib
; ************************************************************
;
; keine der Routinen verändert Register

.CSEG

X9C_Reset:
; ** Schleifer auf 0, dann Standby-Mode
        push    r16
        ldi     r16,100
        rcall   X9C_WiperDownTo         ; 100 Schritte runter, also RW,RL=0 Ohm, RW,RH=100kOhm
        pop     r16
        ret

X9C_Select0:
        cbi     POT_PORT,POT_CS0        ; selektiere Poti0
        ret

X9C_Deselect0:
        sbi     POT_PORT,POT_CS0        ; selektiere Poti0
        ret

X9C_WiperUpTo:
; ** Schleifer r16 Stufen hoch
        push    r16
        sbi     POT_PORT,POT_UD         ; Up
        cpi     r16,0
        breq    X9C_WUT_Null
X9C_WUT:
        sbi     POT_PORT,POT_INC
        cbi     POT_PORT,POT_INC        ; fallende Flanke -> StepUp
        dec     r16
        brne    X9C_WUT
X9C_WUT_Null:
        pop     r16
        ret

X9C_WiperDownTo:
; ** Schleifer r16 Stufen runter
        push    r16
        cbi     POT_PORT,POT_UD         ; Down
        cpi     r16,0
        breq    X9C_WDT_Null
X9C_WDT:
        sbi     POT_PORT,POT_INC
        cbi     POT_PORT,POT_INC        ; fallende Flanke -> StepDown
```

```
        dec     r16
        brne    X9C_WDT
X9C_WDT_Null:
        pop     r16
        ret
```

SensorVerarbeitung_lib.asm

```
; *************************************************************
```

```
; * Library:          SensorVerarbeitung_lib
; * Autor:            Oliver Dahlmann
; * Version:          1.0
; * Beschreibung:     Diese Bibliothek verknüpft einen
; *                   Ultraschall-Messwert, einen Infrarot-Messwert
; *                   und den Zustand des Stossstangenkontaktes zu
; *                   einem wahrscheinlichen Entfernungswert
; ****************************************************************
;
; Pre-Condition:
; IR-Wert in mm             : r19:r18
; US-Wert in mm             : r21:r20

; Post-Condition:
; Rückgabewert in mm        : r23:r22
; Differenz                 : r17:r16
; Status                    : r24           (1->IR, 2->US)

.EQU    VertrauеIRMax           =       700     ; unterhalb dieses Wertes (Rohdaten vom ADC) sind die Werte des IR-Sensors wenig vertrauenswürdig
.EQU    VertrauеUSMin           =       140     ; unterhalb dieses Wertes sind die Ultraschalldaten unglaubwürdig
.EQU    USIR_Verschiebung       =       100     ; Verschiebung des Wertes um n mm nach oben, sofern dieser nicht über den Stossstangentaster ermittelt wurde

.EQU    VVA_STATUS_BUMPER       =       3
.EQU    VVA_STATUS_US           =       2
.EQU    VVA_STATUS_IR           =       1

.CSEG

SensorVerarbeitung:
; ** Prüfen, welcher Wert kleiner ist, IR oder US
; ** IR-Wert in mm          : r19:r18
; ** US-Wert in mm          : r21:r20
; ** Ist US > IR ?
        cp      r21,r19
        brlo    SenV_US                 ; IRhigh > UShigh
        brne    SenV_IR                 ; IRhigh < UShigh
; ** USH=IRH, also Lowbytes vergleichen
        cp      r20,r18
        brsh    SenV_IR                 ; UShigh = IRhigh, USlow > IRlow
SenV_US:                                ; US <= IR
; ** IR >= US
        push    r19                     ; IR Wert sichern
        push    r18

        sub     r18,r20
        sbc     r19,r21                 ; Diff:=IR-US
        mov     r17,r19
        mov     r16,r18                 ; Differenz in r17r16 absichern, wird von Debug-Routine ausgegeben

        pop     r18
        pop     r19                     ; IR Wert wiederherstellen
        rjmp    SenV_returnUS
```

```
SenV_IR:
; ** US > IR
        push    r21                     ; US Wert sichern
        push    r20

        sub     r20,r18
        sbc     r21,r19
        mov     r17,r21
        mov     r16,r20                 ; Differenz in r17r16 absichern, wird von Debug-Routine ausgegeben

        pop     r20
        pop     r21                     ; US Wert wiederherstellen
; ** jetzt prüfen, ob der Infrarotwert > max. mm, dann Ultraschallwert benutzen
; ** sonst IR-Wert übergeben
        push    r21
        push    r20                     ; US Wert absichern
        ldi     r20,LOW(VertrauenIRmax)
        ldi     r21,HIGH(VertrauenIRmax)
        sub     r20,r18
        sbc     r21,r19                 ; (VertrauenIRmax-IR)
        pop     r20
        pop     r21                     ; US Wert wiederherstellen
        brpl    SenV_returnIR
SenV_returnUS:
; ** somit wird der Ultraschallwert übergeben
        mov     r23,r21
        mov     r22,r20
        ldi     r24,VVA_STATUS_US
        rjmp    SenV_Bumper
SenV_returnIR:
; ** Der Infrarotwert war im vertrauenswürdigen Messbereich, daher wird der IR-Wert zurückgegeben
        mov     r23,r19
        mov     r22,r18
        ldi     r24,VVA_STATUS_IR
SenV_Bumper:
; ** Als letztes die Kontakte in der Stossstange abfragen
        sbi     PORTC,BUMPER_LEFT
        sbi     PORTC,BUMPER_RIGHT      ; Pullup-Widerstände aktivieren
        in      r18,PINC
        andi    r18,((1<<BUMPER_LEFT)+(1<<BUMPER_RIGHT))
        cpi     r18,0                   ; 0 wenn offen, 1 wenn geschlossen
        breq    SenV_Ende               ; wenn Bumper Kontakt -> Ausgabewert=0mm
SenV_BumperKontakt:
        clr     r23
        clr     r22                     ; Rückgabewert = 0
        ldi     r24,VVA_STATUS_BUMPER
        ret
SenV_Ende:
; ** hier angekommen, wurde ein Wert vom US- oder IR-Sensor ermittelt
; ** die Taster in der Stossstange haben KEINE Berührung signalisiert
; ** Damit der RCX deutlich zwischen durch Zusammenstoss ermittelten und
; ** durch US oder IR ermittelten d=0 unterscheiden kann, wird hier auf den
```

