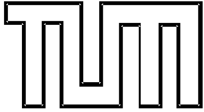


LEHRSTUHL FÜR REALZEIT–COMPUTERSYSTEME
TECHNISCHE UNIVERSITÄT MÜNCHEN
UNIV.–PROF. DR.–ING. G. FÄRBER

Bau eines elektronischen Magnetkompasses und Untersuchungen zur Verwendbarkeit bei mobilen Robotersystemen

Robin Gruber

Studienarbeit

Bau eines elektronischen Magnetkompasses und Untersuchungen zur Verwendbarkeit bei mobilen Robotersystemen

Studienarbeit

Ausgeführt am Lehrstuhl für Realzeit–Computersysteme
Technische Universität München
Prof. Dr.–Ing. Georg Färber

Betreuer: Georg Passig, RCS

Bearbeiter: Robin Gruber
Lindenschmitstr. 29a
81371 München

Eingereicht im Januar 2000

Zusammenfassung

Der elektronische Magnetkompaß mit serieller Schnittstelle, der im Rahmen dieser Arbeit konstruiert wurde (Pläne und Nachbauanleitung finden sich im Anhang) erreicht trotz seines geringen Preises (Bauteilekosten von ca. 50 Euro) eine gute Genauigkeit (Fehler < ±2° im homogenen Magnetfeld, mit Kalibrierung) bei hoher Meßfrequenz (> 10 Messungen / s). Außerdem hat sich gezeigt, daß Störungen, die durch in der Nähe befindliche Eisen– oder Stahlteile, sowie statische Störfelder, etwa durch einen Lautsprecher mit Hilfe der Kalibrierung gut kompensiert werden können, solange die Störamplitude nicht größer ist als die Erdmagnetfeldamplitude.
Selbstverständlich kann eine Kompaß nur so gut sein wie das ihn umgebende Magnetfeld. In Gebäuden moderner Architektur, die sehr viel Stahl enthalten (wie z.B. der Neubau im TU–Innenhof) kann das Magnetfeld teilweise extrem inhomogen sein. Abweichungen von über 45° wurden stellenweise beobachtet. Ein normaler Einsatz des Kompasses unter diesen Bedingungen kann gerade mal als grobe Abschätzung für die Orientierung eines Objekts herangezogen werden, etwa wenn eine andere Messung zwei mögliche Werte liefert, von denen der richtige bestimmt werden muß.
Eine Möglichkeit für die Verbesserung der Messung trotz inhomogenem Magnetfeld wird in Abschnitt 6 angesprochen, sie ist aber nicht Inhalt dieser Arbeit.

Inhalt

0 Anwendungsbereiche für einen elektonischen Kompaß......................1
1 Meßprinzip..2
1.1 Erdmagnetfeldmessung ohne Störeinflüsse..2
1.2 Unterdrückung von Störungen...2
2 Funktion der Schaltung...4
2.1 Magnetfeldsensoren...4
2.2 der verwendete Microcontroller..4
2.3 Schaltungsbeschreibung...5
2.3.1 Stromversorgung..5
2.3.2 Erzeugung der Ummagnetisierungspulse...5
2.3.3 Verstärkung des Sensorsignals..6
2.3.4 Prozessorperipherie..7
2.3.5 Kompensationsmaßnahmen..7
3 Beschreibung der Firmware...8
3.1 Port– und Variablendefinitionen..8
3.2 Initialisierung des Microcontrollers..9
3.3 Meßroutine...9
3.4 Senderoutine...10
3.5 Empfangsroutine..10
4 Software...11
4.1 Kompassbilbliothek..11
4.2 Programm zur Kalibrierung...13
4.3 Testprogramm...14
5 Meßergebnisse...15
6 Ausblicke..17

Anhang
A Anleitung zum Aufbau der Schaltung..18
A.1 Zum Aufbau benötigte Dateien, Programme und Geräte.........................18
A.2 Eigenbau einer doppelseitigen Platine..18
A.3 Bohrdurchmesser..19
A.4 Bestückung..19
A.5 Inbetriebnahme..19
B Bedienung des Assemblers und des In−System−Programmers..........20
C Kalibrierung..21
C.1 Abgleich auf gleiche Sensoramplitude...21
C.2 Abgleich am Objekt...21
D Schaltplan...22
E Bestückungsplan...23
F Stückliste...24
G Quellcode der Firmware..25
H Quellcode der Software...31
H.1 Headerfile: marcie.h..31
H.2 Kompaßbibliothek: marcie.cc...32
H.3 Testsoftware: marcietest.cc..35
H.2 Kalibrierungssoftware: marciecalib.cc...37
I Technische Daten..40

Literatur...41

0 Anwendungbereiche für einen elektronischen Kompaß

Die meisten Positions– und Orientierungssensoren, die bei mobilen Robotern Verwendung finden, besitzen den Nachteil, daß sie inkrementaler Natur sind und daher im Laufe einer Bewegung Fehler aufsummieren. Eine einfache Möglichkeit für die Bestimmung der absoluten Orientierung eines Objektes stellt ein Kompaß dar. Elektronische Kompasse werden derzeit allerdings kaum in mobilen Robotern eingesetzt, da die Meinung vorherrschend ist, daß die magnetischen Störfelder, die ein derartiger Roboter produziert, eine Messung unmöglich machen.
Ziel dieser Studienarbeit ist der Bau eines störungstoleranten, preiswerten, elektronischen Kompasses mit serieller Schnittstelle und die Untersuchung, unter welchen Bedingungen der Einsatz eines derartigen Gerätes bei mobilen Robotern sinvoll ist.

1 Meßprinzip

1.1 Erdmagnetfeldmessung ohne äußere Störeinflüsse

Zur Bestimmung der absoluten Orientierung mit Hilfe des Erdmagnetfelds sind zwei, aufeinander möglichst senkrecht stehende Magnetfeldsensoren erforderlich. Dreht man diese Sensoranordnung um die vertikale Achse, und setzt man weiterhin Proportionalität zwischen der Feldstärke und der Sensorspannung voraus, so zeigen die Sensorspannungen einen cosinus– bzw. sinusförmigen Verlauf, da durch die Sensoren die x– und die y–Komponenten des Magnetfelds erfaßt werden (Abb. 1.1).
Sind nun die Sensorspannungen und die Anordnung der Sensoren bekannt, so kann daraus eindeutig der Winkel zwischen der Sensoranordnung und den Feldlinien des Erdmagnetfelds bestimmt werden.
Zu beachten ist dabei nur, daß der geographische Nordpol einen magnetischen Südpol darstellt.

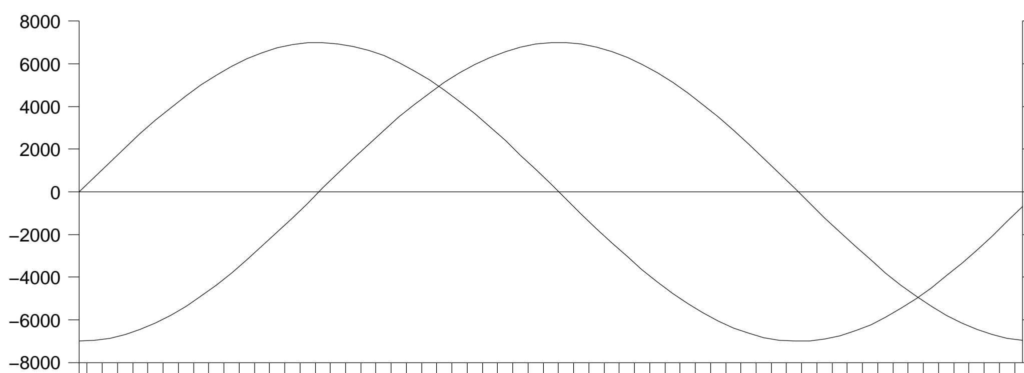


Abb. 1.1: theoretischer Verlauf der Sensorspannungen bei einer Drehung um 360°

1.2 Unterdrücken von Störungen

Im allgemeinen ist das Magnetfeld jedoch nicht so homogen, wie man es sich für eine präziese Messung wünschen würde. Zum einen ist der Kompaß normalerweise an einem Gerät montiert, das in relativ geringer Entfernung zum Magnetfeldsensor einige ferromagnetische oder sogar magnetisierte Materialien enthält, zum anderen werden häufig durch größere Mengen an Eisen oder Stahl im Umfeld des Kompasses die Feldlinien des Erdmagnetfelds verbogen (Abb. 1.2).

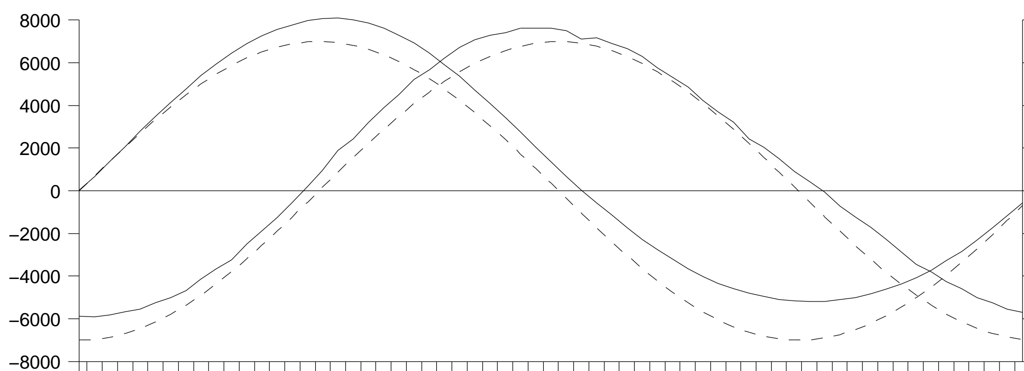


Abb. 1.2: tatsächlicher Verlauf der Sensorspannungen bei einer Drehung um 360° (die idealen Werte sind gestrichelt gezeichnet)

Meßfehler, die durch das Objekt, an dem der Sensor montiert ist verursacht werden, sind leicht durch eine Kalibrierung zu beseitigen. Bei dieser Art der Kalibrierung wird das äußere Magnetfeld als homogen vorausgesetzt.
Zunächst rotiert man das Objekt, an dem der Kompaß fest angebracht ist, mit konstanter Drehgeschwindigkeit etwas mehr als zweimal um seine vertikale Achse. Gleichzeitig werden in festen Zeitintervallen die Sensordaten erfaßt.
Zur Bestimmung der Periodendauer, also der Anzahl der Meßwerte pro Umdrehung wird eine Autokorrelation der Meßwerte durchgeführt und das zweite Minimum der Korrelationsfunktion bestimmt. Nun wird der Punkt der Meßreihe ermittelt, an dem der Wert des Ost–West–Sensors sein Betragsminimum erreicht und der Nord–Süd–Sensor einen negativen Meßwert liefert. Dieser Punkt wird als Norden angenommen. Sollte dieser Wert von der tatsächlichen Nordrichtung abweichen, so ist diese Abweichung gesondert zu bestimmen und als Konstante der Kalibrierung beizufügen.
Der nächste Schritt besteht darin, die Mittelwerte des Nord–Süd– und des Ost–West–Sensors über eine Periode zu bestimmen. Da diese sinus– bzw. cosinusförmigen Verlauf zeigen sollten, müßte die Mittelwertbildung null ergeben. Tritt dennoch ein Gleichanteil auf, so ist anzunehmen, daß dieser durch äußere Störfelder verursacht wird. Die Meßreihe wird nun von diesem Gleichanteil befreit. Anschließend werden die Nord–Süd– und die Ost–West–Komponenten in Winkel und Amplitudenwerte umgerechnet.
Zuletzt wird eine Tabelle angelegt, die angefangen mit dem Norden–Wert, jedem gemessenen Winkel einen realen Winkel zuweist. Dieser reale Winkel wird bestimmt, indem eine Periode der gemessenen Winkel in gleiche Teile zerlegt und jedem Teil ein seiner Position entsprechender realer Winkel zugewiesen wird. Außerdem wird die Amplitude des Magnetfelds für jeden gemessenen Winkel gespeichert.
Ist diese Kalibriertabelle einmal angelegt, so kann für jeden gemessen Winkel der reale Winkel bestimmt werden. Dazu werden zunächst die gemessenen Nord–Süd– und Ost–West–Werte vom Gleichanteil befreit, dann daraus der Winkel berechet und dieser anhand der Kalibiertabelle in den realen Winkel umgerechnet, wobei zwischen zwei Einträgen der Kalibriertabelle linear interpoliert wird.

Außer den Verzerrungen im Magnetfeld, die durch das Objekt, an dem der Sensor befestigt ist entstehen, gibt es auch die Möglichkeit, daß das Magnetfeld bereits im Umfeld des Kompasses inhomogen ist. Hieraus resultierende Meßfehler sind nicht mehr durch eine einfache Kalibrierung zu beseitigen. Allerdings stellt die Differenz zwischen der gemessenen und der in der Kalibriertabelle gespeicherten Amplitude eine Abschätzung für die Qualität der Messung dar. Sind gepeicherte und gemessene Amplitude gleich, so ergibt sich ein Qualitätswert von 1, weichen beide Werte voneinander ab, so sinkt dieser Qualitätswert. Dabei ist zu beachten, daß ein niedriger Qualitätswert ein Zeichen für eine schlechte Messung ist, aber ein hoher Qualitätswert keine gute Messung garantieren kann. Das ist einfach nachzuvollziehen, wenn man sich eine äußere Störquelle vorstellt, die das Erdmagnetfeld dreht, ohne seine Amplitude zu beeinflussen.
Vielmehr wäre eine Karte der magnetischen Feldlinien für den Bereich, den der Roboter befährt notwendig, um präzise Meßergebnisse zu erhalten.
Die Notwendigkeit einer derarigen Magnetfeldkarte muß allerdings nicht zwangsläufig einen Nachteil darstellen. Ist das Muster der Feldlinien erst einmal bekannt, so kann es, ähnlich einer Bodenmarkierung, zur Positionsbestimmung herangezogen werden.

2 Funktion der Schaltung

2.1 Die Magnetfeldsensoren

Das Herzstück der Schaltung bilden zwei integrierte Schaltkreise vom Typ KMZ 51 (Philips) [Phil]. Der KMZ 51 enthält bereits alle Komponenten, die für eine hochempfindliche Magnetfeldmessung benötigt werden, wie eine magnetoresistive Brücke, eine Kompensationsspule (in der Schaltung nicht verwendet) und eine Spule zur Ummagnetisierung der Brücke.
Letztere, im weiteren als Flip–Spule bezeichnet, erfüllt zwei Aufgaben. Zum einen dient sie dazu, die für die hohe Empfindlichkeit des Sensors notwendige Vormagnetisierung auch nach Einwirkung magnetischer Störeinflüsse wiederherzustellen, zum anderen stellt sie eine Möglichkeit dar, den Offset eines nachgeschalteten Differenzverstärkers zu eliminieren. Durch die Flip–Spule kann die Orientierung der Sensorvormagnetisierung und damit auch die Empfindlichkeit des Sensors umgedreht werden. Das heißt, je nach Vorzeichen der Magnetisierung liefert der Sensor ein zur Feldstärke positiv proportionales oder negativ proportionales Signal.

S sei die Empfindlichkeit des Sensors, H die Feldstärke des Erdmagnetfelds, U_s die Sensorspannung

$U_{s+}=S*H$ (bei positiver Magnetisierung)
oder
$U_{s-}=-S*H$ (bei negativer Magnetisierung) (2.1).

Wird dieses Signal nun einem offsetbehafteten Differenzverstärker zugeführt, so liefert dieser:

a sei der Verstärkungsfaktor, d die Offsetspannung

$U_{a+}=S*H*a+d*a$ oder $U_{a-}=-S*H*a+d*a$ (2.2)

Sorgt man nun dafür, daß je ein Meßwert bei positiver und bei negativer Magnetisierung aufgenommen und aus beiden die Differenz gebildet wird, so erhält man:

$U_a=U_{a+}-U_{a-}=S*H*A+d*a-(-S*H*a+d*a)=2*S*H*a$ (2.3)

Obige Überlegungen sind natürlich nur dann richtig, wenn der Verstärker trotz des Offsets im linearen Bereich arbeitet und der Offsetdrift zwischen positiver und negativer Messungen vernachlässigbar klein ist. Der Einsatz eines Differenzverstärkers mit niedriger Offsetspannung ist also trotzdem erforderlich.

