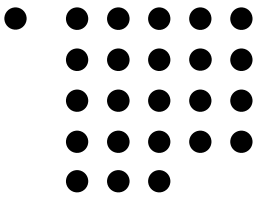


Fachhochschule Köln
University of Applied Sciences Cologne

07 Fakultät für Informations-, Medien- und Elektrotechnik

Studiengang Technische Informatik

Institut für Nachrichtentechnik
Labor für Digitaltechnik

Bachelorarbeit

Thema: Steuerung eines autonom fahrenden Roboters auf der Basis von RFID-Positionsmarken und Richtungsinformationen

Student : **Ralph Erdmann**

Matrikelnummer: 11056064

Referent : Herr Prof. Dr.-Ing. Hartung

Korreferent : Herr Dipl.-Ing. Krawutschke

Abgabedatum : 14. Juli 2009

Hiermit versichere ich, dass ich diese Arbeit selbständig angefertigt und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe.

_____

Ralph Erdmann

Inhaltsverzeichnis

1 Einleitung

Die Grundlage für diese Arbeit ist das opensource Roboterkettenfahrzeug SRV1 der Firma Surveyor [www-surveyor]. Im Auslieferungszustand ist es möglich das Fahrzeug über ein Steuerprogramm fernzusteuern. Dies entspricht dem Funktionsumfang eines normalen ferngesteuerten Fahrzeugs, erweitert um eine Kamera, jedoch ohne jegliche Autonomie oder Intelligenz. Da das Fahrzeug auf einem embedded[1] Board aufgebaut ist, welches über eine WLAN[2] Verbindung verfügt, bietet es einen guten Ausgangspunkt für Erweiterungen.

Ziel dieser Arbeit ist es nun eine Grundlage für autonome Fahrten zu schaffen.
Als Betriebssystem wird uClinux, ein Linux für embedded Plattformen, genutzt. Das Fahrzeug soll um einen zur Orientierung dienenden Kompass erweitert werden. Weiterhin kommt ein RFID[3]-Reader zum Einsatz. Mithilfe dessen werden RFID-Transponder[4] auf einer Teststrecke als künstliche Landmarken zur Positionsbestimmung genutzt. Es wird eine Software für autonome Fahrten entwickelt, die es ermöglicht, durch Verwendung der erweiterten Hardware, den Roboter eigenständig Punkte der Teststrecke anfahren zu lassen. Hierfür ist es erst notwendig alle Hardware im Betriebssystem, beispielsweise durch Treiber, verfügbar zu machen.

[1] In die Anwendung integrierte / eingebettete elektronische Schaltung, die möglichst optimal an die Aufgabe angepasst ist. Verfügt über limitierte Ressourcen.
[2] WLAN, oder auch IEEE 802.11, ist ein bekannter Standard zur drahtlosen Übertragung unter dem Fokus hoher Datenraten, beispielsweise im Bereich von PCs
[3] Radio Frequency Identification, eine Technik zur kontaktlosen Datenübertragung zwischen Datenträger und Lesegerät, siehe auch 4.2.1 “Grundlagen RFID”.
[4] Datenträger der RFID-Technik, im Folgenden auch “RFID-Tag” oder nur kurz “Tag” genannt

Ein Anwendungsgebiet für ein Fahrzeug dieser Art sind Botenfahrten. Portiert man die Ergebnisse auf andere Fahrzeugplattformen, so sind viele andere Anwendungen denkbar, beispielsweise als Rettungsfahrzeug oder im Haushalt.

Im Kontext des Roboterfahrzeugs wurde bereits eine Diplomarbeit verfasst [moedrath09]. Das Hauptaugenmerk lag auf dem Entwurf eines Steuerungssystems für Fahrzeuge dieser Art. Hier wurden auch verschiedene Möglichkeiten der Positionsbestimmung erläutert. Die Möglichkeiten zur Kopplung mit der bereits abgeschlossenen Arbeit werden betrachtet.

Auch sollen Mittel zum debugging[5] der Software gefunden werden, um ein professionelles Arbeiten zu ermöglichen.

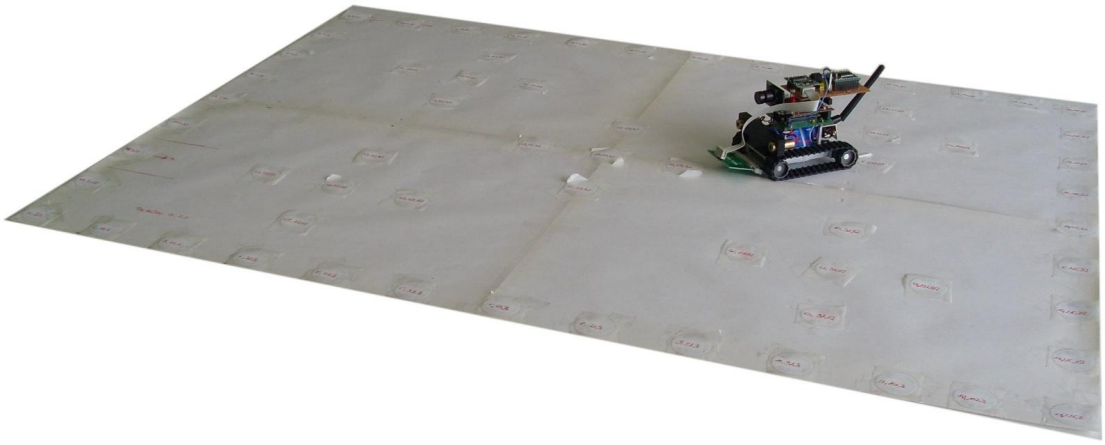


Abbildung 1.1: Übersicht - Der erweiterte Roboter auf der Teststrecke

[5] Fehlersuche und -behebung

2 Grundversion des Roboterfahrzeuges SRV-1

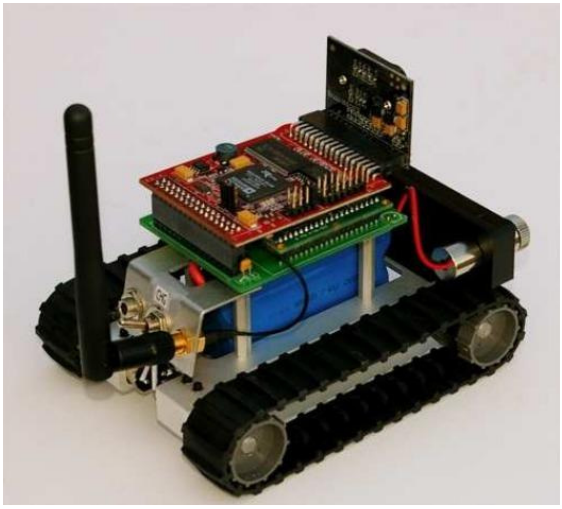


Abbildung 2.1: Roboter SRV1 Grundversion Seitenansicht

2.1 Ausstattung

Die folgende Liste enthält die wichtigsten Hardwarebestandteile:

Prozessor: Analog Devices Blackfin BF537, 1000Mips, 500MHz

Arbeitsspeicher: 32 MB SDRAM

Festspeicher: 4MB SPI Flash

Schnittstelle: JTAG (Joint Test Action Group)

Timer

Kamera: Omnivision OV9655, 1,3 Megapixel, Auflösung von 160x128 bis 1280x1024

Ser. Schnittstelle: 2 UART (Universal Asynchronous Receiver Transmitter)