```
; ** Rückgabe-Wert ein Offset in Höhe von n mm, spezifiziert durch 'USIR_Verschiebung'
; ** am Beginn dieser Datei, addiert.
        ldi     r18,LOW(USIR_Verschiebung)
        add     r22,r18
        ldi     r18,HIGH(USIR_Verschiebung)
        adc     r23,r18
        ret
```

RCX_Output_lib.asm

```
; ************************************************************
; * Library:          RCX_Output_lib für AVR Prozessoren
```

```
; * Autor:              Oliver Dahlmann
; * Version:            1.0
; * Beschreibung:       Bibliothek mit Ausgaberoutine für Lego-RCX-
; *                     Ansteuerung
; *                     Ein auszugebender 16Bit-Wert wird skaliert
; *                     und im Nullpunkt verschoben um den Erfassungs-
; *                     bereich des Lego RCX-Eingangs so weit wie
; *                     möglich auszunutzen.
; ***************************************************************

; Pre-Condition:
; r17:r16   ..        16Bit-Wert in mm für die Ausgabe, 0000 - 5000
; Post-Condition:
; r17:r16   ..        16Bit-Wert für D/A-Wandler
;
; Experimentell ermittelt: RCX 0x4200 entspricht einem DAC-Wert von 7200, RCX 0xFF40 entspricht einem DAC-Wert von 55000
; Veränderung nahezu linear
;
; DAC = DacMin + ((DacMax - DacMin) / Wertebereich) * Wert
;
; Mit einem Wertebereich von 0 .. 2000 mm lautet die Gleichung also:
;                                   7200 + ( 47800 / 5000 ) * Wert
;
; Rechenreihenfolge um Rundungsfehler zu vermeiden:
; E1=((DacMax-DacMin)*256)/Wertebereich, E2=E1*Wert, E3=DacMax-(E2/256)
;
; Wenn der übergebene Wert den Wertebereich übersteigt, wird Wert:=MAX(Wertebereich)
.EQU    DacMin          =       7200
.EQU    DacMax          =       55000

.CSEG

RCX_Output:
; ** in r19r18 wird der Wertebereich übergeben (z.B. 5000 für 0..5000mm)
; ** in r17r16 wird der Wert in mm übergeben
        push    r24
        push    r23
        push    r22
        push    r21
        push    r20

        cp      r17,r19                 ; HighBytes vergleichen, Wert > Wertebereich?
        breq    RCXO_GleichHigh
        rjmp    RCXO_WerteUngleich
RCXO_GleichHigh:
        cp      r16,r18                 ; LowBytes vergleichen
                                        ; wenn MAX(Wertebereich) = Wert -> C=0, Z=1
                                        ; wenn MAX(Wertebereich) < Wert -> C=0, Z=0
                                        ; wenn MAX(Wertebereich) > Wert -> C=1, Z=0
RCXO_WerteUngleich:
        brcs    RCXO_WertGueltig
        mov     r17,r19
```

```
        mov     r16,r18                 ; da der Wert den Wertebereich überstieg, auf Maximum setzen
RCXO_WertGueltig:
        push    r17
        push    r16
        mov     r17,r19
        mov     r16,r18
        ldi     r22,HIGH(DacMax-DacMin)
        ldi     r21,LOW(DacMax-DacMin)
        clr     r20
        clr     r18
        rcall   Divide24x24
        mov     r19,r1
        mov     r18,r0
        pop     r16
        pop     r17
        rcall   Multi16x16
        ldi     r17,HIGH(DacMin)
        ldi     r16,LOW(DacMin)
        add     r16,r21
        adc     r17,r22
        pop     r20
        pop     r21
        pop     r22
        pop     r23
        pop     r24
        ret
```

EEPROM_lib.asm

```
; ************************************************************
```

```
; * Library:          EEPROM_lib für AVR Prozessoren
; * Autor:            Oliver Dahlmann
; * Version:          1.0
; * Beschreibung:     Bibliothek mit Zugriffsroutinen für das
; *                   interne EEEPROM, basierend auf Aplication
; *                   Note AVR100: Accessing the EEPROM von Atmel
; ************************************************************
;
; Achtung: r16 wird von EEPROM_Write und EEPROM_Read zerstört
.CSEG

EEPROM_Write:                       ; vor dem Aufruf r16=Databyte, r17=Adress
        sbic    EECR,EEWE
        rjmp    EEPROM_Write        ; Warte, bis EEWE=0
        out     EEAR,r17
        out     EEDR,r16
        cli
        sbi     EECR,EEMWE          ; Master Write Enable
        sbi     EECR,EEWE           ; Write Strobe
        sei
        ret

EEPROM_Read:                        ; vor dem Aufruf r16=Adress, nach dem Aufruf r16=Data
        sbic    EECR,EEWE
        rjmp    EEPROM_Read         ; Warte, bis EEWE=0, falls direkt nach EEPROM_Write aufgerufen
        out     EEAR,r16            ; Adresse ausgeben
        sbi     EECR,EERE           ; Read Strobe
        in      r16,EEDR
        ret
```