2.2 Der verwendete Microcontroler

Zur Verarbeitung der Daten wird ein AT90S4433 (Atmel) [Atm] verwendet. Die Wahl fiel auf diesen Microcontroler, da er als schneller 8–Bit–RISC–Prozessor mit 32 Registern ausgeführt ist und in einem PLCC 32–Gehäuse bereits den größten Teil der erforderlicher Prozessorperipherie enthält. Dazu gehören unter anderem:

- 4 kB Flash–ROM, der das Programm trägt und der über eine ISP–Schnittstelle in der Schaltung programmierbar ist,
- 128 Byte SRAM,
- ein 8– und ein 16–Bit–Timer,
- ein 10–Bit–AD–Wandler, der über einen Multiplexer auf 6 Eingänge geschaltet werden kann,

- eine serielle Schnittstelle mit variable Baudrate,
- 20 programmierbare IO–Leitungen.

Die einzigen zusätzlichen Komponenten, die für den Betrieb dieses Microcontrolers erforderlich sind, sind ein Quarz mit zwei Kondensatoren.

2.3 Schaltungsbeschreibung

2.3.1 Stromversorgung

Die Betriebsspannung von 11 bis 16 Volt Gleichspannung liegt an den Pins 1 und 2 an. Diode D1 schützt die Schaltung vor unbeabsichtigter Verpolung, sichert die Ladung auf C4 bei kurzzeitigem Einbruch der Versorgungsspannung und dient bei Kurzschlüssen im weiteren Teil der Schaltung als »Sicherung«. IC 6 stabilisiert die Eingangsspannung auf 8 Volt, die für die Flip–Spule benötigt wird. Da beim Ummagnetisieren des Sensors Stromspitzen bis über 2A auftreten, dient C5 zur Glättung.
Diese Spannung von 8 Volt wird durch IC 7 auf 5 Volt herabgesetzt, und dient dem Logikteil als Versorgung. Die 5 Volt werden außerdem nach der Glättung über ein RC–Glied, bestehend aus R7 und C7 dem Analogteil zur Verfügung gestellt.
Einer der Operationsverstärker in IC 8 dient in Verbindung mit R8, R9, R32 und C17 dazu, um die für die Differenzverstärker benötigte halbe Betriebsspannung zu erzeugen.

2.3.2 Erzeugung der Ummagnetisierungspulse

Das Umschalten der Magnetisierung wird durch den Microcontroler ausgelöst. Dazu werden über den Port B, Pins 0 & 1 zwei Inverter bestehend aus T2, T3, R5, R6, R3 und R4 gesteuert. Diese dienen als Pegelumsetzer von 5 auf 8 Volt. Jeder der beiden Inverter steuert drei CMOS–Schmitt–Trigger–Inverter an, die zur Erhöhung des Ausgangsstromes auch ausgangsseitig parallelgeschaltet sind. Die Gatter treiben eine aus MOS–Transistoren bestehende Endstufe, die dank ihres niedrigen Innenwiderstandes von max. 0.1 Ω auch bei 8 Volt in der Lage ist, den zum Ummagnetisieren erforderlichen Strom von ca. 2 A durch die Flip–Spule mit ihrem Innenwiderstand von bis zu 5Ω zu treiben.
Bei dieser Art der Beschaltung ist durch die Firmware sicherzustellen, daß nie beide Transistoren gleichzeitig leitend sind, da dies sonst zu einem Einbruch der Versorgungsspannung und somit zum Neustart des Microcontrolers führen würde.
Das Einschalten der Anordnung ist auch bei unprogrammiertem Microcontroler unbedenklich, da die Ports des Prozessors dann hochohmig sind, T2 und T3 somit sperren, alle Gatterausgänge auf LOW liegen und daher nur der P–Kanal–Fet leitend ist.
Die Spannungsänderung, die beim Umschalten von T1 auftritt, wird durch den Tantalkondensator C1 in einen kurzen Stromimpuls durch die Flip–Spulen von IC1 und IC2 umgesetzt.
Laut Spezifikation der Sensoren muß der Strom durch die Flipspule für mindestens eine Mikrosekunde mindestens 800mA betragen. Dabei darf ein Wert von 1500mA nicht überschritten werden, um Schäden am IC zu vermeiden. Zur Strombegrenzung dienen die Widerstände R13, R14 und R15, sowie R16, R17 und R18, von denen je nach Widerstand der Flipspule, der zwischen 1 und 5 Ohm liegen kann, jeweils zwei oder drei bestückt sind.
Außerdem ist die Verlustleistung der Flip–Spule zu berücksichtigen, die maximal 50mW betragen darf. Unter der Worst–Case–Annahme, daß die Spulenversorgung 9V, der Flip–Spulenwiderstand 5Ω und daher der Vorwiderstand 3,3Ω beträgt, ergibt sich die Energie, die aus einer Kondensatorumladung in der Flip–Spule verbleibt zu

$$E_{Spule} = \frac{\frac{1}{2} * C * U^2 * (R_s^2)}{(R_s^2 + R_v^2)} =$$

$$\frac{\frac{1}{2} * 10\mu F * (9V)^2 * ((5\Omega)^2)}{((3.3\Omega)^2 + (5\Omega)^2)} = 282\mu J \quad (2.4)$$

$$t_{mess} = \frac{E_{Spule}}{P_{max}} = \frac{282\mu J}{50mW} = 5{,}6\,ms \quad (2.5)$$

Die maximale Meßrate beträgt in diesem Fall 88.6 1/s, da zu einer Messung zwei Umladungen gehören. Dieser Wert wird im normalen Betrieb deutlich unterschritten.

2.3.3 Verstärkung des Sensorausgangssignals

Die Sensoren IC 1 und IC 2 liefern ein zum Magnetfeld proportionales Ausgangssignal von ca. 16 mV/V / kA/m. Das heißt, bei einer Betriebsspannung des Sensors von 5 Volt und einer Feldstärke des Erdmagnetfelds von 50A/m ist mit einem Ausgangsspannungshub von ±4mV bei unbelasteter Brücke zu rechnen. Diese Spannung soll nun auf einen Wert von ±1.25V verstärkt werden. Dazu ist ein Verstärkungsfaktor von 312 erforderlich.
IC 3 enthält zwei Präzisionsoperationsverstärker, die eine Offsetspannung von maximal 150µV, eine Verstärkung von mindestens $1{,}5 * 10^6$ und ein niedriges Rauschen aufweisen.
Die Verstärkung dieser Differenzverstärkerstufen beträgt, durch R1, R2, R21 und R25 bzw. R22, R23, R24 und R26 eingestellt 47. Diese niedrige Verstärkung der ersten Stufe stellt einen Kompromiß dar. Dieser ist bedingt durch den Eingangswiderstand, der 100kΩ nicht unterschreiten sollte, da sonst die Belastung der Brücke (ca. 2kΩ Innenwiderstand) nicht mehr vernachlässigbar ist und den Gegenkopplungswiderstand, der nicht größer als 4,7MΩ ausfallen sollte, da sonst der Eingangswiderstand des Operationsverstärkers (ca. 300 MΩ) ebenfalls nicht mehr vernachlässigbar ist.
Das nun auf ca. 180mV verstärkte Differenzsignal wird erneut je einem Verstärker zugeführt, der augrund seiner Beschaltung eine regelbare Verstärkung zwischen 8.5 und 14.7 aufweist. Dieser Verstärker vom Typ TLC274 (Texas Instruments) [Tex] besitzt zwar eine höhere Offsetspannung als IC3, diese spielt aber bei dem bereits verstärkten Signal kaum eine Rolle.
Der nun resultierende höhere Verstärkungsfaktor zwischen 400 und 690 hat sich in als sinnvoll erwiesen, da im normalen Anwendungsfall aufgrund der Inklination des Erdmagnetfelds die Magnetfeldlinien gegen die Hauptrichtung der Empfindlichkeit der Sensoren geneigt sind.
Die Kopplung zweier Verstärkerstufen wie im vorliegenden Fall besitzt einige Vorteile. Erstens reichen pro Zweig zwei Operationsverstärker aus, im Gegensatz zu einer Lösung mit einem Instrumentenverstärker, der drei OPs benötigt oder enthält [Elr95]. Zweitens muß hier nur der erste OP eine niedrige Offsetspannung aufweisen, was sich in den Schaltungskosten positiv auswirkt. Drittens ist eine Regelung der Verstärkung bei dieser Beschaltung einfacher, als dies bei einer einzelnen Differenzstufe der Fall wäre. Viertens kann der empfindliche Verstärker an die geglättete analoge Versorgungsspannung von 5 Volt angeschlossen werden und trotzdem erreicht der Ausgang des zweiten OPs einen Hub von mehr als 5 Volt, da er mit +8 Volt versorgt wird.

Die nun verstärkten Signale können über Pin 7 und Pin 8 zu Meßzwecken abgegriffen werden. Pin 9 dient dann als Masse.
Nach einer Glättung der Sensorsignale durch R19 und C15 bzw. R20 und C16, die einen Teil des

Rauschens, die Störungen, die durch den Ummagnetisierungspuls entstehen und externe höherfrequente Einstreuungen dämpfen, werden die Spannungen den Multiplexereingängen des AD−Wandlers zugeführt.

2.3.4 Prozessorperipherie

Die rot/grüne Duoleuchtdiode D2 wird als Statusanzeige verwendet. Q1 sorgt in Verbindung mit C2 und C3, sowie dem prozessorinternen Oszillator für die Taktrate von 3.6864 Mhz, die auch zur Erzeugung des Datentakts für die RS232−Schnittstelle dient.
Letztere wird realisiert durch IC10. Der IC vom Typ ICL 232 setzt die vom Prozessor erhaltenen 5 Volt Logikpegel in Spannungen von ±10V um, die der IC mit Hilfe der Kondensatoren C10 bis C13 aus der Betriebsspannung von +5V selbst erzeugt. Außerdem werden ankommende RS232−Pegel in 5V Logikpegel umgesetzt. Der IC besitzt in jeder Richtung zwei Umsetzer, so daß die Signale RX, TX, DTR und DSR realisiert werden konnten.

2.3.5 Kompensationsmaßnahmen

Damit Schwankungen in der Versorgungsspannung nicht zu Verfälschungen des Meßergebnisses führen, ist der Referenzeingang des AD−Wandlers auch an die Spannungsversorgung des Analogteils gekoppelt. Eine niedrigere Versorgungsspannung führt zu einer Senkung der Sensor−Brückenspannung. Diese wird linear verstärkt und dem Wandler zugeführt. Da aber die Referenzspannung des Wandlers in dem gleichen Maße sinkt, wie die Brückenspannung, wird dieser Fehler eliminiert.
Der Temperaturdrift des Sensors von −0.4%/K spielt ebenfalls keine Rolle, da für die Bestimmung der Richtung nur das Verhältnis der Meßgrößen relevant ist. Dieses bleibt immer konstant, weil sich aufgrund der geringen Entfernung der Sensoren beide gleichmäßig erwärmen. Lediglich die aus den Meßwerten berechnete Qualitätsangabe wird bei Änderung der Temperatur sinken.

3 Beschreibung der Firmware

Der Meßablauf und die Datenübetragung zum Rechner werden von einem Microcontroller durchgeführt. Das in AVR–RISC–Assembler geschriebene Programm ist in Anhang G zu finden.
Nach der Initialisierung des Prozessors bleibt das Programm in einer Endlosschleife. Alle weiteren Operationen werden durch Interrupts ausgelöst. Dazu gehören die Steuerung des Meßablaufs, die Erzeugung einer internen Prozessorzeit, das Senden der Meßdaten zusammen mit einem Zeitstempel, sowie das Empfangen von Befehlen und deren Auswertung.

3.1 Port– und Variablendefinitionen

Am Anfang des Quellcodes wird eine Include–Datei eingebunden, die einige Portdefinitionen enthält. Diese Datei gehört zum Lieferumfang des Assemblers. Da die Include–Datei für den AT90S4433 zum Zeitpunkt der Entwicklung des Kompasses nicht verfügbar war, wird hier die für den AT90S2313 verwendet und um die nicht enthaltenen Definitionen ergänzt (»definition of some ports«).
Der Prozessor stellt 32 8–Bit Register zur Verfügung, daher sind bis auf den 10 Byte großen Sendepuffer alle Variablen als Registervariablen angelegt. Die Zuweisung von Variablennamen zu Registern findet in der Sektion »register definitions« statt. TXTEMP ist eine im Sendeinterrupt benötigte temporäre Variable. MEASNUMBER enthält die Anzahl der Messungen, die nach jedem Ummagnetisierungspuls durchgeführt werden sollen.
CLOCK1 bis CLOCK5 sind Zähler, welche die Prozessorzeit enthalten. CLOCKDIV, CLOCKONE und CLOCKZERO werden als Konstanten verwendet, die 18, 1 bzw 0 enthalten. Das ist notwendig, da die Register mit Nummern kleiner als 16 nicht mit einer Konstanten verglichen werden können und der Befehlssatz keinen Assembler–Befehl für die Addition eines Registers mit einer Konstanten enthält.
TEMP stellt eine allgemeine temporäre Variable dar.
Das letzte Bit von POSNEG enthält die Information, ob der nächste Ummagnetisierungspuls positiv (0) oder negativ (1) sein soll.
Die Anzahl der durchgeführten Messungen steht in MEASCOUNTER. SENDCOUNTER enthält die aktuelle Anzahl der über die serielle Schnittstelle übertragenen Bytes.
RXCOMMAND trägt das letzte empfangene Komando, in RXVALUE wird der zugehörige Parameter gespeichert. RXTEMP wird als temporäre Variable von der Empfangsroutine benötigt.
Das vorletzte Bit im STATUS–Register gibt an, ob das nächste Byte, das empfangen wird ein Komanndo oder ein Parameter ist. Das letzte Bit gibt an, ob eine kontinuierliche Messung oder Messung auf Anfrage betrieben wird.
TEMPHI und TEMPLO sind zwei Hilfsvariablen, die für die Summen– bzw. Differenzbildung der Meßwerte benötigt werden. MEASNSHI, MEASNSLO, MEASEWHI, MEASEWLO enthalten schließlich die eigentlichen Meßwerte.

3.2 Initialisierung des Microcontrollers

Wird bei dem AT90S4433 ein Interrupt ausgelöst, so führt der Prozessor die Speicherzelle aus, deren Adresse (im Wort–Format) der Interruptnummer entspricht. Ein Interrupt mit niedriger Nummer besitzt eine höhere Priorität. Der Interrupt mit Nummer 0, also mit der höchsten Priorität, ist der Reset–Interrupt. Dieser wird ausgeführt, wenn der Prozessor gestartet wird.
In der Sektion »definition of interrupts« werden die Interruptroutinen den zugehörigen Interrupts zugewiesen.
IRQ0 wird nach dem Einschalten ausgeführt und springt ins Hauptprogramm.
Sobald der 16–Bit–Zähler einen einstellbaren Vergleichswert erreicht, wird IRQ4 ausgelöst. Dieser

wird dazu verwendet, um die Messung zu steuern.
IRQ6 wird aktiv, wenn im 8–Bit–Zähler ein Überlauf auftritt, damit wird die Prozessorzeit generiert.
Wird ein Byte von der seriellen Schnittstelle empfangen, wird IRQ8 ausgelöst. IRQ9 zeigt an, daß das Senderegister leer ist und ein weiteres Byte geschrieben werden kann.

Im MAIN–Programm wird der Microcontroller und seine Peripherie konfiguriert. Zunächst wird der Stack–Pointer gesetzt, damit eine spätere Interruptverarbeitung einwandfrei ablaufen kann.
Als nächstes werden die IO–Ports entsprechend ihrer Beschaltung als Ein– bzw. Ausgänge festgelegt, die LED auf gelb geschaltet und DSR auf low gelegt.
Es folgt die Konfiguration des prozessorinternen UARTs. Empfangs– und Sendeeinheit werden eingeschaltet, der Empfangsinterrupt wird aktiviert und die Baudrate auf 19200 festgelegt. Weitere Teilerfaktoren für andere Baudraten finden sich in in der Dokumentation des AT90S4433 [Atm].
Anschließend werden die Timer konfiguriert. Der 8–Bit–Timer erhält den durch 8 geteilten CPU–Takt. Außerdem wird der Timer–Overflow–Interrupt aktiviert, der bei jedem Zählerüberlauf einen Interrupt auslöst, so daß die CLOCKTICK–Routine 1800 mal pro Sekunde aufgerufen wird.
Des weiteren werden die Zählvariablen auf null gesetzt und die Konstanten zugewiesen.
Der 16–Bit–Timer wird so konfiguriert, daß er mit 1/8 des CPU–Taktes angesteuert wird, bei Erreichen eines Vergleichswertes einen Interrupt auslöst und gleichzeitig den Zähler wieder auf null setzt. Der voreingestelle Wert für den Vergleichswert beträgt 0x4000, das entspricht einer Meßrate von ca. 0.9 1/s bei voller Auflösung.
Gegen Ende des Bootvorgangs werden noch die Meßvariablen und der Messungszähler auf null gesetzt, sowie dem Status, dem POSNEG–Flag und MEASNUMBER Standartwerte zugewiesen.
Nun wird dem Computer durch setzen der DSR–Leitung mitgeteilt, daß der Kompaß bereit ist. Der Microcontroller wartet seinerseits nun auf das DTR–Signal vom Computer. Liegt dieses an, so werden die Interrupts freigegeben, die LED ausgeschaltet und das Programm verweilt in einer Endlosschleife. Diese enthält ein SLEEP, um den Prozessor zwischen den Interrupts anzuhalten. Sollte DTR während des Betriebs auf low gehen, wird der Programmanfang angesprungen und so ein Reset ausgeführt.