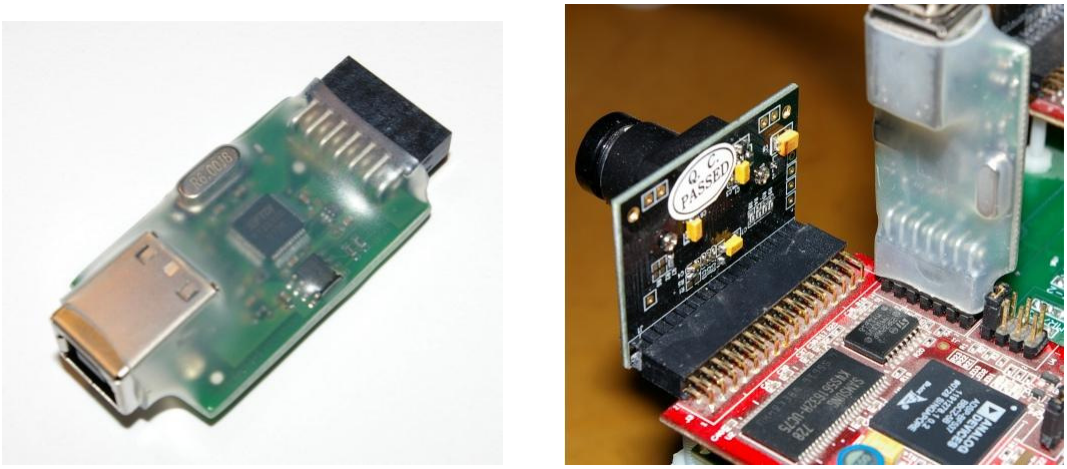
Serieller Bus: SPI (Serial Peripheral Interface), I^2C (Inter-Integrated Circuit)

Par. Eingänge: 16GPIO (General Purpose Input/Output)

Antrieb: 4 Präzisions DC Getriebemotoren

WLAN: Seriell zu WLAN Adapter – Lantronix Matchport, WLAN 802.11g

2.2 ICEBear - Programmier- und debugging-Adapter



(a) Oben (b) Angeschlossen

Abbildung 2.2: ICEBear JTAG Adapter

Zu dem Roboter gehört der ICEBear JTAG Adapter. Es ist ein JTAG zu USB Verbinder der Firma Section5 ([www-section5]), entworfen für Blackfin CPUs von Analog Devices.

JTAG (Joint Test Action Group) entwickelte den Standard IEEE 1149.1. Das JTAG-Protokoll ermöglicht das Programmieren, Debuggen und Testen von ICs, Prozessoren und FPGAs direkt in der Schaltung.
Sein Hauptanwendung findet der Adapter in der Datenübertragung zwischen Roboter und Entwicklungs-PC, die debugging Möglichkeiten werden in Kapitel 5 untersucht.

2.3 Grundkonfiguration

Die nötigen Schritte zur Grundkonfiguration des Roboters, insbesondere der Kommunikation (UART, WLAN, IP-Adresse, Netzbereich ...) sind bereits in [moedrath09] beschrieben.

Hier sei auf das dritte Kapitel "*Surveyor SRV-1 - Das Roboterfahrzeug*" verwiesen, im besonderen auf folgende Abschnitte:

- 3.2 Die UART Schnittstellen
- 3.3 Der Matchport
- 3.3.1 Arten der Konfiguration
- 3.3.1.1 Konfiguration über die Serielle Schnittstelle
- 3.3.2 Wichtige Einstellparameter des Matchports
- 3.3.2.1 Server
- 3.3.2.2 Channel 1 & Channel 2
- 3.3.2.3 WLAN
- 3.3.2.4 Defaults, Exit without save, Save and exit

Zur besseren Übersicht befinden sich die genannten Abschnitte als Kopie im Anhang unter "12.1 Grundkonfiguration des Roboters".

3 Erweiterung Teil 1 - Das Betriebssystem uClinux

In diesem Kapitel soll ein Überblick über Linux auf eingebetten Systemen gegeben werden. Weiterhin enthält es eine Anleitung zur Installation und Konfiguration von uClinux auf dem Roboter, genauer auf dem Blackfin BF537 Board, sowie der Einrichtung einer passenden Entwicklungsumgebung. Ebenso werden die wichtigsten Programme und Arbeitsschritte erläutert.
Dieses Kapitel setzt voraus, dass die unter 2.3 erklärte Grundkonfiguration abgeschlossen ist und der Roboter im Netzwerk erreichbar ist.

3.1 Grundlagen

3.1.1 Linux in embedded Systemen

So genannte "eingebettete" Systeme (Embedded Systems) sind in die Anwendung integriert (eingebettet) und bestehen aus einer elektronischen Schaltung, die möglichst optimal an die Aufgabe der Anwendung angepasst ist. Daher sind die Ressourcen (Speicher, Prozessor) auf einem Embedded System oft limitiert. Die Software dieser Anwendung ist optimal auf den Einsatz angepasst und nahtlos in die Applikation integriert. In vielen Fällen besteht sie nur aus den Komponenten, die für das Ausführen der Anwendung wirklich nötig sind, und ist von allem überflüssigen Ballast befreit.
Die Vorteile und Stärken von Linux können auch im Embedded Bereich gut genutzt werden. Immer öfter ist es auch für Embedded Anwendungen notwendig, über das Internet zu kommunizieren. Linux stellt bereits eine Reihe von wichtigen Protokollen und Pro-

grammen zur Verfügung. Dazu kommen Multi-Tasking und Multi-User-Fähigkeit sowie die sprichwörtliche Stabilität. (Nach [brinker07] Seite 20.)

3.1.2 uClinux als Embedded Linux

uClinux ist eine der am meisten genutzten Embedded-Linux-Varianten. Die Namensgebung "uClinux" kombiniert die Abkürzung uC (eigentlich µC) für Micro-Controller mit Linux. Es ist ein Betriebssystem, das speziell auf Systeme ohne MMU (Memory Management Unit) zugeschnitten ist. Verwendet wird die auch auf Desktop Systemen aktuelle Kernel-Serie 2.6.x.
Es findet eine für Blackfin Prozessoren portierte uClinux Variante Verwendung, welche auf der Homepage des Projekts [www-uclinux] frei zum Download bereitgestellt wird.

3.1.3 Besonderheiten eines Systems ohne MMU

Spezialisierte Mikroprozessoren für Embedded Anwendungen unterstützen keinen virtuellen Speicher (Virtual Memory - VM), da sie keine Speicherverwaltung (Memory Managment Unit - MMU) haben. Das ist ein Kompromiss für einfacheres Chip-Design und geringere Produktionskosten. Viele Embedded Anwendungen haben keinen Plattenspeicher und nur begrenzten Arbeitsspeicher, der keine komplexe Speicherverwaltung benötigt. Da somit keine Speicherschutz mittels Hardware erreicht werden kann, muss insbesondere das Betriebssystem sehr sorgfältig entworfen und ausgiebig getestet sein, was jedoch für den Einsatz von Linux spricht.
Unter uClinux gibt es keinen virtuellen Speicher. Das bedeutet, es ist nicht ohne Weiteres möglich, Speicher zu einem bereits laufenden Prozess hinzuzufügen. Die Benutzung von virtuellem Speicher setzt das vorhandensein einer Speicherverwaltung (MMU) voraus, daher ist uClinux oft mit dem Attribut NOMMU verbunden. (Nach [brinker07] Seite 23.)

3.2 Die Entwicklungsumgebung

Workflow Übersicht Das Betriebssystem für den Roboter wird auf dem Entwicklungs-PC konfiguriert. Das fertig konfigurierte System liegt dann als Image Datei vor, welche in den nicht-flüchtigen Flash Speicher des Roboters übertragen werden muss.

Im Flash-Speicher muss sich ebenfalls ein Bootloader befinden. Dieser lädt das Linux-Image in den flüchtigen SDRAM des Roboters und startet den Bootvorgang. (Alle Änderungen, die man zur Laufzeit am Betriebssystem vornimmt gehen also mit Ausschalten des Systems verloren).

Bootloader und Linux-Image müssen sich den 4 MByte großen Festspeicher teilen. Der Flash-Speicher ist in 64 Sektoren à 64 KByte aufgeteilt. Die ersten 3 Sektoren sind für den Bootloader reserviert, so dass 61 Sektoren = 3,8125 MByte für das Linux-Image bleiben.

Zusätzliche (hier: selbstgeschriebene) Programme können entweder fest ins Image eingebunden werden oder "on the fly" im Betrieb übertragen werden.