3.3 Meßroutine

Die Meßroutine wird ausgeführt, sobald der 16–Bit–Timer den eingestellten Vergleichswert erreicht hat. Dort wird zunächst überprüft, ob es sich um eine positive oder eine negative Messung handelt. Im Falle einer positiven Messung wird zunächst die LED grün geschaltet. Ist MEASCOUNTER noch null, so wird er auf eins gesetzt und ein positiver Ummagnetisierungspuls wird erzeugt. Eine kurze Verzögerung verhindet das gleichzeitige Leiten der Endstufentransistoren. Danach wird die Interruptroutine beendet.
Ist MEASCOUNT aber größer als null, so findet eine Messung statt. Dazu wird der Multiplexer des AD–Wandlers zuerst auf den Nord–Süd–Sensor geschaltet, die Wandlung gestartet und deren Ende abgewartet. Das Meßergebnis wird zu dem bisherigen Nord–Süd–Meßwert addiert.
Anschließend findet der gleiche Vorgang mit dem Ost–West–Sensor statt. Danach wird überprüft, ob die gewünschte Anzahl Messungen bereits erfolgt ist. Wenn nicht, wird der Messungszähler um eins erhöht und die Interruptroutine beendet.
Falls ja wird der Meßzähler wieder auf null und das Polaritätsflag POSNEG gesetzt, um anzuzeigen, daß die nächste Messung negativ ist und wieder mit einem Ummagnetisierungspuls beginnt.
Bei der negativen Messung wird die Leuchtdiode wieder ausgeschaltet, dadurch ergibt sich ein Blinken der Leuchtdiode im Takt der Messungen.
Danach wird wieder der Sensor ummagnetisiert und anschließend der Nord–Süd– und der Ost–West–Wert ermittelt, aber diesmal wird der Wert vom aktuellen Meßwert abgezogen. Durch diese Differenzbildung wird der Offsetfehler der Operationsverstärker eliminiert.

Wurde die entsprechende Anzahl Messungen durchgeführt, wird der Meßzähler wieder auf null gesetzt und das POSNEG–Flag gelöscht. Dieses Aufsummieren über mehrere Messungen erhöht die Genauigkeit der Messung, da zum einen Rauschen ausgemittelt und zum anderen die Auflösung der AD–Wandler dadurch gesteigert wird.
Danach wird ein Präfix (0x41) zusammen mit den Meßdaten und dem aktuellen Zeitstempel in den Puffer für die zu sendenden Daten geschrieben und die Senderoutine ausgeführt. Die Zeitangabe entspricht also dem Zeitpunkt des Meßendes, nicht dem des Meßanfangs. Sobald alle Daten an den Computer übertragen sind, werden die Variablen für den Meßwert wieder auf null gesetzt und, falls nur eine Messung auf Anfrage durchgeführt wurde, der Timer–Compare–Interrupt abgeschaltet.

3.4 Senderoutine

In der Senderoutine wird das erste Byte aus dem Sendepuffer ausgelesen und in das UART–Data–Register geschrieben. Danach wird der UART–Data–Register–Empty–Interrupt aktiviert und die Interruptroutine wieder verlassen.
Sobald das erste Byte gesendet wird, wird vom Prozessor der Data–Register–Empty–Interrupt ausgelöst, der, wenn noch nicht alle Bytes geschickt wurden, das nächste Byte aus dem Sendepuffer liest und dem UART–Data–Register zuführt.

3.5 Empfangsroutine

Wird ein Byte von der seriellen Schnittstelle empfangen, so wird der RX–Complete–Interrupt ausgeführt. Da diese Interruptroutine aufgrund ihrer Priorität unterbrochen werden könnte, was nicht gewünscht ist, werden während ihrer Ausführung alle Interrupts gesperrt.
Ein Befehl vom Computer an den Microcontroller besteht immer aus zwei Bytes. Zuerst kommt ein Befehl, bestehend aus einem Buchstaben zwischen A und F und danach das Argument. Dazu wird anhand des letzten Bits im STATUS–Flag bestimmt, ob das empfangene Byte einen Befehl oder ein Argument darstellt.
Ist es ein Befehl, so wird dessen Gültigkeit sichergestellt, die LED auf rot geschaltet und Bit 0 im STATUS–Flag gesetzt um anzuzeigen, daß das nächste empfangene Byte ein Argument sein wird. Da der Befehl erst später benötigt wird, wird er in RXCOMMAND abgelegt.
Wird ein Argument empfangen, so wird es in RXVALUE gespeichert und Bit 0 von STATUS wieder gelöscht. Erst jetzt wird das Programm anhand des vorher empfangenen Befehls verzweigt:

- Kommando »A« startet die nächste Messung, falls Messung auf Anfrage eingestellt ist. RXVALUE ist hier belanglos.
- Kommando »B« schaltet zwischen kontinuierlicher Messung (RXVALUE=1) und Messung auf Anfrage (RXVALUE=0) um.
- Kommando »C« stellt die Zeit zwischen dem Ummagnetisieren und der ersten Messung bzw. zwischen den einzelnen Messungen ein. Dazu wird RXVALUE in das obere Byte des Vergleichswertes für den 16–Bit–Timer geschrieben. Die resultierende Intervallzeit beträgt 0,56ms*RXVALUE.
- Kommando »D« stellt die Anzahl der Messungen ein, die nach jedem Ummagnetisierungspuls ausgefürt werden sollen. Werte größer als 32 sind zwar möglich, aber nur in Spezialfällen sinnvoll, da es bei einer Vollaussteuerung des AD–Wandlers und mehr als 32 Messungen zu einem Zahlenüberlauf kommt und damit die Meßwerte unsinnige Größen annehmen können. Null ist als Wert nicht erlaubt.
- Kommando »E« schaltet die Status–LED ein (RXVALUE=1) oder aus (RXVALUE=0). Das kann benutzt werden, falls die Schaltung aus einer Batterie versorgt wird und Strom gespart werden soll, oder wenn eine flackernde Leuchtdiode andere Messungen stört.
- Kommando »F« setzt die Prozessoruhr zurück.

4 Software

Die Software ist in C (mit einigen C++–Erweiterungen) für ein Unix–System geschrieben und besteht aus drei Teilen: einer Bibliothek mit Headerfile, welche die zur Ansteuerung und Betrieb des Kompasses nötigen Funktionen enthält, einem Kalibrierungs– und einem Testprogramm. Der Quellcode für die angegebenen Programme ist in Anhang H zu finden.
Zum Compilieren der Programme reicht ein »make«.
Da die Software auf die serielle Schnittstelle, an welcher der Kompaß angeschlossen ist zugreifen muß, ist natürlich sicherzustellen, daß Lese– und Schreibzugriff auf die entsprechende Gerätedatei gewährt wird.

4.1 Kompassbibliothek (marcie.cc)

Die Bilbliothek marcie.cc enthält alle wichtigen Funktionen zur Benutzung des Kompasses. Im folgenden wird der Aufbau der Funktionen erklärt, eine Beschreibung der Programmier–schnittstellen findet sich im Headerfile marcie.h .

marcie_init() öffnet die Gerätedatei für die serielle Schnittstelle und speichert die alten Einstellungen. Danach werden alle noch im Puffer der Schnitstelle befindlichen Daten gelöscht und die neuen Einstellungen (19200 baud, 8 Datebits, 1 Stopbit) gesetzt.
Die Variable marciefdflag, die standartmäßig auf –1 gesetzt ist wird auf 0 gesetzt, um anzuzeigen, daß die Schnittstelle geöffnet ist.
Zuletzt wird noch die Funktion marcie_closeserial() als exit–Funktion eingetragen, d.h. Diese Funktion wird beim Beenden des Programms ausgeführt.

marcie_closeserial() schaltet beim Kompaß die kontinuierliche Messung aus. Das verhindert das Auftreten eines falschen Meßwertes nach folgendem Szenario:
Nehmen wir an, der Kompaß wäre noch aktiv und würde Daten liefern während die Schnittstelle gerade geöffnet wird. Alle noch im Schnittstellenpuffer befindlichen Daten werden dabei automatisch gelöscht. Danach wird der Kompaß auf Messung auf Anfrage geschaltet. Die laufende Messung wird aber noch durchgeführt, so daß nach deren Ende ein Meßwert geschickt wird, der natürlich nicht erwartet wird und beim weiteren Polling die Werte um einen Satz verschiebt.
Nach dem Abschalten des Kompasses setzt die Funktion die Schnittstelle auf die alten Einstellungen zurück und schließt die Gerätedatei.

marcie_sendcommand() schickt ein Kommando bestehend aus einem Befehl (A..F) und einem Argument (0..255) an den Kompaß. Diese Funktion wird von den folgenden Funktionen benutzt.

marcie_setmeascount() stellt die Anzahl der Messungen zwischen jeweils zwei Flip–Pulsen ein. Diese Anzahl wird auch in marciemeascount gespeichert, um bei der Berechnung der Gültigkeit eine Referenzamplitude zu haben, da die gemessene Amplitude natürlich mit zunehmender Anzahl sich addierender Messungen größer wird.
marcie_setmeasspeed() setzt die Meßgeschwindigkeit in 0.56ms–Schritten. Dabei ist die eingestellte Zeitspanne jeweils das Intervall zwischen dem Flip–Puls und der ersten Messung, zwischen den einzelnen Messungen, sowie zwischen der letzten Messung und dem nächsten Flip–Puls.
marcie_requestmeas() fordert eine Messung an. Bei eingeschalteter kontinuierlicher Messung hat diese Funktion keine Wirkung.
marcie_setcontmeas() schaltet zwischen kontinuierlicher Messung und Messung auf Anfrage um.
marcie_setclearclock() setzt die Kompaßuhr auf null.
marcie_ledonoff() schaltet die LED im Kompaß ein bzw. aus.

marcie_installhandler() richtet einen Signal–Handler ein, der jedes Mal, wenn ein Meßwert vom Kompaß vorliegt aufgerufen wird. Dazu wird zunächst überprüft, ob die Gerätedatei bereits geöffnet ist und noch kein Handler installiert ist. Das ist genau dann der Fall, wenn marciefdflag null ist.
Danach wird der Handler eingerichtet und der Dateizugriff auf asynchron geschaltet. marciefdflag wird dann mit FASYNC belegt um anzuzeigen, daß bereits ein Handler installiert ist und um bei späterem Umkonfigurieren der Schnitstelle den asychronen Modus beizubehalten.

marcie_getrawvalue() liefert die Meßwerte, bestehend aus Status, Nord–Süd–Wert, Ost–West–Wert und Uhr, wie sie vom Kompaß kommen. Wenn Blocking eingeschaltet ist, also die Funktion erst zurückkehren soll, wenn ein Meßwert empfangen wurde, dann wird der Dateizugriff entsprechend konfiguriert. Nun wird versucht, ein Zeichen zu lesen.
Liegt kein Zeichen an, und ist Nonblocking–Mode eingeschaltet, so kehrt die Funktion zurück, ansonsten wartet sie, bis ein Zeichen anliegt. Ist das gelesene Zeichen etwas anderes als der Header der Meßdaten vom Kompaß, so wird das Zeichen ignoriert und auf das nächste gewartet. Das stellt einen Schutz gegen falsche Synchronisation mit den Kompaßdaten dar, denn falls ein Byte verloren geht, würden die Meßdaten verschoben ankommen und daher falsch interpretiert. Wenn aber nun in jedem Fall auf den Datenheader gewartet wird, so kann ein auftretender Synchronisationsfehler wieder beseitigt werden, was anderweitig nicht möglich wäre. Natürlich können die Meßdaten zufälligerweise auch ein »A« enthalten, auf das nach einem Synchronisationsverlust dann falsch synchronisiert würde, aber durch die sich ändernden Daten ist es unwahrscheinlich, daß das über mehrere Meßwerte hin passiert.
Sobald also der korrekte Header empfangen wurde, wird der Dateizugriff in jedem Fall auf Blocking geschaltet, da die restlichen Bytes in jedem Fall gelesen werden sollen.
Diese bestehen aus einem Status–Byte, jeweils zwei Byte für NS– und OW–Wert, die danach in einen signed short int umgewandelt werden und vier Byte für die Kompaßuhr, die zu einem unsigned long int zusammengefügt werden.

marcie_loadcaltab() lädt die Kalibriertabelle in den Speicher. Für den Fall, daß kein Dateiname angegeben ist, wird er auf einen Standard–Wert gesetzt, anschließend wird die Datei geöffnet.
Läßt sich die Datei normal öffnen, so wird sie eingelesen. Dabei wird zunächst der Datei–Header übersprungen und dann der Nord–Süd– und der Ost–West–Gleichanteil gelesen. Anschließend folgen zeilenweise 361 Werte bestehend aus dem realen Winkel und einem Amplitudenwert für diesen Winkel. Zuletzt wird die Datei wieder geschlossen.

marcie_getangle() liefert den unkalibrierten Winkel, sowie Status, Gültigkeit und Uhr. Diese Funktion benutzt marcie_getrawvalue() um die Meßwerte zu erhalten und rechnet die Nord–Süd– und die Ost–West–Werte in Winkel und Gültigkeit um. Die Gültigkeit ist 1 für den Fall, daß die Magnetfeldamplitude genau einem Wert von 500 mal der Anzahl der durchgeführten Messungen pro Flip–Puls entspricht und wird kleiner, je weiter die Amplitude von diesem Wert entfernt ist.

marcie_getcalangle() gibt den kalibrierten Winkel, zusammen mit dem Status, der Gültigkeit und der Uhr zurück. Dazu wird auch hier mit marcie_getrawvalue() der Meßwert vom Kompaß gelesen und daraus ein Winkel und eine Amplitude berechnet. Dieser Winkel wird nun mit Hilfe der Kalibriertabelle in den realen Winkel umgerechnet. Liegt der gemessene Winkel zwischen zwei Tabelleneinträgen, so wird zwischen diesen linear interpoliert. Die Gültigkeit ist 1, falls der gemessene Wert der Amplitude aus der Kalibiertabelle bei diesem Winkel entspricht und wird kleiner, je weiter die beiden Werte auseinanderliegen.

4.2 Programm zur Kalibrierung (marciecalib.cc)

Um den Kompaß auch an Objekten montieren zu können, die einige Eisen– oder Stahlteile enthalten, ist eine Kalibrierung erforderlich, welche die Verzerrungen im Magnetfeld korrigiert.

Zunächst belegt das Programm genügend Speicher, um bei der folgenden Messung alle Meßwerte aufnehmen zu können. Als Dateiname für die Kalibriertabelle wird der erste Kommandozeilenparameter verwendet, oder falls keiner angegeben wird, ein Standartwert (marcie.cal). Die Datei wird dann geöffnet. Außerdem wird das Terminal, unter dem das Programm läuft so umgestellt, daß das Programm einzelne Tastendrücke lesen kann.
Nun wird der Kompaß initialisiert. Dabei wird eine hohe Meßgeschwindigkeit und volle Genauigkeit eingestellt. Der Benutzer versetzt nun den Roboter in Rotation und startet durch einen Tastendruck die Messung.
Während der folgenden etwa 2 ½ Umdrehungen des Roboters werden in konstanten Zeitintervallen die Meßwerte aufgezeichnet. Ist die Rotation beendet, so wird überprüft, ob eine gewisse Mindestanzahl an Meßwerten erhalten wurde. Wenn dies sichergestellt ist, so wird eine Autokorrelation der NS– und der EW–Werte durchgeführt. Das Mimimum des Autokorrelationswertes wird bestimmt, um die Periode des Signals zu ermitteln. Durch einen Offset bei der Autokorrelation wird garantiert, daß nicht das erste, sondern das zweite Minimum gefunden wird.
Nun wird auch klar, warum mindestens zwei, jedoch weniger als drei Umdrehungen lang gemessen werden muß. Bei weniger als zwei Umdrehungen kann die Periode nicht bestimmt werden und bei mehr als drei Umdrehungen kann es passieren, daß die erste mit der dritten Umdrehung besser korreliert, als mit der zweiten und daher das Programm die doppelte Periode annimmt.
Ist die Periode bestimmt, so wird innerhalb dieser Periode der Gleichanteil berechnet, und anschließend vom gespeicherten Signal abgezogen. Dabei wird auch der resultierende Winkel berechnet.
Das gespeicherte Signal wird nun nach dem Nulldurchgang des Ost–West–Wertes bei negativen Nord–Süd–Wert durchsucht. Dieser Punkt wird im folgenden als Norden angenommen.
Angefangen mit diesem Punkt wird nun eine Kalibriertabelle angelegt, die jedem gemessenen Winkel seinen realen Winkel zuweist. Da die Rotation des Roboters als konstant angenommen wird, ist die Position des Winkels in der Liste der aufgenommenen Meßwerte proportional zum Winkel. Nachdem auch die Anzahl der Meßwerte pro Umdrehung und der Nullpunkt, sprich der Norden–Eintrag bekannt ist, kann aus der Position des Meßwerts exakt der echte Winkel berechnet werden.
Aufgrund der Tatsache, daß die Kalibriertabelle natürlich nur endlich viele Einträge hat, müssen die gemessenen Winkel auf ganze Zahlen gerundet werden, um die Zeile für einen Eintrag zu liefern. Um dem so entstehenden Fehler Rechnung zu tragen, wird der an dieser Stelle eingetragene reale Winkel mit einem Gewichtsfaktor versehen, der von eins ab kleiner wird, je weiter der gemessene Winkel von dem gerundeten Wert entfernt ist. Nach der Gewichtung wird dieser reale Winkel in der Zeile der Tabelle, die sich aus dem gemessenen Winkel ergibt, ebenso wie sein Gewichtungsfaktor in je einer Spalte aufaddiert. Das Speichern des Gewichtungsfaktors ist nötig um den Winkel später wieder von seiner Gewichtung befreien zu können.
Außerdem ist auf die Unstetigkeitsstelle in der Tabelle zwischen 360 und 0 Grad zu achten. Es kann bei der Messung durch geringe Meßfehler durchaus passieren, daß ein gemessener Winkel von knapp über 0° einem realen von knapp unter 360° entspricht oder umgekehrt. Würden nun in einer Zeile der Kalibriertabelle einige Winkel von etwa 360° mit einigen von 0° verrechnet, so ergäben sich völlig unsinnige Einträge. Um dies zu verhindern, erfolgt eine Plausibilitätsprüfung der Werte, bevor sie eingetragen werden.
Die so gewonnene Kalibiertabelle wird anschließend nach fehlenden oder zu ungenauen (kleiner Gewichtsfaktor) Einträgen durchsucht. Wird ein solches Loch gefunden, so versucht das Programm, dieses durch lineare Interpolation der benachbarten Werte zu gewinnen, falls diese stark genug sind. Ist auch einer der beiden Nachbarwerte zu ungenau, so gibt das Programm eine

Meldung aus und die Messung muß mit geringerer Rotationsgeschwindigkeit wiederholt werden.
Zum Schluß werden die Einträge wieder von ihrer Gewichtung befreit und zusammen mit den Gleichanteilen für Nord−Süd und Ost−West in eine Datei geschrieben.

4.3 Testprogramm (marcietest.cc)

Das Programm marcietest dient dazu, den Kompaß auszuprobieren und den ersten Kalibrierschritt durchzuführen. Außerden kann es als Programmierbeispiel benutzt werden.
Das Programm enthält vier Funktionen, die vom Hauptprogramm aus aufgerufen werden können. Dazu müssen die Kommentarzeichen vor der entsprechenden Funktion entfernt, und das Programm neu compiliert werden. Eventuell muß auch der Name der Gerätedatei für die entsprechende serielle Schnitstelle angepaßt werden.
testraw() testet das direkte Lesen vom Kompaß. Diese Funktion wird auch für den Abgleich auf gleiche Sensoramplituden (siehe Anhang C.1) benötigt.
testnormal() liest die berechneten Winkel und Gültigkeiten aus und zeigt die an.
testcalib() stellt die realen Winkel dar und testsignal() ist ein Test für die asynchrone Bearbeitung der Daten.
Das Programm läuft, bis es durch einen Tastendruck beendet wird.

5 Messergebnisse

Um die Funktionsfähigkeit und die Genauigkeit des Kompasses zu prüfen wurden einige Messungen durchgeführt. Dazu wurde zunächst auf freiem Feld die Hardwarekalibrierung (siehe C.1) durchgeführt und der Kompaß anschließend an Marvin, dem mobilen Roboter des Lehrstuhls für Realzeit–Computersysteme montiert. Die Montage erfolgte so, das sich nicht in direkter Umgebung des Kompasses (<10 cm) Eisen– oder Stahlteile befanden, aber in nur etwa 20cm Abstand vom ebenfalls auf Marvin montierten Lautsprecher, so daß mit gewissen Störfeldern zu rechnen war.
Die Messung fand im Neubau im TU–Innenhof statt, ein Gebäude, das aufgrund seiner modernen Architektur sehr viele Stahl enthält und das mit einem Metalldach gedeckt ist.
Bei der ersten Messung rotierte Marvin um seine Achse mit konstanter Geschwindigkeit, dabei wurden in gleichbleibenden Zeitintervallen die Orientierungswerte aufgezeichnet. Abbildung 5.1 zeigt für eine Umdrehung die Abhängigkeit des gemessenen Winkels von realen Winkel (durchgezogene Linie). Der Wunschverlauf ist durch die Diagonale (gepunktete Linie) angegeben.

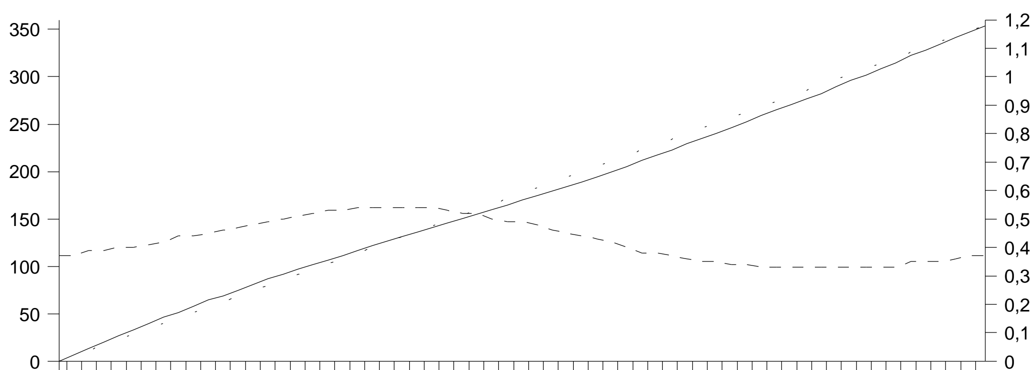


Abb. 5.1: gemessener Winkel zu realem Winkel und Gültigkeit ohne Kalibrierung

Bei dieser Messung ist noch eine deutliche Abweichung des gemessenen Winkels vom realen Winkel zu erkennen.
Der ebenfalls eingezeichnete Gültigkeitswert (gestrichelte Linie, rechte Y–Achse) deutet aufgrund seiner Amplitude von nur etwa 0,5 darauf hin, daß das Magnetfeld um den Roboter nicht dem im Freifeld entspricht.

Vor der zweiten Messung wurde eine Kalibrierung (siehe Anhang C.2) durchgeführt. Für die weitere Bestimmung der Orientierung wurde dann die erzeugte Kalibriertabelle verwendet.
Ebenso wie Abb. 5.1 zeigt Abb. 5.2 den gemessenen Winkel (linke x–Achse) in Abhängigkeit des realen Winkels. Allerdings sieht man hier deutlich, daß die Abweichungen des gemessenen Winkels vom realen Winkel weniger als zwei Grad betragen.
Außerdem ist zu erkennen, daß der Gültigkeitswert (gestrichelte Linie, rechte y–Achse) nun immer eins ist. Das resultiert daher, daß bei Verwendung der Kalibriertabelle auch der Gültigkeitswert korrigiert wird. Dieser bezieht sich nun nicht mehr auf die Feldstärke im Freifeld, sondern auf die Feldstärke, die bei der Kalibrierung für den jeweiligen Drehwinkel gemessen wurde (relative Gültigkeit).

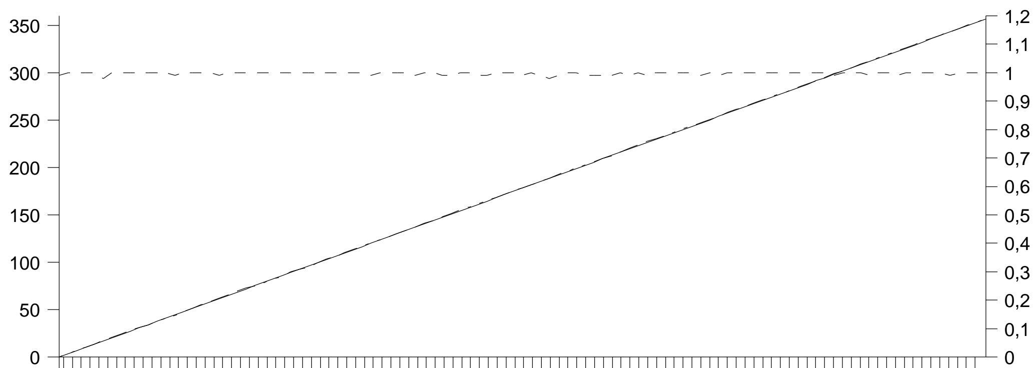


Abb. 5.2: gemessener Winkel zu realem Winkel und relkative Gültigkeit mit Kalibrierung

Auch bei der letzten Meßreihe wurde die Kalibriertabelle verwendet. Dabei fuhr Marvin etwa 12m geradeaus während die Meßwerte aufgenommen wurden. Abb. 5.3 zeigt den gemessenen Winkel über die abgefahrene Strecke (durchgezogene Linie, linke Y−Achse).

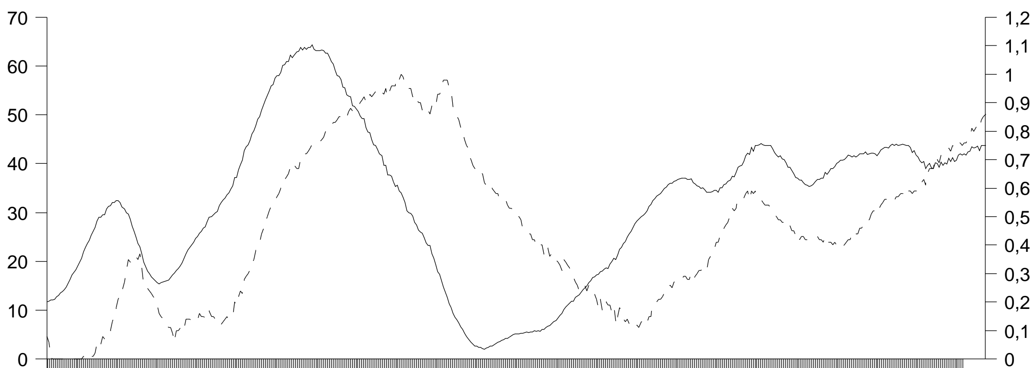


Abb. 5.1: gemessener Winkel und relative Gültigkeit bei gerader Fahrt in inhomogenem Feld

Die hier ebenfalls aufgetragene relative Gültigkeit (gestrichelte Linie, rechte y−Achse) deutet auf starke Schwankungen in der Feldamplitude hin.
Die große Streuung in den Werten ist auf eine sehr inhomogene Feldverteilung im Umfeld des Roboters zurückzuführen. Diese wird durch die eingangs erwähnte moderne Architektur mit vielen Stahlelementen verursacht. Auch die Verifikation der Meßergebnisse mit einem mechanischen Kompaß führte zu den gleichen Ergebnissen
Ein normaler Betrieb des Kompasses ist in dieser Umgebung praktisch nicht möglich (Alternativen: siehe Abschnitt 6).

6 Ausblicke

Wie die Messungen aus Abschnitt 5 zeigen, kann ein magnetischer Kompaß nur präzise Ergebnisse liefern, wenn das ihn umgebende Magnetfeld homogen ist.
Ein stark inhomogenes Magnetfeld muß die Verwendung eines magnetischen Kompasses jedoch nicht von vornherein ausschließen. Es ist nur nötig eine andere Meßmethode heranzuziehen. Dazu ist eine Karte der Magnetfeldlinien des Gebäudes anzulegen. Fährt der Roboter nun ein gewisses Areal (grob bestimmt durch seine Odometrie) ab, so kann er mit Hilfe des Kompasses den Verlauf der Magnetfeldlinien in diesem kleinen Bereich bestimmen. Das sich so ergebende Muster wird mit der gespeicherten Karte verglichen und kann nun, falls es eindeutig zuweisbar ist, sowohl zur Bestimmung der absoluten Orientierung, als auch zur Bestimmung der absoluten Position herangezogen werden.

Anhang A
Anleitung zum Aufbau der Schaltung

A.1 Zum Aufbau benötigte Dateien, Programme und Geräte

Zum Aufbau der Schaltung werden außer dem Schaltplan (Anhang D), dem Bestückungsplan (Anhang E) und der Stückliste (Anhang F) auch noch die Layoutdateien (kompasslayout_b.gif & kompasslayout_t.gif) oder die Board–Datei (kompass.brd) benötigt. Das Layout erhält Originalgröße, wenn die Layoutdateien mit 600 dpi ausgedruckt werden.
Zur Programmierung des Microcontrollers benötigt man einen AVR–ISP–Programer für AT90–Prozessoren nebst Programmiersoftware und AVR–Assembler (Anhang B), sowie einen PC mit paralleler Schnittstelle und Win95/98 oder NT.

A.2 Eigenbau einer doppelseitigen Leiterplatte

Zunächst wird mit Hilfe der Layoutfolien oder dem Eagle [Cads] (für den privaten Gebrauch reicht hier auch der Eagle 3.5 Light) oder einem anderen Layoutprogramm, welches das Eagle–Format lesen kann, eine doppelseitige Platine hergestellt. Ist die Erstellung einer doppelseitigen Platine nicht möglich, so reicht auch eine einseitige, mit Drahtbrücken anstatt der Leiterbahnen auf der Unterseite der Platine. (Achtung: die zwei Löcher in einer der Kupferflächen stellen eine Verbindung dar!)
Dank der geringen Anzahl von Durchkontaktierungen und dem Umstand, daß sich keine Durchkontaktierungen unter Bauteilen befinden, ist auch eine Herstellung mit Hobbyelektroniker–Mitteln möglich. Da die Schaltung nicht für Elektronik–Neulinge gedacht ist, erspare ich mir an dieser Stelle eine Anleitung zum Selberbau von Platinen. Wer allerdings selbst Platinen entwickelt, aber sich noch nicht an zweilagige herangewagt hat, für den an dieser Stelle ein paar Tips.
Zunächst besorgt man sich eine einfache Platine im Europaformat, sägt diese durch, so daß man zwei Platinenstücke mit 50x160 mm Größe erhält. Diese klebt man im rechten(!) Winkel Stoß an Stoß zu einem L zusammen. Dieses L dient im weiteren als Anschlag. Die ausgedruckten Layoutfolien werden nun großzügig (ca. 5mm um die Kreuze an den Ecken) ausgeschnitten und mit Klebeband am Platinen–L fixiert. Jeweils eine Folie auf der Vorder– und Rückseite des L´s. Dies hat so zu erfolgen, daß drei der Kreuze auf den Folien jeweils an den Kanten bzw. in der Ecke des L´s zu liegen kommen, die Schrift auf beiden Folien jeweils von oben lesbar ist und die Löcher sowie das verbleibende Kreuz bei der Durchsicht genau übereinanderliegen (siehe Abb. A.1).