Wie man die genannten Schritte durchführt ist in den nächsten Abschnitten beschrieben.

Entwicklungs-PC Als Entwicklungs-PC dient ein Ubuntu Linux System der Version 8.10. Generell ist hier auch jede andere Linux Distribution denkbar, in den Debian basierten Sourcen (die auch Ubuntu nutzen kann) finden sich allerdings einige Pakete von Analog Devices und Section5.

In diesen Quellen befinden sich auch die uClinux Quellen mit Toolchain, allerdings nur das letzte offizielle Release 2008R1.5-RC3. Gearbeitet wurde mit der uClinux Version trunk-svn-7974. Das Kürzel SVN (Subversion) lässt erkennen, dass es sich nicht um einen offiziellen Release Kandidaten handelt, sondern um eine Zwischenversion auf dem Weg dahin. Eine SVN hat den Vorteil, dass die aktuellsten Programmversionen enthalten sind, allerdings mit dem Risiko, dass diese noch nicht richtig getestet worden sind und Fehler enthalten können.

Auf diese "Schnappschuss" Version wurde umgestiegen, da im letzten offiziellen Release der benötigte GDBserver (siehe dazu Kapitel 5 "debugging Möglichkeiten") einen Fehler hatte, der in der SVN Version behoben wurde. Weiterhin verfügt die SVN Version über zusätzliche I^2C Tools, die im letzten Release noch nicht vorhanden waren. (Mit diesen Tools lassen sich grundlegende Operationen durchführen, wie beispielsweise alle Busteilnehmer anzeigen zu lassen.)

Benötigt werden die uClinux Quellen sowie die passende Toolchain. Beides steht auf der uClinux Homepage zum Download bereit [www-uclinux]. Die in dieser Arbeit verwendeten Quellen befinden sich ebenfalls auf der beiliegenden DVD.

uClinux Quellen Die Datei *uclinux-dist-trunk-svn-7974.src.tar.gz* kann an beliebiger Stelle entpackt werden. (Beispielweise im Home Verzeichnis). Bevor die Quellen allerdings genutzt werden können müssen diese konfiguriert werden, was im anschließenden Abschnitt beschrieben wird.

Toolchain Die Toolchain ist in zwei Dateien aufgeteilt.

blackfin-toolchain-uclinux-SVN.i386.tar.gz

blackfin-toolchain-linux-uclibc-SVN.i386.tar.gz

Hier ist auf das Kürzel SVN zu achten, damit die Toolchain auch zu den Kernel-Quellen passt. Die Dateien werden entpackt nach */opt/uClinux/*.

Anschließend wird die Path Variable des Benutzers entsprechend angepasst um die Tools von überall nutzen zu können. Die Datei *.bashrc* im Home Verzeichnis des Nutzers wird um folgende Zeile erweitert:

export PATH=$PATH:/opt/uClinux/bfin-uclinux/bin:/opt/uClinux/
bfin-linux-uclibc/bin

Die Toolchain ist fertig eingerichtet.

Kompiliert wird linuxtypisch mit Gcc (GNU-Compiler), allerdings in der für Blackfin portierten Version *bfin-uclinux-gcc*.

Zusatzprogramme Um die Entwicklungsumgebung abzurunden sind noch einige Zusatztools zu installieren.

Bevor dies über das Debian / Ubuntu Packetsystem geschehen kann sind folgende Quellen in die Datei */etc/apt/sources.list* hinzuzufügen:
deb http://download.analog.com/27516/distros/debian stable main
deb http://www.section5.ch/debian etch main contrib non-free
deb http://ftp.de.debian.org/debian etch main

Nachdem dies geschehen ist und ein *apt-get update* durchgeführt wurde können nun mit den folgenden Befehlen die benötigten Zusatzprogramme installiert werden.

apt-get install icebear-gdbproxy bfinsight bfloader libbfemu-dev - Installiert Tools von Section5 die für den Betrieb des ICEBear Adapters benötigt werden.
apt-get install lrzsz - Tools zum Datenaustausch.
apt-get install libncurses5-dev - Zur Kernelkonfiguration benötigte Bibliotheken.

3.3 Anpassen und Compilieren des Kernels

3.3.1 Kernelkonfiguration

Die Konfiguration des Kernels wird wie folgt Schritt für Schritt durchgeführt:
In einer Konsole begibt man sich in das Verzeichnis, in dem die Kernelquellen entpackt vorliegen, und nimmt die Konfiguration vor:
- *make menuconfig*
Konfigurationsmenü öffnet sich:
- Vendor/Product Selection Surveyor als Hersteller und SRV1 als Board wählen
- Kernel/Library/Defaults Selection den Punkt Customize Kernel Settings anwählen
- Customize User Settings anwählen
- exit, exit, yes
Das Menü schließt sich, als nächstes folgt eine (einmalige!) Konfiguration durch Fragen in der Konsole:
- Hier finden jeweils die Standardwerte Verwendung, weshalb jede Frage mit Enter bestätigt werden kann (Es handelt sich um sehr viele Fragen ...).
Das Menü zur Kernelkonfiguration öffnet sich, hier wird folgende Auswahl getroffen:

- kernel conf:
- Device Driver -> Character Devices -> Serial Drivers -> Enable UART1 , um den zweiten UART zu aktivieren
- exit exit

Das Menü zu den User Settings öffnet sich:

- application conf:
- Miscellaneous Applications -> gdbserver (old)
- Flash Tools -> mtd-utils abwählen(!) das Packet ist defekt, es führt zu Build Errors
- Blackfin app programs -> I2C Tools
- BusyBox -> Busybox Settings -> Busybox Library Tuning ->

-> Command line editing

-> vi-style line editing commands

-> Tab completion

-> Fancy shell prompts

- exit, exit, exit, yes

Man befindet sich wieder auf der Konsole und kann nun mit einem *make* den eigentlichen Build durchführen.

Der Make-Vorgang sollte nun ohne Fehler abgearbeitet werden. Das fertige Kernel Image befindet sich nun unter dem Namen *uImage.ext2* im Verzeichnis *images*. (Hier empfiehlt es sich nach einem erfolgreichen Build die Image Datei an einen anderen Ort zu kopieren und mit einem aussagekräftigen Namen zu versehen (*uImageRalphSVN7974-260609.ext2* als Beispiel) um später einen Überblick über verschiedene Versionen zu haben.

Sollte ein Fehler bei der Konfiguration auftreten kann mit einem *make clean* von neuem begonnen werden.

Das fertige Image kann nun auf den Roboter geflasht werden.

3.3.2 Zusatzprogramme fest einbinden

Um selbstgeschriebene Zusatzprogramme in das Kernel-Image einzubinden geht man wie folgt vor:

Das Dateisystem befindet sich im Verzeichnis romfs. Hier finden sich alle von Linux gewohnten Verzeichnisse (etc, bin, sys, var ...). Romfs bildet das Root Verzeichnis (/) des fertigen Linux Systems.
Hier kann man nun frei seine Daten ablegen. Um selbstgeschriebene Programme klar von den restlichen zu trennen wurden diese im Verzeichnis *myprogs* abgelegt.
Wenn alle benötigten Programme kopiert sind kann man mit dem Befehl *make image* im Hauptverzeichnis das Image neu erstellen. Dieser Befehl bezieht sich rein auf das Image, Kernelkonfigurationen werden hier nicht verändert.
Das fertige Image befindet sich an schon genannter Stelle mit besagtem Namen.

3.3.3 path Variable anpassen

Um das Arbeiten mit den eigenen Programmen auf dem Roboter zu erleichtern wurde das Verzeichnis *myprogs* wie folgt in die Path Variable des Systems eingebunden. (Path Variable des uClinux Systems welches auf dem Roboter ausgeführt wird, nicht die Path Variable auf dem Entwicklungs-PC.)
Im Hauptverzeichnis von *romfs* wird die Datei *.profile* mit folgendem Inhalt angelegt:
export PATH=$PATH:/myprogs/
Die Datei *.profile* wird bei jedem Login des Benutzers Root gelesen. Der Befehl fügt den Ordner *myprogs* zu den bestehenden Ordnern in der Variable *PATH* hinzu.
Durch diese Änderung kann man aus jedem Ordner seine selbst geschriebenen Programme ohne Pfadangabe aufrufen.

3.3.4 Optimierung des Platzverbrauchs

Abhängig von der Größe der selbstgeschriebenen Programme kann es sinnvoll sein die Größe des uClinux Images zu optimieren. Zum einen kann man in der Kernelkonfiguration (*make menuconfig*) nicht benötigte standard Programme abwählen. Die Ersparnis liegt hier aber im Bereich von 100 KByte.
Deutlich größeres Potential von mehreren MByte bieten die Librarys. Diese befinden sich im Filesystem unter *romfs/lib/*. Hier muss aber vorher kritisch bewertet werden welche

Librarys nicht benötigt werden.
Das Löschen der Fortran Bibliotheken (*rm libgfortran**) brachte eine Ersparnis von 700 KByte.

3.4 Telnet Verbindung zum Roboter aufbauen

Der Syntax ist wie folgt:
telnet <ip Adresse srv1> <port, standardmäßig 10001>
Ist die Verbindung geöffnet kann man mit uClinux arbeiten und mit dem Steuerzeichen *strg+altgr+9* Befehle an die Telnet Sitzung absetzen. Genannt sei hier der Befehl *c* (close) zum schließen der Telnet Session.

3.5 Flashen des Images

Es gibt verschiedene Möglichkeiten das Image von uClinux in den Festspeicher des Roboters zu laden. Zwei davon sollen hier genannt werden.
Eine Möglichkeit ist die Verwendung des ICEBear JTAG Adapters. Eine Alternative dazu ist das Übertragen des Images über eine Telnet Verbindung mit den Programmen loady und sz.
Die ICEBear Methode ist deutlich schneller. Das Übertragen eines 3MB Images dauert ca. 3 Minuten im Gegensatz zu ca. 10 Minuten bei der Telnet Methode. Die Übertragung mit Telnet erfordert allerdings keine zusätzliche Hardware und das Zusatzboard muss nicht abgenommen werden.

3.5.1 Flashen mit ICE-Bear

Verwendung findet das Programm *flashload*, welches Teil der "Blackfin Flash loader tools" aus dem Paket *bfloader* ist. Zum Betrieb muss das Blackfin Board per ICEBear Adapter mit dem Entwicklungs-PC verbunden sein.

Bei der Parametrierung hilft das Skript *flashme.sh*. Dessen Benutzung gestaltet sich wie folgt:

flashme.sh uboot - Schreibt nur den Bootloader in den Flash Speicher des Roboters. Namen und Position des U-Boot-Images ist im Skript hinterlegt.

flashme.sh linux <Name des Linux Images> - Schreibt nur das angegebene Linux Image.

flashme.sh all <Name des Linux Images> - Überträgt Bootloader und das angegebene Linux Image in den Speicher.

In der jetzigen Version des Roboters muss erst das Erweiterungsboard vom Blackfinboard getrennt werden, um den JTAG Anschluss zu erreichen.

3.5.2 Flashen mit Telnet

Um das Image per Telnet zu übertragen muss man erst die Kommandozeile des Bootloaders aufrufen. Dafür öffnet man eine Telnet Verbindung zum Roboter und wartet den normalen Bootvorgang von uClinux ab. Anschließend startet man das Betriebssystem mit dem Befehl *reboot* neu.

Beim Neustart muss man nun folgenden Countdown abpassen:

Hit any key to stop autoboot: 5 4 3 2 1.

Nachdem der Bootvorgang mit einem beliebigen Tastendruck gestoppt wurde, befindet man sich auf der Kommandozeile des Bootloaders.

Die Eingabe-Shell sieht wie folgt aus: *srv1>*

Auf dem Bootloader muss nun mit *srv1> loady* das Programm ymodem-receive aufgerufen werden. Es wartet nun auf eine eingehende Datei. Das Telnet Fenster muss nun mit *strg+altgr+9* und c geschlossen werden.

Vom Client kann man nun mit dem Tool YMODEM file send das Image schicken.

Der Syntax ist wie folgt:

sz -Y –tcp-client <ip Adresse srv1>:10001 <name des images>

(Achtung: Der lange Bindestrich muss durch ein MinusMinus ersetzt werden, Formatierungsproblem)

Die Platzhalter müssen entsprechend ersetzt werden.

Nach erfolgreicher Übertragung kann man den Bootvorgang mit *srv1> bootm* neu starten. (Telnet Verbindung vorher wieder aufbauen) Das neue uClinux Image wird geladen.

3.6 Übertragen von Dateien im Betrieb

Um Daten im laufenden Betrieb an den Roboter zu übertragen, beispielsweise zu Testzwecken, muss auf dem Roboter das Programm *rz* (ZMODEM file receive) aufgerufen werden. Es wartet anschließend auf eingehende Daten auf dem Port der Telnet Verbindung. Damit Dateien empfangen werden können muss die Telnet Verbindung geschlossen werden. Dafür die Telnet Sitzung mit *strg+altgr+9* und c schließen.
Dateien schickt man nun vom Entwicklungs-PC mit dem Programm *sz* (ZMODEM file send) mit folgendem Syntax:
sz –tcp-client <ip Adresse srv1>:10001 <dateiname der zu sendenden datei>
(Achtung: Der lange Bindestrich muss durch ein MinusMinus ersetzt werden, Formatierungsproblem)
Bei Dateiname können auch mehrere Dateien angegeben werden.

3.7 Anpassen des Bootloaders

Nachdem man eigene Programme zum Image hinzugefügt hat, kann es zum Abbruch des Bootvorgangs des Roboters mit einer *"Bad CRC"* Nachricht kommen, obwohl das Image korrekt übertragen wurde und der Flash-Speicher des Roboters nicht voll ist.
Dieser Fehler ist in einer Einstellung des Bootloaders begründet, er lädt nur Images bis zu einer bestimmten Größe, überschreitet ein Image diese Größe quittiert der Loader dies mit der oben genannten Meldung.
Mit der folgenden Anpassung im Quellcode des Bootloaders wurde diese Grenze erhöht: Die Quellen befinden sich auf der beigelegten DVD und stehen ebenfalls auf der uClinux Homepage in Form der Datei *u-boot-1.1.6-2008R1.5.tar.gz* zum Download bereit. Diese können an beliebiger Stelle entpackt werden. Mit dem Befehl *make bf537-srv1_config* wird eine Grundkonfiguration passend zum Blackfin Board durchgeführt.

In der Datei *includes/configs/bf537-srv1.h* muss nun folgende Zeile geändert werden:

Von:

flashboot= eeprom read 0x1000000 0x30000 ***0x320000****;*

nach:

flashboot= eeprom read 0x1000000 0x30000 ***0x3d0000****;*

Die Parameter des Befehls eeprom read sind wie folgt zu interpretieren:

eeprom read **Adresse Offset Größe**

Die Obergrenze für die Größe eines Images wird also von 3276800 Byte / 3,125 MByte (Hex 0x32000 Byte) auf 3997696 Byte / 3,8125 MByte (Hex 0x3d0000 Byte) erhöht.

Anschließend kann man mit einem *make* das Image des Bootloaders erstellen. Das auf dem Roboter verwendete Image, welches sich auch auf der beiliegenden DVD findet, ist entsprechend angepasst.

4 Erweiterung Teil 2 - Hardware

In diesem Kapitel werden die hardwareseitig am Fahrzeug vorgenommen Veränderungen behandelt. Diese Umfassen den Anschluss der Zusatzhardware Kompass und RFID-Reader mit Antenne über das Erweiterungsboard. Die Auswahl des RFID Readers wird erklärt. Zum Abschluss werden die Eigenschaften des erweiterten Fahrzeugs zusammengefasst.