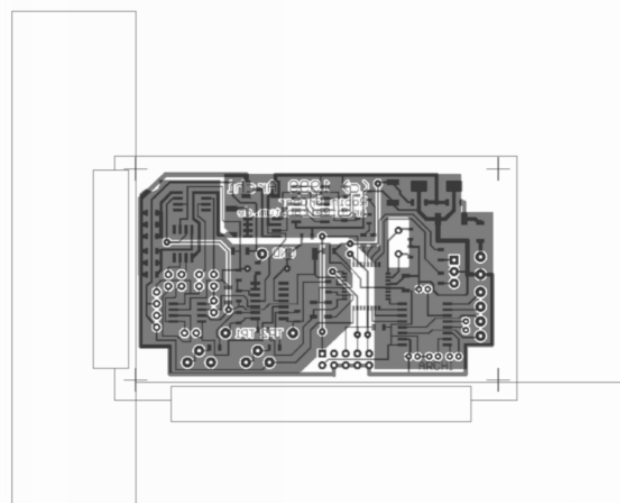


Abb. A.1

Als nächstes wird eine doppelseitige, fotobeschichtete Platine zurechtgesägt, entgratet, in das L eingelegt und auf jeder Seite belichtet. Danach Entwickeln, Ätzen, bohren, vom Fotolack befreien, auf Wunsch verzinnen und mit Lötlack einsprühen.
Zuletzt steckt man 0.6mm dicken Draht in die Durchkontaktierungslöcher und lötet diesen oben und unten fest.

A.3 Bohrdurchmesser

Alle Durchkontaktierungen (Vias) sind mit 0.6mm zu bohren, die Löt– und Anschlußpins mit 1mm, die Trimmer 1.2mm, alle anderen Löcher mit 0.8mm.

A.4 Bestückung

Zunächst werden die beiden Sensoren IC1 und IC2 bestückt. Dabei ist darauf zu achten, daß diese möglichst senkrecht zueinander und parallel bzw. senkrecht zur Platine aufgelötet werden.
Die weitere Bestückung erfolgt, mit Ausnahme der Bauteile C1 und R13 bis R18, laut Bestückungsplan (Anhang). Dabei kommen alle bedrahteten Bauteile auf die Bestückungsseite (die Seite mit den großen Masseflächen) und alle SMD–Bauteile auf die Lötseite. Der Quarz wird stehend eingebaut und kann mit dem Gehäuse an der Kupferfläche festgelötet werden. Soll die Schaltung später aus einer 9V–Quelle (Batterie oder Akku) gespeist werden, so muß IC 6 durch eine Drahtbrücke oder einen Null–Ohm–Widerstand ersetzt werden.
Anschließend mißt man den Widerstand zwischen dem Minus–Anschluß von C1 und den sensornahen Anschlüssen der Widerstände R13, R14 und R15. Dieser Wert muß zwischen 1 und 5 Ohm betragen. Ist er größer als 2,5 Ohm so müssen alle drei Widerstände bestückt werden, ansonsten nur zwei. Ebenso verfährt man mit R15, R16 und R17. Danach kann auch C1 bestückt werden.
Beim Anschluß der Schaltung an den Computer sind GND (Schaltung: Pin 2) mit GND (Stecker: Pin 5), TX (Schaltung: Pin 5) mit TX (Stecker: Pin 3), RX (Schaltung: Pin 4) mit RX (Stecker: Pin 2), DSR (Schaltung: Pin 3) mit DSR (Stecker: Pin 6) und DTR (Schaltung: Pin 6) mit DTR (Stecker: Pin 4) zu verbinden. Die Pinangaben beziehen sich auf eine 9 polige SUB–D–Buchse mit Blick auf die Lötseite (siehe Abb. A.2).

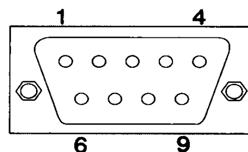


Abb. A.2

Der Pluspol der Stromversorgung ist mit Pin 1 der Schaltung, der Minuspol mit Pin 2 zu verbinden.

A.5 Inbetriebnahme

Eine Versorgungsspannung von 12V (9V wenn IC6 fehlt) ist anzulegen. Die Stromaufnahme muß unter 50mA liegen. Danach ist mit einem Oszilloskop mit 1:10–Tastkopf die Schwingung von Q1 zu überprüfen.
Hinweis: Solange der Microcontroller keine Ummangetisierungsimpulse liefert, sind keine sinnvollen Sensorwerte zu erwarten.
Nun wird der Microcontroller programmiert (Anhang B). Verläuft dies einwandfrei, so sollte die Schaltung bereits erste Werte liefern, sobald die Testsoftware (siehe 4.3) gestartet wird. Anschließend kann die Kalibrierung (Anhang C) vorgenommen werden.

Anhang B
Bedienung des Assemblers und des In–System–Programmers

Die folgenden Ausführungen stellen keine vollständige Bedienungsanleitung für den AVR–Assembler bzw. den ISP–Programmer dar, sondern enthalten nur die erforderlichen Schritte, um den Firmware–Sourcecode zu assemblieren und zu programmieren.
Die Programme kann man bei Atmel [Atm] erhalten. Ein Programmiergerät ist über Elektronikfachhandel zu beziehen.
Um den Mircrocontroller des Kompasses zu programmieren, muß man zunächst den AVR–Assembler starten und über das File–Menü die Datei marciefw.asm laden. Sollten Änderungen in der Firmware gewünscht werden, so sind diese auch hier durchzuführen.
Danach betätigt man den Assemble–Knopf. Nun sollte eine Meldung über den Erfolg der Assemblierung erscheinen. Der Assembler hat bei der Assemblierung unter anderem eine Datei marciefw.hex generiert. Diese wird im folgenden benötigt.
Spätestens jetzt sollte der Programmierer in den Druckerport gesteckt und das Programmierkabel mit dem Kompaß verbunden werden. Außerdem muß der Kompaß während der Programmierung mit Strom versorgt werden.
Als nächstes startet man den ISP–Programer. Falls noch nicht geschehen, konfguriert man den Druckerport, an dem der Programer steckt über das Menü Options ==> Change Printer–Port.
Bevor der Prozessor programmiert werden kann, muß ein neues Projekt erstellt werden. Dazu wird über das Menü Project==>New der Prozessor ausgewählt.
Achtung: Die von mir verwendete Version 2.6 hat einen Fehler, der dazu führt, daß der AT90S4433 nicht als solcher erkannt wird. Bei der Wahl des Prozessors muß daher AT90S/LS4434 eingestellt werden.
Danach erscheinen drei Fenster. Von diesen selektiert man das mit der Bezeichnung Program–Memory. Nun lädt man über das Menü File==>Load die Datei marciefw.hex.
Anschließend definiert man noch die Programmierungsoptionen über das Menü Program==>Autoprogram–Options. Dabei müssen aktiviert sein: Reload Files, Erase Device, Program Device und Verify Device. Die anderen Optionen werden hier nicht benötigt und verzögern die Programmierung nur unnötig.
Jetzt kann durch das Menü Program==>Autoprogram oder den Autoprogram–Knopf die Programmierung gestartet werden. Diese sollte ohne Fehlermeldungen ablaufen. Danach ist der Kompaß programmiert und bereit für die Kalibrierung (Anhang C).
Falls Zweifel bezüglich der Funktionsfähigkeit des Prozessors bestehen, so kann diese durch das Menü Program==>Health Check überprüft werden.

Anhang C
Kalibrierung

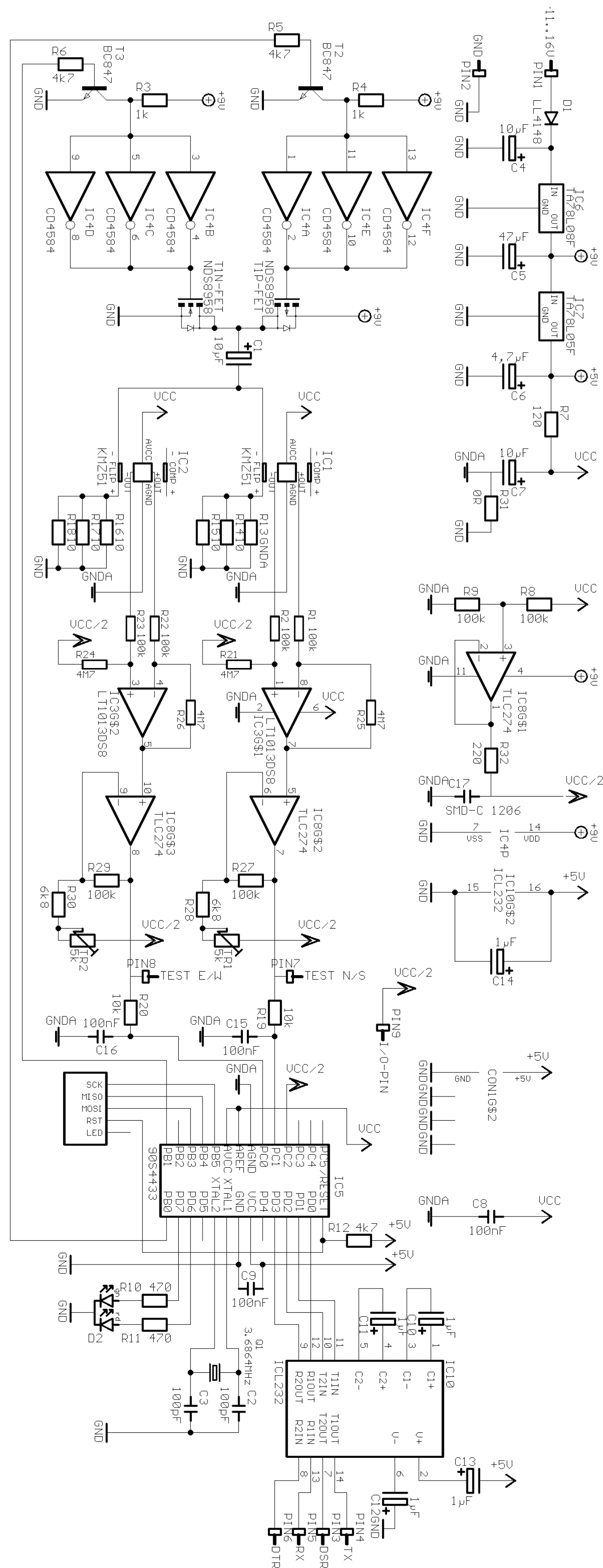
C.1 Abgleich auf gleiche Sensoramplitude

Der erste Abgleichschritt sollte in störungsfreier Umgebung erfolgen, das heißt, es dürfen sich keine magnetischen oder ferromagnetischen Gegenstände in der Umgebung befinden. Außerdem ist für eine ebene, waagrechte Unterlage zu sorgen. Am besten funktioniert der Abgleich mit einem Laptop und dem Sensor am langen (mind. 1m) Kabel, auf einem Steinquader inmitten einer großen Wiese.
Der Kompaß wird mit Computer und Stromversorgung verbunden, und das Programm marcietest (siehe 4.3) wird gestartet. Nun dreht man die Schaltung (ohne dabei mit den Fingern die Leiterbahnen zu berühren) auf ihrer waagrechten Unterlage, bis einer der Sensorwerte ein Maximum erreicht. Dieses Maximum wird nun durch Verstellen des zugehöriges Potis auf einen Wert von 16.000 gebracht. Nun führt man diese Prozedur auch für den anderen Wert durch. Ist es aufrund ungewöhnlicher magnetischer Verhältnisse (z.B. bei Betrieb in Pol– oder Äquatornähe) nicht möglich, einen Wert von 16.000 zu erreichen, so sollten beide Amplituden zumindest gleich groß eingestellt werden. In diesem Fall muß aber auch der Amplitudenwert in der Bibliotheksdatei marcie.cc angepaßt werden. Die Konstante MARCIE_SENSORAMP muß dabei auf den gemessenen Amplitudenwert geteilt durch 32 gesetzt werden.
Falls Amplitudenwerte von etwa 30.000 nicht zu unterschreiten sein sollten, so liegen erhebliche Störfelder vor, die einen Betrieb des Kompasses in dieser Umgebung verhindern.
Damit ist der erste Kalibrierschritt beendet.

C.2 Abgleich am Objekt

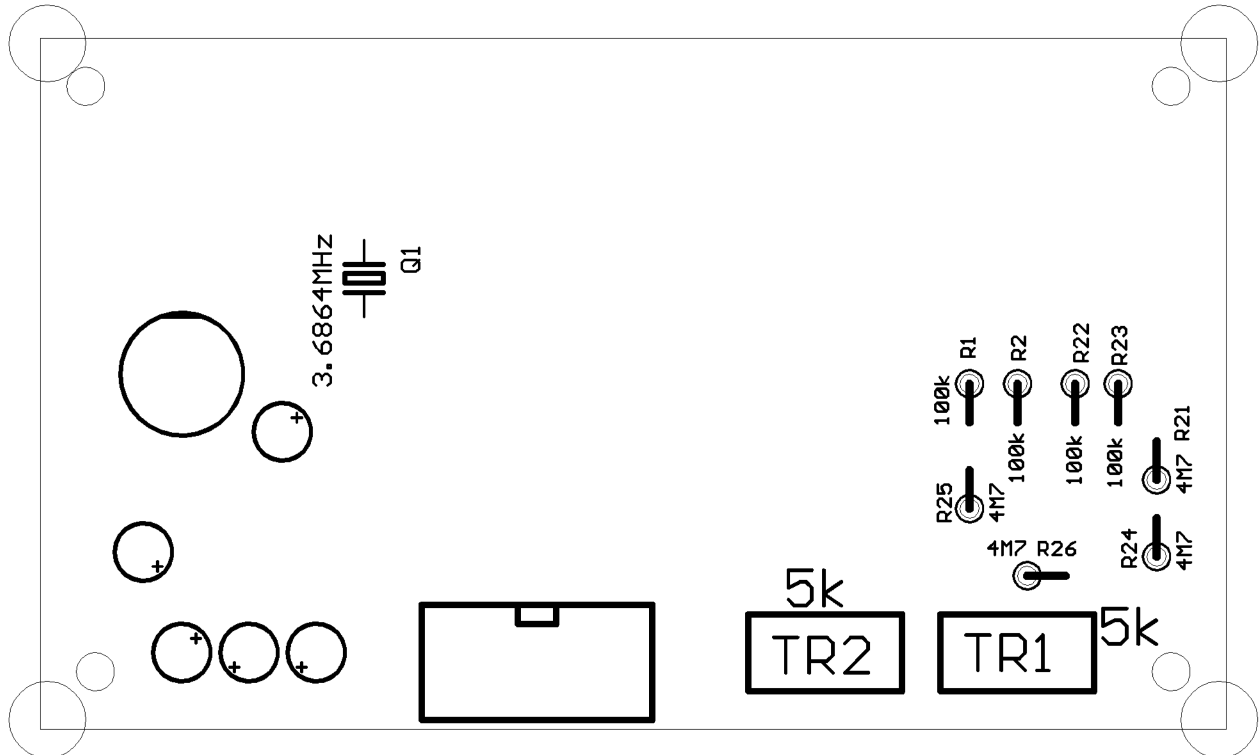
Nach dem Abgleich der Sensoramplituden erfolgt der Abgleich am Objekt. Dieser Schritt ist nur dann erforderlich, wenn der Kompaß an einem Objekt montiert wird, das in weniger als ca. 1m vom Kompaß Eisen– oder Stahlteile enthält, oder anderweitige magnetische Störfelder produziert. Die Montage eines Kompasses direkt auf ferromagnetischen Materialien ist generell nicht empfehlenswert.
Zur Durchführung des Abgleichs wird das Objekt mit montiertem Kompaß in einer möglichst störungsarmen Umgebung mit homogenen Magnetfeld positioniert und das Programm marciecalib (siehe 4.2) gestartet. Nun wird das Objekt in eine konstante Drehung im Uhrzeigersinn (von oben betrachtet) um die vertikale Achse versetzt und die Kalibrierung durch Tastendruck gestartet. Nach dem Tastendruck wartet man etwa 2 ½ Umdrehungen des Objekts ab und drückt erneut eine Taste. Die Rotation kann nun wieder angehalten werden.
Nun berechnet das Programm die Kalibriertabelle und speichert diese ab. Wurde die Rotation zu langsam durchgeführt oder war die Rotationsgeschwindigkeit nicht ausreichend hoch, so gibt das Kalibrierprogramm eine Fehlermeldung aus, und die Prozedur muß mit niedrigerer Rotationsgeschwindigkeit wiederholt werden. Ist die Kalibrierung abgeschlossen, so kann das Ergebnis mit dem Programm marcietest (siehe 4.3) begutachtet werden.
Immer wenn bauartliche Veränderungen an dem Objekt, das den Kompaß trägt vorgenommen werden, so ist eine erneute Kalibrierung durchzuführen.