4.1 Der Kompass CMPS03

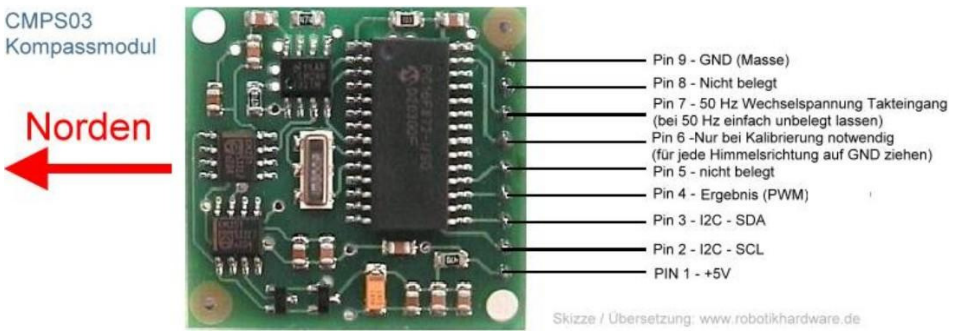


Abbildung 4.1: Kompass CMPS03

Verwendet wird das Kompassmodul CMPS03 der Firma Devantech Ltd. England. Es arbeitet mit 5 Volt Betriebsspannung und bietet laut Datenblatt eine Genauigkeit von 3 bis 4 Grad bei korrekter Kalibrierung. Die Richtungsinformationen können über den I^2C Bus ausgelesen werden (Zu diesem Zweck wird auch ein PWM Signal bereitgestellt, welches aber für diese Arbeit nicht relevant ist).
Die Werte liegen in verschiedenen ein Byte breiten Registern bereit.

Register 1 Wertebereich 0 bis 255, muss entsprechend in einen Gradwert umgerechnet werden. (3599 / 255 * gelesener Wert)

Register 2+3 Wertebereich 0 bis 3599 (0.0 bis 359.9 Grad).

Die Kalibrierung erfolgt durch Ausrichten des Moduls in je eine der Himmelsrichtungen mit anschließendem Reset. Laut Datenblatt spielt die Reihenfolge der Himmelsrichtungen keine Rolle, es wurde sich bei der Kalibrierung allerdings an Norden, Osten, Süden, Westen gehalten. Der Vorgang muss sorgfältig durchgeführt werden. Ein grobes Ausrichten in die Himmelsrichtung reicht hier nicht. Bewährt hat es sich das Modul auf einer mit einem Karten-Kompass ausgerichteten Schablone zu positionieren. Die Kalibrierung muss nur einmalig durchgeführt und sollte nur bei Problemen wiederholt werden.
Das Reset Signal kann auf zwei Arten erzeugt werden. Einmal über Kurzschluss von Pin9(GND) und Pin6(Reset) oder über ein entsprechendes I^2C Signal (Schreiben von 0xFF / 255 in Register 15).
Weiterhin ist auf eine exakte horizontale Ausrichtung des Moduls zu achten, damit es zu keinem Fehlverhalten kommt.

4.2 Der RFID Reader

4.2.1 Grundlagen RFID

Die RFID (Radio Frequency Identification) Technik bietet die flexible Möglichkeit Daten zwischen einem Datenträger und einem Lesegerät kontaktlos zu übertragen. Ein Lesegerät beinhaltet typischerweise ein Hochfrequenzmodul, eine Kontrolleinheit sowie ein Koppelelement zum Empfänger. Meist sind die Lesegeräte mit einer zusätzlichen Schnittstelle (RS232, RS485 ...) ausgestattet, um die Daten an ein anderes System weiterzugeben.
Der Transponder, der eigentliche Datenträger eines RFID-Systems, besteht üblicherweise aus einem Koppelelement (Spule) sowie einem Mikrochip. Außerhalb der Reichweite eines Lesegerätes verhält sich der Transponder, der in der Regel keine eigene Spannungsversorgung besitzt, vollkommen passiv. Erst innerhalb der Reichweite eines Lesegerätes wird der Transponder aktiviert. Die zum Betrieb des Transponders benötigte Energie

wird ebenso wie Takt und Daten durch die Koppeleinheit kontaktlos zum Transponder übertragen. (Vergleiche [finkenzeller06] Kapitel 1 - Einführung) Die Anwendungsgebiete von RFID sind vielfältig. So findet die Technik als Identifikationsverfahren in Form von Barcode-Systemen sowie Chip- und Speicherkarten Verwendung. Ein wichtiges Merkmal ist hier die Frequenz, welche maßgeblich die mögliche Reichweite bestimmt.
In dieser Arbeit soll ein RFID Reader im 13.56MHz Hochfrequenzbereich verwendet werden. Die maximale Reichweite eines solchen Systems wird mit bis zu einem Meter angegeben, es wird aber mit einer Reichweite von bis zu 10 cm gerechnet. Die RFID-Transponder werden wie zu Anfang erwähnt als Landmarken zur Orientierung genutzt.

4.2.2 Auswahl

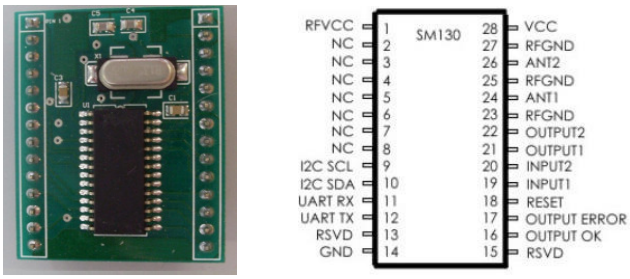
Aus folgender Ausgangssituation musste der Roboter mit einem RFID-Reader ausgestattet werden: Es waren bereits 2 RFID-Reader vorhanden. Zum einen das "TAGnology ACG HF Multi ISO OEM Module" und zum anderen der "openpcd.org RFID-Reader". Beide Reader lassen sich nicht ohne weiteres an den Roboter anschließen, da beide Module standardmäßig die serielle Schnittstelle verwenden, von der der Roboter aber über keine weitere freie verfügt. Das TAGnology Modul bietet ausschließlich eine serielle Schnittstelle, der openpcd.org Reader bietet zusätzlich eine I^2C Schnittstelle, die aber als "for future use" gekennzeichnet ist. Der Roboter bietet für Erweiterungen den SPI und I^2C Bus an. Es stellten sich nun folgende Möglichkeiten:

- a) Einen Wandler von seriell auf I^2C oder SPI entwickeln, um einen der vorhandenen Reader verwenden zu können
- b) weitere serielle Schnittstelle über den I^2C oder SPI Bus schaffen um dann dort einen der vorhandenen Reader anzuschließen
- c) Den openpcd.org Reader über I^2C ansprechen
- d) ein Modul beschaffen, welches den I^2C oder SPI Bus verwendet.

Möglichkeit a) und b) wurden ausgeschlossen, da der Aufwand als zu hoch eingeschätzt wurde. Der Opensource Gedanke der Variante c) bietet ein großes Lernpotential. Da das Hauptaugenmerk dieser Arbeit allerdings nicht auf der Konfiguration eines RFID-Readers liegt und der Zeitaufwand zu hoch wäre, musste auch diese Möglichkeit verworfen werden.
Schlussendlich wurde Möglichkeit d), die Beschaffung eines Moduls passend für einen der Busse, gewählt.
Bei Recherchen stellte sich schnell heraus, dass der Markt für RFID Reader von Modellen mit serieller Schnittstelle dominiert wird und Geräte für den I^2C oder SPI Bus eher wenig verbreitet sind.

4.2.3 Der Reader SM130 13.56 MHz Mifare r/w

Die Wahl fiel auf das Modul "SM130 13.56 MHz Mifare r/w Module" der Firma Son-Micro [www-sonmicro]. Es kann über RS232 und I^2C Bus angesprochen werden, unterstützt allerdings nur Transponder die Mifare Codes verwenden (ISO14443A Mifare Classic 1K, Mifare Classic 4K und Mifare Ultralight). Diese Einschränkung wurde aufgrund der schwierigen Beschaffungslage in Kauf genommen. Das Modul und eine Übersicht der Belegung zeigt Abbildung 4.2. Weiterhin ist von Vorteil, dass nun Kompass und RFID-Reader den gleichen Bus verwenden.



(a) Ansicht (b) Belegung

Abbildung 4.2: RFID Reader SM130

4.2.3.1 Firmware flashen für I²C

Bevor der RFID Reader über den I²C Bus erreichbar ist muss das Modul auf die neueste Firmware gepatcht werden. Die benötigte Software und das Firmware Image sind über den Support von SonMicro zu beziehen und befinden sich ebenfalls auf der DVD. Der Flashvorgang geschieht über die serielle Schnittstelle mithilfe eines Windows Tools.

4.2.3.2 IF232LV RS232 Adapter

Um das Modul über den seriellen Anschluss mit einem PC verbinden zu können muss ein RS232 Wandler genutzt werden. Dieser verringert die PC-seitig genutzten 15V auf für das Modul verträgliche 5V. Genutzt wird das "IF232LV RS232 Interface Module" der Firma MCT Paul & Scherer Mikrocomputertechnik GmbH [www-mct]. Es bietet einen PC kompatiblen 9-pin Sub-D Stecker und arbeitet mit einer Versorgungspannung von 3V bis 5.5V. Hier eine Übersicht über die Pinbelegung, wobei für das Modul nur Pin1 - VCC, Pin4 - RxD, Pin5 - Txd und Pin10 - GND relevant sind:

PIN	Function	Function	Pin
1	**VCC**	DSR	2
3	RI	**RxD**	**4**
5	**TxD**	DTR	6
7	RTS	CTS	8
9	DCD	**GND**	**10**

Folgende Anschlüsse müssen wenigstens verbunden werden um den Flashvorgang durchzuführen: Pin1 und Pin28 des RFID-Moduls, sowie Pin1 des Adapters, müssen mit 5V Versorgungsspannung verbunden werden. Pin14 des Moduls und Pin10 des Adapters müssen mit Masse verbunden werden. Für die Kommunikation werden Pin11 des Moduls mit Pin4 des Adapters (RX-Leitung), sowie Pin12 des Moduls mit Pin5 des Adapters (TX-Leitung) verbunden. Der Adapter wird mit einem 9 poligen seriellen Kabel mit dem PC verbunden.

4.2.3.3 Windows Software

Auf der Hersteller Homepage [www-sonmicro] stehen unter anderem die folgenden Windows Tools zum Download zur Verfügung. Die Programme befinden sich ebenfalls auf der beiliegenden DVD.
Das "Firmware Upgrade Tool" (*sm130_fu.exe*) ist selbsterklärend. Nach Auswahl des Firmware-Images und klick auf Auto-Upgrade wird das Modul geflasht.
Das "SMRFID Mifare Tool" (*smrfid_mifare_v1_2_1.exe*) ist für Testzwecke gut geeignet. Es kommuniziert per serieller Schnittstelle mit dem Modul und bietet grafisch Zugang zu allen Funktionen. Hilfreich ist die Anzeige im unteren Bereich des Interfaces. Hier werden die durchgeführten Operationen in der Konvention der I^2C Befehle angezeigt. So kann man hier die Befehle und Antworten betrachten und daraus Informationen für die Kommunikation unter Linux ziehen. Hilfreich war es benötigte Operationen mit dem Windows Programm durchzugehen und sich das Log der Kommunikation abzuspeichern, um dann die erstellten Linux Programme zu überprüfen.

4.2.3.4 Belegung

Folgende Anschlüsse werden benötigt um die grundlegende Kommunikation mit dem Modul per I^2C aufzubauen (Dem Data Sheet des Moduls entnommen):
1+28 RFVCC/VCC - 5V Supply Voltage
9 I2C SCL - I2C Clock
10 I2C SDA - I2C Data
14 GND - Ground Connection
Um auch die serielle Schnittstelle nutzen zu können kommen hinzu:
11 UART RX UART RX Pin of SM130 (0 - 5V TTL)
12 UART TX UART TX Pin of SM130 (0 – 5V TTL)
Anschluss der Antenne:
23,25,27 RFGND - Ground - Should be connected to ground externally and for better performance can be connected to antenna ground plane
24, 26 ANT1, ANT2 - Antenna Pin to drive and demodulate. Should be connected to one

of the symmetrical antenna/inductor end
Für debugging / LED's:
21 OUTPUT1 / DREADY - Data Ready Pin. Functional if the "Seek For Tag" command was executed previously. A Logic 1 pulse will be sent as soon as valid Mifare Tag enters into the RF Field. Useful to generate interrupt on Master MCU instead of polling I2C continuously.
17 TAGF - Same as Data Ready Pin but with longer pulse. This pin can trigger external device, drive buzzer circuit or simply can be connected to LED indicating there is a valid tag in the field
16 OUTPUT OK / CREAD - This pin indicates status of "Seek for Tag" command. If it is logic high, then it means the Continuous Read ("Seek For Tag") is active and module is searching for Mifare Tag continuously.

Diese und weitere Informationen finden sich im Data Sheet, Application Note und User Manual des Moduls, welche sich auf der DVD befinden. Achtung: in der Application Note sind die Bezeichnungen der Pin Belegungen für den Betrieb mit I^2C teilweise leicht abgewandelt.

4.2.4 Antenne

Die Antenne PA1356-1 wurde passend zum RFID-Modul ebenfalls über SonMicro [www-sonmicro] bezogen. Es ist eine Antenne in Form eines Double Layer PCB Boards mit passender Schaltung, in einer Größe von 55x55 mm mit einer Dicke von 1.6 mm.

4.2.5 RFID Transponder Mifare 1k

Passend zum RFID Reader wurden Transponder der Firma Raflatac beschafft. Diese verwenden den Mifare 1k Standard, sind rund und haben einen Durchmesser von ca. 40mm. [www-raflatac]

4.2.5.1 Grundlagen Mifare

Mifare ist eine von Philips (heute: NXP Semiconductors) entwickelte Transpondertechnik. Der Speicherbereich eines Mifare-Transponders ist in 16 voneinander unabhängige Segmente (Sektoren) aufgeteilt. Jeder Sektor ist durch zwei verschiedene Schlüssel vor unberechtigten Zugriffen geschützt. Über ein Register können unterschiedliche Zugriffsrechte für die beiden Schlüssel vergeben werden. Somit lassen sich bis zu 16 voneinander unabhängige und gegenseitig geschützte Applikation auf dem Transponder unterbringen. (Nach [finkenzeller06] Seite 331).
Ein Schaubild der Speichereinteilung findet sich im Anhang unter "12.2 Schaubild Mifare Speichereinteilung".

4.2.5.2 Speichereinteilung

Ein Mifare 1k Transponder bietet 1 KByte = 1024 Byte Speicher. Der Speicher ist aufgeteilt in 16 Sektoren a 4 Blöcken mit je 16 Byte Speicher. (16 Byte * 4 * 64 = 1024 Byte). Für Nutzdaten steht allerdings weniger Speicher zur Verfügung, da in jedem Sektor ein Block für die Einstellung der Zugriffskontrolle reserviert ist. Weiterhin befindet sich in Block0 des nullten Sektors die Seriennummer.
Es sind also 752 Byte (= 16 Byte * (3 * 16 -1)) für Nutzdaten verfügbar.
In diesem Projekt sollen die 3 Attribute Id Nummer, Koordinate-X und Koordinate-y auf einem Transponder abgelegt werden:
Sektor0 - Block1 - Byte 0 und 1 = **Id**
Sektor0 - Block1 - Byte 2 und 3 = **X-Koordinate**
Sektor0 - Block1 - Byte 4 und 5 = **Y-Koordinate**
Werden die Koordinaten in ganzen Zentimetern mit positivem Vorzeichen abgelegt ergibt sich so durch die 2 Byte Datenbreite die maximale Größe des Koordinatensystems mit 655,35 Quadrat-Metern und 65536 Transpondern. Weitere Informationen für spätere Projekte können nach gleichem Muster in darauf folgenden Speicherbereichen abgelegt werden. Zur Ver- und Entschlüsselung wird der unveränderte Standardschlüssel verwendet.