Anhang D
Schaltplan

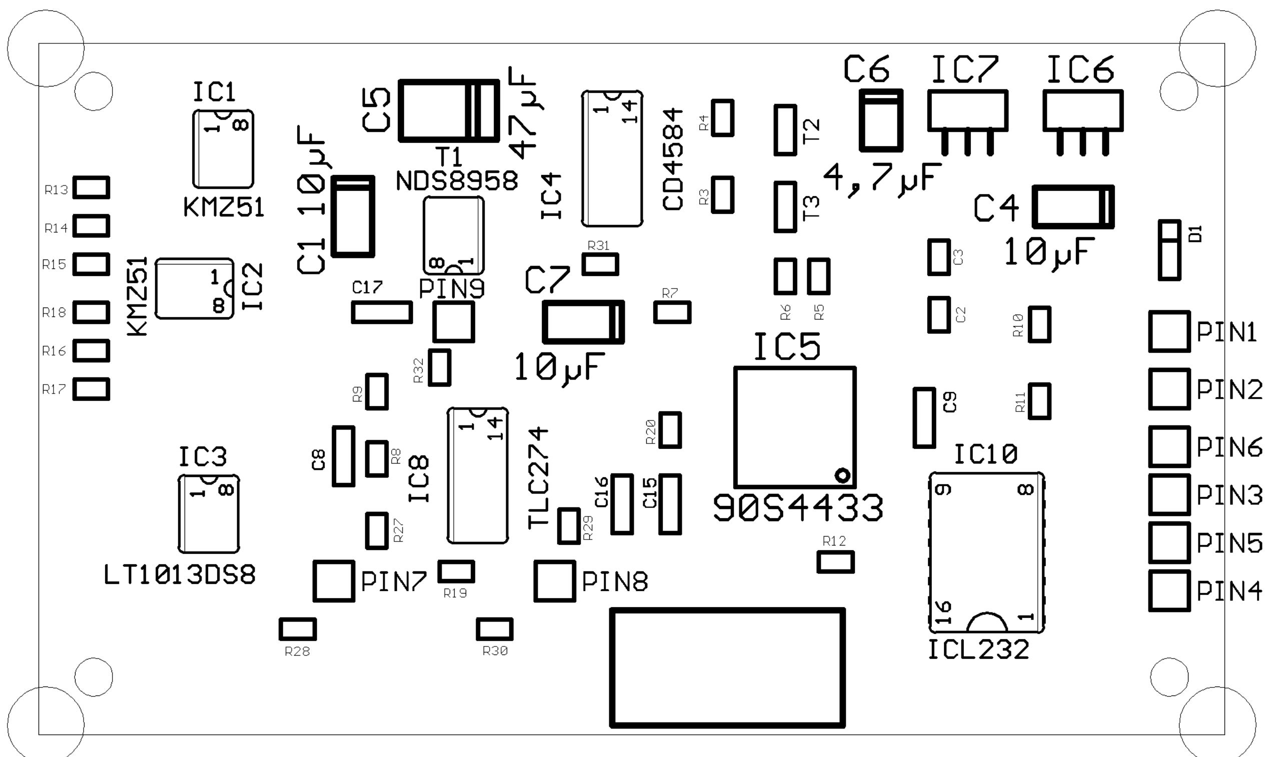


Anhang E
Bestückungsplan

Bestückungsseite:



Lötseite:



Anhang F
Stückliste

alle IC´s in SMD

Anz.	Bezeichnung	
2	KMZ 51 (Magnetfeldsensoren)	
1	AT90S4433 XXX	
1	ICL 232	
1	TA 78L08 F	
1	TA 78L08 F	
1	TLC 274	
1	LT 1013 DS8 (Achtung! Nur DS8, sonst andere Anschlußbelegung!)	
1	CD 4584	
1	NDS 8958	
2	BC 847	
1	Duo−LED rot / grün, 3mm	
1	LL4148	
1	Quarz 3,6864 Mhz	
1	47 µF 10V Tantal	
3	10 µF 16V Tantal	
1	4,7µF 16V Tantal	
5	1µF 16 V Elko stehend RM 2	
5	100nF keramisch BF: 1206	
2	100pF keramisch BF: 0805	
1	0Ω	0805
6	10Ω	0805
1	120Ω	0805
1	220Ω	0805
2	470Ω	0805
2	1k	0805
3	4,7k	0805
2	6,8k	0805
2	10k	0805
8	100k 1/4W	
4	4,7M 1/4W	
2	Trimmer 5k stehend RM 5	
1	10 pol. 2 reihig Steckerleiste RM 2,5	
3	Lötnägel 1mm	
1	Sub−D−Kupplung mit Gehäuse 9 oder 25 pol.	
	Kabel	

Anhang G
Quellcode der Firmware

```
;**************************************************************************
;*
;*
;* Number               :Archi 6
;* File Name              :"marciefw.asm"
;* Title                  :compass
;* Last change            :99-12-15
;* Author               :Robin Gruber gruber@rcs.ei.tum.de
;* Version                :0.8
;* Target MCU             :AT90S4433
;*
;* DESCRIPTION:
;*             Firmware for MARCIE, the electronic compass
;*             (c) 1999 RCS TU-Muncich
;*
;**************************************************************************


.include "2313def.inc"

; definition of some constants
.equ SENDBUFFER       =0x60          ; memory-location that keeps the bytes to send
.equ BYTESTOSEND=0x0A                ; number of bytes to send

.equ MEAS      =0x05                 ; default value for the number of measurements

.equ ADCONFIG  =0xC4                 ; value for configuring the ADC

; definition of some ports
.equ DDRC      =0x14                 ; data-direction-register
.equ PORTC     =0x15                 ; portc-output

.equ ADMUX     =0x07                 ; adress of ADC's multiplexer
.equ ADCSR     =0x06                 ; ADC-control-register
.equ ADCH      =0x05
.equ ADCL      =0x04                 ; ADC data-register

.equ NS        =0x00                 ; MUX selection-constants
.equ EW        =0x01

.equ LEDSON    =0xC6
.equ LEDSOFF   =0x06

.equ NFET      =1
.equ PFET      =0
.equ RED       =6
.equ GREEN     =7
.equ DTR       =3
.equ DSR       =2


; register definitions
.def TXTEMP    =r01                  ; temporary variable for the TX-interrupt
.def MEASNUMBER =r02                 ; number of measurements performed for each flip

.def CLOCK1    =r03                  ; prescaling-conter for clock
.def CLOCK2    =r04
.def CLOCK3    =r05
.def CLOCK4    =r06
.def CLOCK5    =r07                  ; 32-bit clock-variable
.def CLOCKDIV  =r08
.def CLOCKONE  =r09
.def CLOCKZERO =r10                  ; register-constants for clock

.def TEMP      =r16
.def POSNEG    =r17
.def MEASCOUNTER=r18
.def SENDCOUNTER=r19

.def RXCOMMAND  =r20
.def RXVALUE    =r21
.def RXTEMP    =r22

.def STATUS    =r23                  ; status-variable
; XXXXXXXX
; ||||||||
; |||||||+--> next byte received via UART is: 0: command, 1: value
; ||||||+---> 1: continous measurement, 0: measurement on request
; reserved

.def TEMPHI    =r24
.def TEMPLO    =r25
.def MEASNSHI  =r26
.def MEASNSLO  =r27
.def MEASEWHI  =r28
.def MEASEWLO  =r29
; registers 30 & 31 form the 16 bit Z-register


; definition of interrupts
```

```
.org $000
       rjmp MAIN                    ; bootup
.org $004
       rjmp TIMERCOMPARE            ; timerroutine
.org $006
       rjmp CLOCKTICK               ; clocktick for the clock
.org $008
       rjmp RXCOMPLETE              ; character received
.org $009
       rjmp TXREGISTEREMPTY         ; send-register ready to refill


.org $00e
MAIN:
       ;set up stack-pointer
       ldi TEMP,low(RAMEND)
       out SPL,TEMP                 ; set stack-pointer to end of RAM

       ; configure ports
       ; port D
       ldi TEMP,LEDSON
       out DDRD,TEMP                ; set port D pins 1, 2, 6 & 7 as outputs
       sbi PORTD,RED
       sbi PORTD,GREEN              ; switch LED to yellow, to indicate we're booting
       sbi PORTD,DSR                ; switch off DSR

       ; port B
       ldi TEMP,0x03
       out DDRB,TEMP                ; set port B pins 0 & 1 as outputs
       ldi TEMP,0x00
       out PORTB,TEMP               ; switch P-FET off and N-FET on

       ; port C (ADC)
       ldi TEMP,0x00
       out DDRC,TEMP                ; port C all pins are inputs
       out PORTC,TEMP               ; switch pullups off

       ; configure UART
       ldi TEMP,0x98
       out UCR,TEMP                 ; RX-interrupt on, RX on, TX on

       ldi TEMP,0x0B
       out UBRR,TEMP                ; set baudrate to 19200 with 3.6864MHz crystal

       ; set up 8 bit timer
       ldi TEMP,0x12
       mov CLOCKDIV,TEMP            ; the prescaler to get 1/100s clock is 18
       clr CLOCK1
       clr CLOCK2
       clr CLOCK3
       clr CLOCK4
       clr CLOCK5                   ; clear the counters
       clr CLOCKZERO                ; set up a register constant with 0
       ldi TEMP,0x01
       mov CLOCKONE,TEMP            ; set up a register constant with 1
       ldi TEMP,0x02
       out TCCR0,TEMP               ; Set 8 bit-timer-clock to CPU-clock / 8

       ; set up 16 bit timer
       ldi TEMP,0x02                ; disable timer-compare interrupt ...
       out TIMSK,TEMP               ; ... enable 8-bit-timer interrupt
       ldi TEMP,0x00
       out TCCR1A,TEMP              ; normal timer mode
       ldi TEMP,0x0A
       out TCCR1B,TEMP              ; timerclock=CPU-clock/8 / clear timer at compare

       ldi TEMP,0x40
       out OCR1AH,TEMP
       ldi TEMP,0x00
       out OCR1AL,TEMP              ; set the initial compare value to 0x4000

       ; reset some variables
       ldi MEASCOUNTER,0x00
       ldi POSNEG,0x00
       ldi STATUS,0x00

       ldi MEASNSHI,0x00
       ldi MEASNSLO,0x00
       ldi MEASEWHI,0x00
       ldi MEASEWLO,0x00            ; clear the measurement values

       ldi TEMP,MEAS                ; load the default-value for the number of...
       mov MEASNUMBER,TEMP          ; ...measurements performed

       cbi PORTD,DSR                ; now we're ready, switch on DSR...
BOOTWAIT:
       sbic PIND,DTR
       brne BOOTWAIT                ; and wait until terminal sets DTR

       cbi PORTD,GREEN
       cbi PORTD,RED                ; switch LED off

       sei                          ; start interrupts, here we go!

ENDLESS:
```

```
        sleep                          ; sleep until the next interrupt occurs
        sbis PIND,DTR                  ; if DTR drops during operation...
        rjmp ENDLESS
        rjmp MAIN                      ; ...perform a reset!


; timer-routine:
TIMERCOMPARE:
        sbrs POSNEG,0                  ; will this flip be positive? ...
        rjmp POSITIVEFLIP              ; ... perform a positive flip

NEGATIVEFLIP:
        cbi PORTD,GREEN                ; switch green LED off
        cpi MEASCOUNTER,0x00           ; if meascounter is zero, just flip...
        brne NEGATIVEMEAS              ; ... otherwise perform the measurement
        ; do the negative flip
        inc MEASCOUNTER                ; increase measurement-counter
        sbi PORTB,PFET                 ; perform the flip:...
        rcall UDELAY                   ; ... switch off P-FET , wait a little...
        sbi PORTB,NFET                 ; ... switch on N-FET
        reti

NEGATIVEMEAS:
        ; do the negative measurement
        ; get NS-value
        ldi TEMP,NS
        out ADMUX,TEMP                 ; switch ADC via MUX to NS-sense

        ldi TEMP,ADCONFIG              ; ADC on, start conversion, freerunmode off...
        out ADCSR,TEMP                 ; ...interrupt off, clock-prescaler is 32

        rcall WAITAD                   ; wait until conversion has finished

        in TEMPLO,ADCL
        in TEMPHI,ADCH                 ; read the values

        sub MEASNSLO,TEMPLO            ; since this is the negative measurement...
        sbc MEASNSHI,TEMPHI            ; ... substract the measurement value from the result

        ; get EW-value
        ldi TEMP,EW
        out ADMUX,TEMP                 ; switch ADC via MUX to NS-sense
        ldi TEMP,ADCONFIG              ; ADC on, start conversion, freerunmode off...
        out ADCSR,TEMP                 ; ...interrupt off, clock-prescaler is 32

        rcall WAITAD                   ; wait until conversion has finished

        in TEMPLO,ADCL
        in TEMPHI,ADCH                 ; read the values

        sub MEASEWLO,TEMPLO            ; since this is the negative measurement...
        sbc MEASEWHI,TEMPHI            ; ... substract the measurement value from the result

        cp MEASCOUNTER,MEASNUMBER      ; enough measurements performed?
        breq CHANGETOPOS               ; switch to the positive measurement
        inc MEASCOUNTER
        reti


CHANGETOPOS:
        ; finished the neagtive measurement
        ldi POSNEG,0x00                ; next measurement will be positive...
        ldi MEASCOUNTER,0x00           ; ... and will start with a flip

        ; now send the data we've got
        ldi SENDCOUNTER,0x00
        ldi ZH,high(SENDBUFFER)
        ldi ZL,low(SENDBUFFER)
        ldi TEMP,0x41
        st Z+,TEMP                     ; push "A" = header
        st Z+,STATUS                   ; push status
        st Z+,MEASNSHI
        st Z+,MEASNSLO
        st Z+,MEASEWHI
        st Z+,MEASEWLO                 ; push measurement data into send-buffer
        st Z+,CLOCK2
        st Z+,CLOCK3
        st Z+,CLOCK4
        st Z+,CLOCK5                   ; push actual clock into send-buffer
        rcall TRANSMIT                 ; transmit it
        ; clear data
        ldi MEASNSHI,0x00
        ldi MEASNSLO,0x00
        ldi MEASEWHI,0x00
        ldi MEASEWLO,0x00              ; clear measurement values for the next run

        ldi TEMP,0x02
        sbrs STATUS,1                  ; if we run measurement on request...
        out TIMSK,TEMP                 ; ... switch off timer-interrupt

        reti


POSITIVEFLIP:
        sbi PORTD,GREEN                        ; switch green LED on
```

```
        cpi MEASCOUNTER,0x00            ; if measurement-counter is zero, just flip ...
        brne POSITIVEMEAS               ; ... otherwise perform the measurement
        ; perform the flip
        inc MEASCOUNTER                 ; increase measurment-counter
        cbi PORTB,NFET                  ; switch off N-FET
        rcall UDELAY                    ; wait a little
        cbi PORTB,PFET                  ; switch on P-FET
        reti


POSITIVEMEAS:
        ; perform the positive measurement
        ; get NS-value
        ldi TEMP,NS
        out ADMUX,TEMP                  ; switch ADC via MUX to NS-sense
        ldi TEMP,ADCONFIG               ; ADC on, start conversion, freerunmode off...
        out ADCSR,TEMP                  ; ...interrupt off, clock-prescaler is 32

        rcall WAITAD                    ; wait until conversion has finished

        in TEMPLO,ADCL
        in TEMPHI,ADCH                  ; read the values

        add MEASNSLO,TEMPLO             ; since this is the positive measurement...
        adc MEASNSHI,TEMPHI             ; ... add the measurement value to the result

        ; get EW-value
        ldi TEMP,EW
        out ADMUX,TEMP                  ; switch ADC via MUX to NS-sense
        ldi TEMP,ADCONFIG               ; ADC on, start conversion, freerunmode off...
        out ADCSR,TEMP                  ; ...interrupt off, clock-prescaler is 32

        rcall WAITAD                    ; wait until conversion has finished

        in TEMPLO,ADCL
        in TEMPHI,ADCH                  ; read the values

        add MEASEWLO,TEMPLO             ; since this is the positive measurement...
        adc MEASEWHI,TEMPHI             ; ... add the measurement value to the result

        cp MEASCOUNTER,MEASNUMBER       ; enough measurements performed?
        breq CHANGETONEG
        inc MEASCOUNTER
        reti

CHANGETONEG:
        ldi POSNEG,0xFF
        ldi MEASCOUNTER,0x00
        reti



UDELAY:
        ; a small delay
        nop
        nop
        nop
        nop
        ret


WAITAD:
        sbic ADCSR,6
        rjmp WAITAD                     ; wait until ADC has finished
        ret


RXCOMPLETE:
        ; a character has been received via UART
        cli                             ; disable interrupts, we don't want to be disturbed!
        sbrc STATUS,0                   ; is this a command?
        rjmp ARGUMENTRECEIVED

COMMANDRECEIVED:
        in RXCOMMAND,UDR                ; read the received command from UART-data-register
        cpi RXCOMMAND,0x41
        brlo RXRETURN                   ; if the value is smaller than 0x41 or ...
        cpi RXCOMMAND,0x47
        brsh RXRETURN                   ; ...greater than 0x46 then it is no command, ignore it!
        sbi PORTD,RED                   ; switch red LED on
        sbr STATUS,0x01                         ; the next byte will be an argument
RXRETURN:
        sei
        reti

ARGUMENTRECEIVED:
        cbi PORTD,RED                   ; switch red LED off
        cbr STATUS,0x01                         ; the next byte will be a command
        in RXVALUE,UDR                  ; read the value
        cpi RXCOMMAND,0x41              ; determine which command is wanted...
        breq COMMAND_A
        cpi RXCOMMAND,0x42
        breq COMMAND_B
        cpi RXCOMMAND,0x43
        breq COMMAND_C
```

```
        cpi RXCOMMAND,0x44
        breq COMMAND_D
        cpi RXCOMMAND,0x45
        breq COMMAND_E
        cpi RXCOMMAND,0x46
        breq COMMAND_F                ; ... and execute it
        sei
        reti

COMMAND_A:
        ; request measurement
        ; syntax: Ax,
        ldi RXTEMP,0x00
        out TCNT1H,RXTEMP
        out TCNT1L,RXTEMP             ; clear timer
        ldi RXTEMP,0x42
        out TIMSK,RXTEMP              ; enable timer-interrupt
        sei
        reti

COMMAND_B:
        ; switch measurement to reqest/continous
        ; syntax: Bx; x=1: continous measurement; x=0: measurement on request
        cbr STATUS,0x02

        sbrc RXVALUE,0
        sbr STATUS,0x02

        ldi RXTEMP,0x42
        sbrc RXVALUE,0                ; if continous measurement is selected,
        out TIMSK,RXTEMP              ; switch on timer-interrupt
        sei
        reti

COMMAND_C:
        ; set time between measurement intervals
        ; syntax: Cx; where x*0.56ms is the time between each measurement and between...
        ;              ... the flipping and the first measurement
        ldi RXTEMP,0x00
        out OCR1AH,RXVALUE
        out OCR1AL,RXTEMP             ;... set the compare value to RXVALUE*256
        sei
        reti

COMMAND_D:
        ; set the number of measurements performed after each flipping pulse
        ; syntax: Dx; where x is the number of measurements [1..255]
        mov MEASNUMBER,RXVALUE
        sei
        reti

COMMAND_E:
        ; switch the status-LED off (in case it disturbs videometric applications
        ;                            or powersaving is important)
        ; syntax: Ex; where x is: 1: LED on; 0: LED off
        ldi RXTEMP,0xC6
        sbrs RXVALUE,0
        ldi RXTEMP,0x06
        out DDRD,RXTEMP               ; set port D pins 1, 2, 6 & 7 as outputs
        sei
        reti

COMMAND_F:
        ; clear the clock-counter
        ; syntax: Fx
        clr CLOCK1
        clr CLOCK2
        clr CLOCK3
        clr CLOCK4
        clr CLOCK5
        sei
        reti


TRANSMIT:
        ; trandmit the bytes in buffer
        ldi ZH,high(SENDBUFFER)
        ldi ZL,low(SENDBUFFER)        ; init Z-pointer with address of SENDBUFFER
        ldi SENDCOUNTER,0x01          ; set SENDCOUNT to one

        ld TEMP,Z+                    ; get data-byte from SRAM
        out UDR,TEMP                  ; write byte to UART TX-register

        sbi UCR,5                     ; enable UART-data-register-empty interrupt

        ret

TXREGISTEREMPTY:
        ld TXTEMP,Z+                  ; load next byte to send
        out UDR,TXTEMP                ; write it to UART-TX register

        inc SENDCOUNTER
        cpi SENDCOUNTER,BYTESTOSEND   ; all bytes already sent?
        brlo RETURN                   ; yes? stop sending data!
        cbi UCR,5                     ; disable UART-data-register-empty interrupt
```

```
RETURN:
        reti


CLOCKTICK:
        cli
        inc CLOCK1
        cp CLOCK1,CLOCKDIV
        breq INCCLOCK
        sei
        reti
INCCLOCK:
        clr CLOCK1
        add CLOCK2,CLOCKONE
        adc CLOCK3,CLOCKZERO
        adc CLOCK4,CLOCKZERO
        adc CLOCK5,CLOCKZERO
        sei
        reti
```

Anhang H
Quellcode der Software

H.1 Headerfile: marcie.h

```
/*
*******************************************
* routines to control MARCIE (headerfile) *
* (c) 1999 RCS TU-Munich                  *
*                                         *
* last change: 1999-12-15 Robin Gruber    *
* email: gruber@rcs.ei.tum.de             *
*******************************************
*/

#ifndef MARCIE_H
#define MARCIE_H
#include <stdio.h>
#include <sys/types.h>
#include <termios.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <math.h>
#include <signal.h>

#define MARCIE_ON 1
#define MARCIE_OFF 0
#define NORTHSHIFT 0.0

/* A short introduction to MARCIE´s (MAgneto Resisive Compass with serial IntErface) usage:
   Connect MARCIE to the computer and to the power-supply, allow
   access to the corresponding serial-device, test MARCIE (with marcietest)
   and calibrate her (with marciecalib).

   Before MARCIE can be used, the serial-device must be opened with marcie_init().
   Then load (if required) the calibration-table with marcie_loadcaltab().

   Set the marsurement-speed with marcie_setmeasspeed() and the number of
   measurements performed after one flipping-pulse with marcie_setmeasnumber()
   to the desired values.

   Optionally activate continous measurement with marcie_setcontmeas().

   If continous measurement is disabled, you have to request each value with
   marcie_requestmes().

   Get the results with marcie_getrawvalue(), marcie_getangle() or marcie_getcalangle().

   The serial device is automatically reset, if the program is terminated normally.
*/


int marcie_init(const char * serialdevice);
/* initializes the UART <serialdevice> is the device used e.g: "/dev/cua0"

   return-value: 0 on success
*/

int marcie_setmeascount(unsigned int count);
/* sets the number of measurements performed after each flipping pulse
   count = 1..32
   normally <count> should be 32 to get maximum resolution
   if a measurement with higher speed and lower resolution is needed, count
   can be less than 32

   return-value: 0 on success
*/

int marcie_setmeasspeed(unsigned int speed);
/* sets the intervall between the measurements and between the flipping-pulse and
   the first measurement in 0.55ms steps
   <speed> = 1..255

   return-value: 0 on success
*/

int marcie_requestmeas(void);
/* If measurement on request is selected, this triggers one measurement.

   return-value: 0 on success
*/

int marcie_setcontmeas(unsigned int cont);
/* sets measurement on request (MARCIE_OFF) or continous measurement (MARCIE_ON)

   return-value: 0 on success
*/

int marcie_clearclock(void);
```

```
/* resets MARCIE´S internal 1/100s clock

   return-value: 0 on success
*/

int marcie_ledonoff(unsigned int onoff);
/* switches marcie´s status-LED on (MARCIE_ON) or off (MARCIE_OFF)

   return-value: 0 on success
*/

int marcie_getrawvalue(signed short & ns, signed short & ew, unsigned long & clock,
              unsigned char & status,int block);
/* reads the present measurement-data

   <ns> and <ew> will be filled with the ns- and the ew-value,

   <clock> gets the timestamp from the 1/100s clock at the end of the measurement

   <status> holds MARCIE´s internal status

   If <blocking> is set (MARCY_ON) the fonction will not return, until measurement-data
   is available.

   return-value: 1 on success; 0 if in nonblocking-mode and no data is available
*/

int marcie_loadcaltab(char * filename);
/* reads the calibration-table (generated by marciecalib) from disk.

   <filename> is the filename of the calibration-file (e.g. "marcie.cal").

   return-value: 0 on success
*/

int marcie_getangle(double & angle, double & validity, unsigned long & clock,
                   unsigned char & status, int blocking);
/* Reads the preasent measurement with angle and validity instead of ns and ew.
   The calibration-table is not used.

   <angle> returns the angle in degree, where 0 is north, 90 is east, 180 south and 270 is west

   <validity> is a value between 0 and 1 that is an approximate indicator for the
   validity of the angle. It is calculated by the difference between the estimated signal-level
   and the real signal-level.

   <clock> gets the timestamp from the 1/100s clock at the end of the measurement

   <status> holds MARCIE´s internal status

   If <blocking> is set (MARCY_ON) the function will not return, until measurement-data
   is available.
*/


int marcie_getcalangle(double & realangle, double & validity, unsigned long & clock,
                   unsigned char & status, int blocking);
/* Reads the preasent measurement with angle and validity instead of ns and ew.
   The calibration-table is used and has to be loaded prior to the use of this function!

   <angle> returns the angle in degree, where 0 is north, 90 is east, 180 south and 270 is west

   <validity> is a value between 0 and 1 that is an approximate indicator for the
   validity of the angle. It is calculated by the difference between the signal-level during
   calibration and the real signal-level.

   <clock> gets the timestamp from the 1/100s clock at the end of the measurement
   status holds MARCIE´s internal status

   If <blocking> is set (MARCY_ON) the function will not return, until measurement-data
   is available.
*/

int marcie_installhandler(__sighandler_t handler);
/* Installs a signal-handler that is called every time, marcie sends a value.
   handler is a function of int returning void ( void handler(int) )
*/
#endif
```

H.2 Kompassbibliothek: marcie.cc

```
/*
****************************************
* routines to control MARCIE           *
* (c) 1999 RCS TU-Munich               *
*                                      *
* last change: 1999-12-15 Robin Gruber *
* email: gruber@rcs.ei.tum.de          *
****************************************
*/


#include "marcie.h"
```

```
#define MARCIE_BAUDRATE B19200
#define MARCIE_SENSORAMP 500

static struct termios oldtio,newtio;
static int marciedevice;

static float marciecalangle[361];
static int marciecalamplitude[361];
static int marciecalns, marciecalew;
static int marciemeascount;
static int marciefdflag=-1;


void marcie_closeserial()
{
    marcie_setcontmeas(MARCIE_OFF);
    tcsetattr(marciedevice,TCSANOW,&oldtio);
    close(marciedevice);
}


int marcie_init(const char * device)
{
    marciedevice = open(device, O_RDWR | O_NOCTTY );
    if (marciedevice <0) {return -1;}
    tcgetattr(marciedevice,&oldtio); /* save current port settings */
    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = MARCIE_BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;
    /* set input mode (non-canonical, no echo,...) */
    newtio.c_lflag = 0;
    newtio.c_cc[VTIME]    = 0;   // inter-character timer unused
    newtio.c_cc[VMIN]     = 10;   // blocking read until 10 chars received
    tcflush(marciedevice, TCIFLUSH);
    tcsetattr(marciedevice,TCSANOW,&newtio);
    marciefdflag=0;
    atexit(marcie_closeserial); // close serial device at program-exit
    return 0;
}


int marcie_sendcommand(unsigned int command, unsigned int argument)
{
    unsigned char w[2];
    if ((command>255)||(argument>255)) {return -1;}
    w[0]=(unsigned char)command;
    w[1]=(unsigned char)argument;
    if (write(marciedevice,w,2)!=2) {return -2;}
    return 0; // success!
}


int marcie_setmeascount(unsigned int count)
{
    if ((count>32)||(count==0)) {return -1;}
    marciemeascount=count;
    return marcie_sendcommand(0x44,count);
}


int marcie_setmeasspeed(unsigned int speed)
{
    if (speed==0) {return -1;}
    return marcie_sendcommand(0x43,speed);
}


int marcie_requestmeas(void)
{
    return marcie_sendcommand(0x41,0);
}


int marcie_setcontmeas(unsigned int cont)
{
    if ((cont!=0)&&(cont!=1)) {return -1;}
    return marcie_sendcommand(0x42,cont);
}


int marcie_clearclock(void)
{
    return marcie_sendcommand(0x46,0);
}


int marcie_ledonoff(unsigned int onoff)
{
    if ((onoff!=0)&&(onoff!=1)) {return -1;}
    return marcie_sendcommand(0x45,onoff);
}
```

```
int marcie_installhandler(__sighandler_t handler)
{
    struct sigaction saio;
    sigset_t sigset;
    memset((void*)(&sigset),0,sizeof(sigset_t));
    /* install the signal handler before making the device asynchronous */
    if (marciefdflag) {return -1;}
    saio.sa_handler = handler;
    saio.sa_mask = sigset;
    saio.sa_flags = 0;
    saio.sa_restorer = NULL;
    sigaction(SIGIO,&saio,NULL);
    /* allow the process to receive SIGIO */
    fcntl(marciedevice, F_SETOWN, getpid());
    /* Make the file descriptor asynchronous (the manual page says only
       O_APPEND and O_NONBLOCK, will work with F_SETFL...) */
    fcntl(marciedevice, F_SETFL, FASYNC);
    marciefdflag=FASYNC;
    return 0;
}


int marcie_getrawvalue(signed short & ns, signed short & ew, unsigned long & clock,
              unsigned char & status, int blocking)
{
    unsigned char c,d,e,f;
    int res;
    // blocking enabled/disabled?
    if (blocking==MARCIE_ON) {fcntl(marciedevice,F_SETFL,marciefdflag);}
    else {fcntl(marciedevice,F_SETFL,O_NONBLOCK|marciefdflag);}
    // wait until an A is read, since this is the header of MARCIE´s answer.
    do {
       res=read(marciedevice,&c,1);
       if (res==0) {return 0;}
       if (res<0) {return -1;}
    } while (c!='A');
    // Switch to blocking
    fcntl(marciedevice,F_SETFL,marciefdflag);
    // read the data
    if (read(marciedevice,&status,1)!=1) {return -2;}
    if (read(marciedevice,&c,1)!=1) {return -3;}
    if (read(marciedevice,&d,1)!=1) {return -4;}
    ns=(signed int)((c<<8)|d);
    if (read(marciedevice,&c,1)!=1) {return -5;}
    if (read(marciedevice,&d,1)!=1) {return -6;}
    ew=(signed short int)((c<<8)|d);
    if (read(marciedevice,&c,1)!=1) {return -7;}
    if (read(marciedevice,&d,1)!=1) {return -8;}
    if (read(marciedevice,&e,1)!=1) {return -9;}
    if (read(marciedevice,&f,1)!=1) {return -10;}
    clock=(unsigned long)((f<<24)|(e<<16)|(d<<8)|(c));
    return 1;
}


int marcie_loadcaltab(char * filename)
{
    int i;
    char dummy[256];
    FILE * calfile;
    if (filename==NULL) {filename="marcie.cal";}
    if ((calfile=fopen(filename,"r"))==NULL) {return -1;}
    if (fgets(dummy,256,calfile)==NULL) {return -1;}
    //printf("%s",dummy);
    if (fgets(dummy,256,calfile)==NULL) {return -1;}
    //printf("%s",dummy);
    fscanf(calfile,"NSDC: %d , EWDC: %d \n",&marciecalns,&marciecalew);
    //printf("NSDC: %d , EWDC: %d \n",marciecalns, marciecalew);
    for (i=0; i<=360; i++) {
       fscanf(calfile," %f , %d \n",&marciecalangle[i],&marciecalamplitude[i]);
       //printf("%5.1f,%d\n",marciecalangle[i],marciecalamplitude[i]);
    }
    fclose(calfile);
    return 0;
}


int marcie_getangle(double & angle, double & validity, unsigned long & clock,
                   unsigned char & status, int blocking)
{
    signed short ns,ew;
    int res;
    double amp;
    res=marcie_getrawvalue(ns,ew,clock,status,blocking);
    if (res<0) {return -1;}
    if (ew>=0) {
       angle=(atan(double(ns)/double(ew))+M_PI/2)/M_PI*180;
    } else {
       angle=(atan(double(ns)/double(ew))+3*M_PI/2)/M_PI*180;
    }
    amp=sqrt(double(ns)*double(ns)+double(ew)*double(ew))/MARCIE_SENSORAMP/marciemeascount;
    if (amp>2.0) {amp=2.0;}
    validity=1.0-fabs(1.0-amp);
    return res;
```

```
}


int marcie_getcalangle(double & realangle, double & validity, unsigned long & clock,
                   unsigned char & status, int blocking)
{
    signed short ns,ew;
    double amplitude,angle,calamplitude,amp;
    int res;
    res=marcie_getrawvalue(ns,ew,clock,status,blocking);
    if (res<=0) {return res;}

    if (ew>=0) {
       angle=(atan(double(ns)/double(ew))+M_PI/2)/M_PI*180;
    } else {
       angle=(atan(double(ns)/double(ew))+3*M_PI/2)/M_PI*180;
    }
    amplitude=sqrt(double(ns)*double(ns)+double(ew)*double(ew));

    realangle=(1.0-(angle-floor(angle)))*marciecalangle[(int)floor(angle)]+
       (1.0-(ceil(angle)-angle))*marciecalangle[(int)ceil(angle)];

    calamplitude=(1.0-(angle-floor(angle)))*marciecalamplitude[(int)floor(angle)]+
       (1.0-(ceil(angle)-angle))*marciecalamplitude[(int)ceil(angle)];
    amp=amplitude/marciemeascount*32.0/calamplitude;
    if (amp>2.0) {amp=2.0;}
    validity=1.0-fabs(1.0-amp);
    return res;
}
```