4.3 Das Erweiterungsboard

4.3.1 Grundlagen I²C Bus

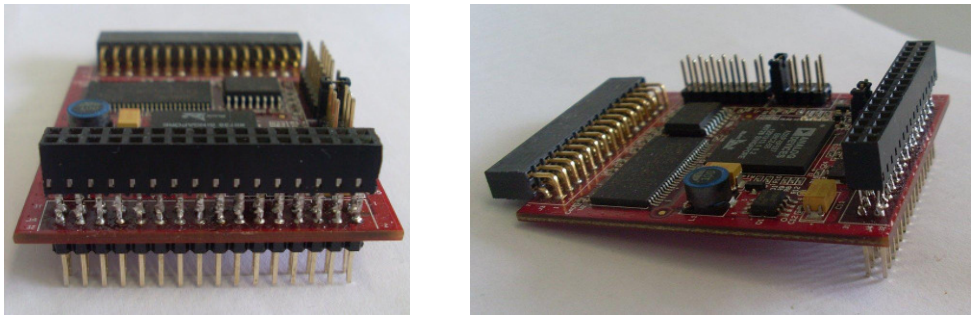
Philips entwickelte diese serielle bidirektionale Zweidraht-Schnittstelle für die Kommunikation zwischen verschiedenen ICs (Integrated Circuit / Integrierter Schaltkreis) in elektronischen Systemen. Daher auch der Name "Inter-IC-Bus", kurz "IIC-Bus" oder I²C-Bus. Das Interface besteht aus den beiden Signalleitungen SDA (serial data) und SCL (serial clock) sowie einer gemeinsamen Masseleitung (ground). Am Bus können mehrere Teilnehmer angeschlossen werden. Jeder der Teilnehmer hat eine vorgegebene sieben Bit lange Adresse (das achte Bit entscheidet, ob gelesen oder geschrieben wird). In den meisten Fällen kontrolliert ein Master die verschiedenen Slaves.
Ein Teilnehmer kann nur Empfänger oder Sender und Empfänger sein. Der Master kontrolliert die Datenflussrichtung und gibt mit dem Clock-Signal die Übertragungsgeschwindigkeit vor. Die SDA- und die SCL-Leitung werden mit Pull-Up Widerständen versehen. Die Bus-Teilnehmer sind mit Open-Drain-Schaltungen an den Leitungen angeschlossen und ziehen im aktiven Zustand die Leitungen gegen GND. Der Bus-Master initiiert eine Schreib- oder Lesesequenz. Dabei beginnt jede Übertragungssequenz mit einer sogenannten Start-Bedingung (SDA geht vor SCL auf 0) und endet mit einer sogenannten Stop-Bedingung (SCL geht vor SDL auf 1).
(Nach [plate07] Seite 189.)

4.3.2 Spannungsversorgung

Die Erweiterungsmodule arbeiten beide mit einer Betriebsspannung von 5V. Um diese bereitzustellen findet ein DC/DC Spannungswandler vom Typ R-78B5.0-1.0 Verwendung (Das Datenblatt befindet sich auf der DVD). Dieser erzeugt aus einer Eingangsspannung von 6.5V bis 30V eine Ausgangsspannung von 5V mit 1 Ampere. Dieser wird mit der Batteriespannung des Roboters von 7.5V betrieben. Die minimale Eingangsspannung des Wandlers von 6.5V stellt kein Problem dar, da eine solch niedrige Spannung nicht ausreicht um das Fahrzeug zu betreiben. (Siehe dazu auch "8.3 Akkulaufzeit").

4.3.3 Verbindung mit dem Blackfin Board

Das Blackfin Board hat zu Erweiterungszwecken einen 32 Pin Expansion Header. Dieser verbindet das Blackfin Board mit dem darunter liegenden Board über eine Steckverbindung. Die Belegung befindet sich im Anhang unter "12.3 Belegung des Expansion Headers des Blackfin Boards".
Um das Erweiterungsboard anschließen zu können wurde auf den Header eine 32 polige Buchse gelötet, in die das Erweiterungsboard eingesteckt werden kann (Gezeigt auf Abbildung 4.3). Das Erweiterungsboard wird mit Pin 14 - I2C CLK (Clock), Pin 15 -



(a) Ansicht 1 (b) Ansicht 2

Abbildung 4.3: Expansion Header um Buchse erweitert

I2C SDA (Data) und Pin 16 - GND (Masse) verbunden. Die Versorgungsspannung wird direkt an der Batterie abgegriffen.

4.3.4 Umsetzung

Aus den genannten Anforderungen entstand das folgende Erweiterungsboard. Abbildung 4.4 zeigt eine schematische Darstellung des Aufbaus, Abbildung 4.5 zeigt das Board in seiner tatsächlichen Umsetzung. Die Widerstände R1 bis R4 (jeweils 20 Ohm) dienen als Schutzwiderstände der Busteilnehmer, R5 bis R7 (jeweils 200 Ohm) bilden die Vorwiderstände der LED's. Die Kapazitäten C1 bis C4 (100 nFarad, 25 µFarad, 100 nFarad, 100 µFarad) dienen der Spannungsstabilisierung und -Glättung und sind an der Referenzbeschaltung des Spannungswandlers angelehnt.

Auf dem Board finden nicht nur das Kompass- und RFID-Modul mit entsprechender Spannungsversorgung Platz, sondern auch Schnittstellen zum RS232 Interface des RFID-Moduls und zum Anschluss der Antenne. Das Board wird über einen Stecker an der Unterseite mit der Buchse des Erweiterungsheaders des Blackfin-Boards verbunden.

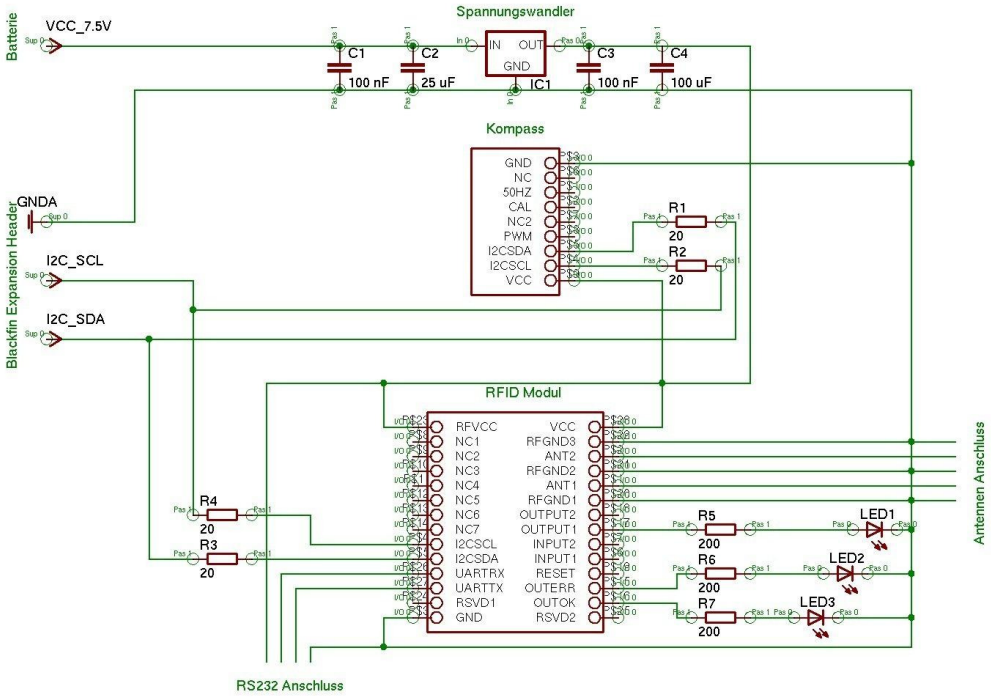
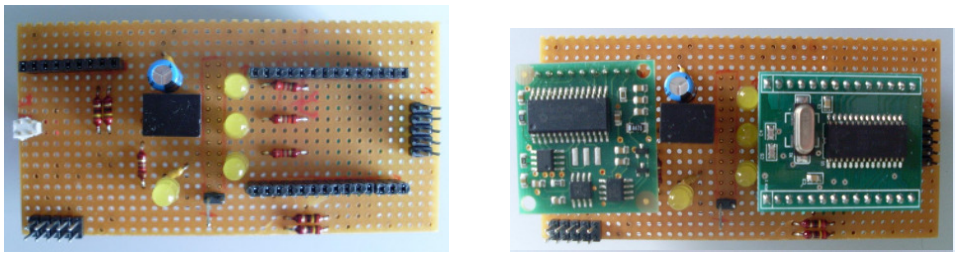