H.3 Testprogramm: marcietest.cc

```
/*
****************************************
* program to test MARCIE V.0.5         *
* (c) 1999 RCS TU-Munich               *
*                                      *
* last change: 2000-01-04 Robin Gruber *
* email: gruber@rcs.ei.tum.de          *
****************************************
*/

#include "marcie.h"

#define MARCIEDEVICE "/dev/ttyS1"


struct termios saved_attributes, tattr;


void restoreterminal(void)
{
    /* Restore the old terminal mode */
    tcsetattr (STDIN_FILENO, TCSANOW, &saved_attributes);
}


void modifyterminal(void)
{
    /* Make sure stdin is a terminal. */
    if (!isatty (STDIN_FILENO))
       {
           fprintf(stderr,"stdin is not a terminal!");
           exit (-1);
       }
    /* Save the terminal attributes so we can restore them later. */
    tcgetattr (STDIN_FILENO, &saved_attributes);
    /* Set the funny terminal modes. */
    tcgetattr (STDIN_FILENO, &tattr);
    tattr.c_lflag &= ~(ICANON|ECHO); /* Clear ICANON and ECHO. */
    tattr.c_cc[VMIN] = 0;
    tattr.c_cc[VTIME] = 0;
    tcsetattr (STDIN_FILENO, TCSAFLUSH, &tattr);
    atexit(restoreterminal);
}


int getkey(char & c)
{
    if (read(STDIN_FILENO, &c, 1)!=1) {return 0;}
    return 1;
}


void testraw(void)
{
    char c;
    unsigned char status;
    signed short ns,ew;
    double angle;
    long unsigned int clock;

    while (!getkey(c)){
```

```
        marcie_getrawvalue(ns, ew,clock,status,MARCIE_ON);
        if (ew>=0) {
            angle=((atan(double(ns)/double(ew))+M_PI/2)/M_PI*180);
        } else {
            angle=((atan(double(ns)/double(ew))+3*M_PI/2)/M_PI*180);
        }
        printf("Status: %3d, NS: %6d, EW: %6d Winkel: %6.1f clock:%9u \n",
               (unsigned int)status,(int)ns,(int)ew,angle,(int)clock);
    }
}


void testnonblock(void)
{
    char c;
    double angle,validity;
    long unsigned int clock;
    unsigned char status;
    while (!getkey(c)) {
        if (marcie_getangle(angle,validity,clock,status,MARCIE_OFF)<=0) {
            fprintf(stderr,"No data available!\n");}
        else {
        printf("angle: %5.1f, validity: %4.2f\n",angle,validity);
        }
        usleep(10000);
    }
}


void testcalib(char * calfile)
{
    double angle,validity;
    long unsigned int clock;
    char c;
    unsigned char status;
    if (marcie_loadcaltab(calfile)) {
        printf("could not open calfile!\n");
    }
    while (!getkey(c)) {
        if (marcie_getcalangle(angle,validity,clock,status,MARCIE_ON)<0) {
            fprintf(stderr,"Unnown error!\n");
            exit(0);}
        printf("angle: %5.1f, validity: %4.2f\n",angle,validity);
    }
}


void handler(int i)
{
    double angle,validity;
    long unsigned int clock;
    unsigned char status;
    if (marcie_getangle(angle,validity,clock,status,MARCIE_OFF)<0) {
        fprintf(stderr,"Unknown error!\n");
        exit(0);
        }
    printf("angle: %5.1f, validity: %4.2f\n",angle,validity);
}


void testsignal(void)
{
    char c;
    marcie_installhandler(handler);
    while (!getkey(c)) {
        sleep(1);
    }
}


int main(int argc, char * argv[])
{
    modifyterminal(); // switch the terminal to special mode
    if (marcie_init(MARCIEDEVICE)!=0) {
        printf("Error opening device %s!\n",MARCIEDEVICE);
        exit(0);
    }

    marcie_setcontmeas(MARCIE_ON);
    marcie_setmeasspeed(2);
    marcie_setmeascount(32);

    // testraw();
    //   testnormal();
    //    testcalib();
        testsignal();
    // testnonblock();

    return 0;
}
```

H.4 Kalibrierprogramm: marciecalib.cc

```
/*
****************************************
* program for MARCIE´s calibration     *
* (c) 1999 RCS TU-Munich               *
*                                      *
* last change: 1999-12-14 Robin Gruber *
* email: gruber@rcs.ei.tum.de          *
****************************************
*/

#include "marcie.h"
#include <stdio.h>
#include <termios.h>
#include <unistd.h>

#define DEVICE "/dev/ttyS1"

#define MAXCOUNT 100000
#define MINCOUNT 800
#define MINIMUMVALIDITY 0.3
#define ERROR(x)  fprintf(stderr,"marciecalib: error: %s \n",x)
#define INFO(x)  fprintf(stderr,"marciecalib: %s\n",x)
#define VERBOSE(x) x



struct termios saved_attributes, tattr;

typedef struct {
    double realangle;
    double validity;
    double amplitude;
} caltabs;


void restoreterminal(void)
{
    /* Restore the old terminal mode */
    tcsetattr (STDIN_FILENO, TCSANOW, &saved_attributes);
}


void modifyterminal(void)
{
    /* Make sure stdin is a terminal. */
    if (!isatty (STDIN_FILENO))
        {
            ERROR("stdin is not a terminal!");
            exit (-1);
        }
    /* Save the terminal attributes so we can restore them later. */
    tcgetattr (STDIN_FILENO, &saved_attributes);
    /* Set the funny terminal modes. */
    tcgetattr (STDIN_FILENO, &tattr);
    tattr.c_lflag &= ~(ICANON|ECHO); /* Clear ICANON and ECHO. */
    tattr.c_cc[VMIN] = 0;
    tattr.c_cc[VTIME] = 0;
    tcsetattr (STDIN_FILENO, TCSAFLUSH, &tattr);
    atexit(restoreterminal);
}


int getkey(char & c)
{
    if (read(STDIN_FILENO, &c, 1)!=1) {return 0;}
    return 1;
}


int main(int argc, char * argv[])
{
    unsigned char status;
    signed short int ns,ew;
    double angle;
    long unsigned int clock;
    signed short * nsmap, * ewmap;
    double * anglemap;
    char c;
    int count;
    int i,k, period, north;
    double corr, maximum, minimum;
    caltabs caltab[361];
    int entry;
    double validity,realangle;
    double nsdc, ewdc;
    FILE * calfile;
    char * filename;

    // allocate memory
    nsmap = (signed short *)malloc(MAXCOUNT*sizeof(short));
    ewmap = (signed short *)malloc(MAXCOUNT*sizeof(short));
    anglemap = (double*)malloc(MAXCOUNT*sizeof(short));
```

```
if ((nsmap==NULL)||(ewmap==NULL)) {
    ERROR("Cannot allocate enough memory!");
    exit(-1);
}

// get filename from command-line
if (argc<2) {
    INFO("no calibration-file specified, assuming  marcie.cal");
    filename="marcie.cal";
} else {filename=argv[1];}

// open calibration file for writing
if ((calfile=fopen(filename,"w"))==NULL) {
    ERROR("Cannot create calibration-file");
    exit(-1);
}


marcie_init(DEVICE);
modifyterminal();
marcie_setcontmeas(MARCIE_OFF);
marcie_setmeasspeed(2);
marcie_setmeascount(32);

INFO("Start rotation of robot, wait until speed ist constant, then press any key!");

count=0;
while (!getkey(c));
INFO("wait 2.1 to 2.5 rotations, then press key again!");
marcie_setcontmeas(MARCIE_ON);

while (!getkey(c)) {
    marcie_getrawvalue(ns,ew,clock,status,MARCIE_ON);
    if (ew>=0) {
        angle=((atan(double(ns)/double(ew))+M_PI/2)/M_PI*180);
    } else {
        angle=((atan(double(ns)/double(ew))+3*M_PI/2)/M_PI*180);
    }
    VERBOSE(printf("Count: %5d, NS: %6d, EW: %6d Winkel: %6.1f clock:%9u \n",
           count,(int)ns,(int)ew,angle,(int)clock);)
    nsmap[count]=ns;
    ewmap[count]=ew;
    anglemap[count]=angle;
    if (count<MAXCOUNT) {
        ERROR("rotation too slow! Repeat calibration with higher rotational-speed!");
        exit(-1);
    }
    count++;
}
if (count<MINCOUNT) {
    INFO("warning: rotation is possibly too fast.");
}
printf("%d measurements performed.\n",count);
printf("performing autocorrelation to determine period...\n");
maximum=0.0;
period=0;

// autocorrelate the signal to determine period
for (i=count/4; i<count/2; i++) {
    corr=0.0;
    for (k=0; k<count/2; k++) {
        corr=corr+nsmap[k]*nsmap[k+i]+ewmap[k]*ewmap[k+i];
    }
    VERBOSE(printf("shift: %5d, correlation: %8.1f\n",i,corr);)
    if (corr>maximum) {maximum=corr; period=i;}
}

printf("periode is %d!\n",period);

// calculate DC-value
nsdc=0.0;
ewdc=0.0;
for (i=0; i<period; i++) {
    nsdc+=double(nsmap[i]);
    ewdc+=double(ewmap[i]);
}
nsdc/=double(period);
ewdc/=double(period);
printf("DC-value: NS: %6.1f EW: %6.1f\n",nsdc,ewdc);

// remove DC-value from signal and calculate angle
for (i=0; i<count; i++) {
    nsmap[i]-=int(nsdc);
    ewmap[i]-=int(ewdc);
    if (ewmap[i]>=0) {
        anglemap[i]=((atan(double(nsmap[i])/double(ewmap[i]))+M_PI/2)/M_PI*180);
    } else {
        anglemap[i]=((atan(double(nsmap[i])/double(ewmap[i]))+3*M_PI/2)/M_PI*180);
    }
    VERBOSE(printf("count: %5d, NS: %6d, EW: %6d angle: %6.1f \n",
                  i,nsmap[i],ewmap[i],anglemap[i]);)
}

// find zero-crossing of EW-value when NS-value is negative --> approximately north
minimum=30000.0;
```

```
    north=0;
    for (i=0; i<period; i++) {
         if ((nsmap[i]<0)&&(abs(ewmap[i])<minimum)) {minimum=abs(ewmap[i]); north=i;}
    }
    printf("north is at %d\n",north);

   // create calibration-table
    for (i=0; i<=360; i++) {
        caltab[i].realangle=0.0;
        caltab[i].validity=0.0;
        caltab[i].amplitude=0.0;
    }
    for (i=0; i<=period; i++) {
        entry=(int)rint(anglemap[i+north]);
        validity=1.0-fabs(2.0*(double(entry)-anglemap[i+north]));
        realangle=double(i)/double(period)*360.0;
        // verify angle
        if (!(((entry<90)&&(realangle>180))||((entry>270)&&(realangle<180)))) {
            caltab[entry].validity+=validity;
             caltab[entry].realangle+=realangle*validity;
            caltab[entry].amplitude+=sqrt(double(nsmap[i+north])*double(nsmap[i+north])+
                                      double(ewmap[i+north])*double(ewmap[i+north]))*validity;
        }
    }

    // verify the resulting values, interpolate if nessecairy
    for (i=0; i<=360; i++) {
        // value #i is not present or too fuzzy --> interpolation
        if (caltab[i].validity<MINIMUMVALIDITY) {
            if ((i>0) &&(i<360)) {
                if ((caltab[i-1].validity>=MINIMUMVALIDITY)
                     &&(caltab[i+1].validity>=MINIMUMVALIDITY)) {
                    // linear interpolation between the left and right neighbour
                    caltab[i].realangle+=(caltab[i-1].realangle+caltab[i+1].realangle)/2;
                    caltab[i].validity +=(caltab[i-1].validity +caltab[i+1].validity )/2;
                    caltab[i].amplitude+=(caltab[i-1].amplitude+caltab[i+1].amplitude)/2;
                    VERBOSE(printf("interpolation\n");)
                }
                else {
                    ERROR("at least one value cannot be calculated!");
                    ERROR("Repeat measurement with lower rotational speed!");
                    //exit(-1);
                     printf("*\n");
                 }
             }
            else {
                if ((i==0)&&(caltab[1].validity>=2*MINIMUMVALIDITY)) {
                    caltab[0].realangle=0.0;
                    caltab[0].validity =caltab[1].validity;
                    caltab[0].amplitude=caltab[1].amplitude;
                    VERBOSE(printf("interpolation left end!\n");)
                } else if ((i==360)&&(caltab[359].validity>=2*MINIMUMVALIDITY)) {
                    caltab[360].realangle=359.9999;
                    caltab[360].validity =caltab[359].validity;
                    caltab[360].amplitude=caltab[359].amplitude;
                    VERBOSE(printf("interpolation right end!\n");)
                } else {
                    ERROR("at least one value cannot be calculated!");
                    ERROR("Repeat measurement with lower rotational speed!");
                    //exit(-1);
                     printf("*\n");
                }
             }
        }
        VERBOSE(printf("angle: %3d, validity: %3.1f, real angle: %5.1f, amplitude: %6.0f\n",
                       i,caltab[i].validity,caltab[i].realangle/caltab[i].validity
                       ,caltab[i].amplitude/caltab[i].validity);)
    }

    // write the calibration-table to disk
    fprintf(calfile,"*** Calibration file for MARCIE v0.8 ***\n");
    fprintf(calfile,"*** This file is generated automatically! Do not edit! ***\n");
    fprintf(calfile,"NSDC: %6d , EWDC: %6d\n",int(nsdc),int(ewdc));
    for (i=0; i<=360; i++) {
        fprintf(calfile,"%5.1f, %5d\n",caltab[i].realangle/caltab[i].validity,
               int(caltab[i].amplitude/caltab[i].validity));
    }
    fprintf(calfile,"*** End of automatically generated part ***\n");
    fprintf(calfile,"*** Comments can be added below! ***\n");
    fclose(calfile);
    marcie_setcontmeas(MARCIE_OFF);
    return 0;
}
```

Anhang I
Technische Daten

Bauteilekosten:	ca. 50 Euro
Betriebsspannung:	11–14 V Gleichspannung (8..9V ohne IC 6. Siehe A.4.)
Stromaufnahme:	max. 50mA
Empfindlichkeit:	ca. 320/ A/m
Datenübetragungsrate:	19200 Baud 8 Datenbits 1 Stopbit, keine Parität
Fehler:	< ±2° bei homogenem Magnetfeld, mit Kalibrierung

Literatur

[Atm] Atmel: http://www.atmel.com
[Phil] Philips: http://www−us.semiconductors.philips.com
[Tex] Texas−Instruments: http://www.ti.com
[Cads] Cadsoft: http://www.cadsoft.de
[Elr95] ELRAD Heft 3/95