Abbildung 4.4: Erweiterungsboard schematisch

4.4 Positionierung der RFID Antenne

Für die Antenne wäre eine Position zentral unterhalb des Fahrzeuges wünschenswert gewesen, da so sichergestellt wäre, dass sich das Fahrzeug beim Lesen eines Transponders tatsächlich auf eben diesem befindet. Tests haben allerdings ergeben, dass die metallene Unterseite des Roboters die Reichweite der Antenne aufhebt, so dass kein sinnvoller Betrieb möglich ist. Alternativ wurde die Antenne an einer Halterung vor dem Fahrzeug montiert. (Siehe Abbildung 4.6)



(a) leer (b) bestückt

Abbildung 4.5: Erweiterungsboard Bild

4.5 Das erweiterte Fahrzeug

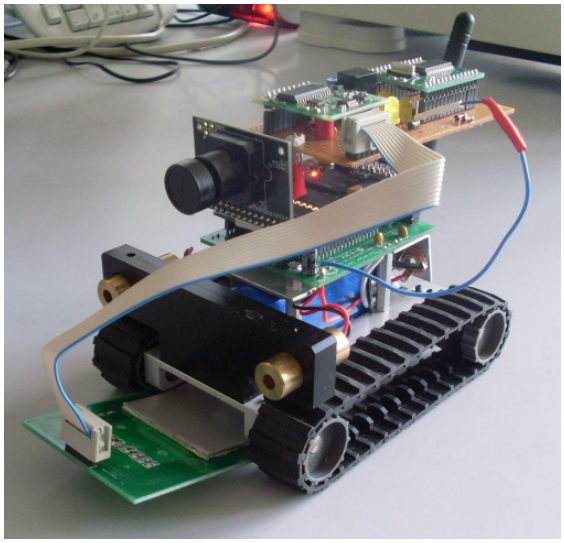


Abbildung 4.6: Roboter SRV1 Erweitert

Die unter 2.1 genannte Ausstattung ist um folgende Punkte erweitert:

Betriebssystem uClinux für Blackfin Prozessor, Version SVN Trunk 7974

Kompass CMPS03, angeschlossen am I^2C Bus

RFID Reader SM130 r/w Modul, angeschlossen am I^2C Bus mit zusätzlichem seriellen Anschluss zu Wartungszwecken

RFID Antenne vor dem Fahrzeug positioniert

Erweiterungsboard Auf diesem sind Kompass und RFID Reader montiert, verfügt über eine Versorgungsspannung von 5V durch Spannungswandler, mit zusätzlichen Status LEDs, sowie Anschlussmöglichkeiten für RS232 und die Antenne

5 Debugging Möglichkeiten

5.1 Vorstellung GDB: The GNU Project Debugger

Der allgemeine debugging Ablauf mit GDB auf einer normalen Workstation gestaltet sich wie folgt:
Vorbereitend müssen die Programme mit bestimmten Flags kompiliert werden, damit der Code für das Debuggen geeignet ist (*-g für debugging Informationen, -O0 um Compiler Optimierungen auszuschalten*). Anschließend wird das Programm mit gdb geladen und es können nun debugging Vorgänge ausgeführt werden (Variablen Überwachen, bis zu bestimmten Haltepunkten ausführen usw.).
Es ist ebenfalls möglich Programme zu debuggen, die auf einem entfernten System ausgeführt werden. Hierfür wird auf dem Zielsystem das Programm per GDBserver geladen, welcher dann einen Port öffnet, auf den sich gdb dann über TCP/IP verbinden kann.
Gdb ist ein command-line Programm wofür aber auch grafische Interfaces wie Insight existieren (hier nicht betrachtet).

5.2 JTAG debugging mit ICEBear

Tests und Rücksprache mit Section5 ergaben, dass der Adapter für unsere debugging Vorhaben nicht geeignet ist. Firmware debugging ist möglich, nicht allerdings das Debuggen von Programmen, die unter uClinux ausgeführt werden. (Martin Strubel, einer der Hauptentwickler von Section5, schrieb in einer E-Mail:*"Fuer User-Space (unter uClinux) benutzt man besser den GDBserver (..) Auf Hardware und Kernel-Level gehts am besten mit dem ICEbear."*) Bfinsight, die von Section 5 portierte Blackfin Variante von

Insight, baut zwar die Verbindung zum Board auf, erkennt allerdings keine laufenden Programme.

5.3 GDB debugging direkt auf dem Blackfin

Wie schon erwähnt ist es mit einer normalen Linux Workstation möglich direkt mit GDB in der Konsole zu debuggen. Eine Möglichkeit wäre gewesen über Telnet eine GDB Session direkt auf dem Blackfin zu starten. Leider ist es nach Aussage eines uClinux Entwicklers nicht möglich und auch nicht vorgesehen GDB auf einem System ohne MMU zu nutzen. (Siehe Forumsbeitrag unter [www-forum1]).

5.4 GDBserver

Die dritte und letzte zu betrachtende Möglichkeit das System mit GDB zu debuggen ist die Verteilung auf GDBserver und Client. Hier war es zu Anfang nötig das uClinux Release zu wechseln. Im letzten offiziellen Release Candidate war ein Fehler im GDB Server, der verhinderte ihn auf einer UART Schnittstelle zu starten. Abhilfe schaffte hier das Update auf einen aktuellen Snapshot.

GDB Server auf dem Blackfin Über den GDBserver wird das zu debuggende Programm auf dem Blackfin geladen. Von den zwei verfügbaren UART Schnittstellen belegt die Telnet Verbindung UART0, die debugging Session wird auf UART1 gestartet. Diese serielle Verbindung ist durch die Umsetzung im Matchport für eine TCP/IP Verbindung als Port nach außen verfügbar.

GDB Verbindung vom Client Vom Client wird nun der Ablauf des Programms auf dem Blackfin gesteuert und der Debug Vorgang durchgeführt.

5.4.1 Einschränkungen

Das System unterliegt aufgrund der 2 UART Schnittstellen gewissen Einschränkungen.

UART0 wird für den Telnet Login benötigt, über Telnet wird der GDBserver auf UART1 gestartet. Beide Schnittstellen sind also belegt.
Zu Einschränkungen kommt es sobald weitere Programme UART1 benutzen sollen. Die in [moedrath09] erstellte Steuerungssoftware benutzt UART1 zum Datenaustausch.
Die serielle Schnittstelle kann nicht zwischen beiden Programmen aufgeteilt werden. Um die Steuerungssoftware trotzdem zu debuggen könnte man diese für die Dauer des debuggings umkonfigurieren, so dass sie keine Daten von UART1 liest, sondern Testdaten aus einer Datei.
Normale Programme können aber auf diesem Wege debugged werden.

5.4.2 Syntax

Hier sei eine kurze Übersicht zum debugging mit GDB genannt.

Programm vorbereiten Das zu debuggende Programm muss wie schon eingehend beschrieben mit den Flags *-g -O0* kompiliert werden.

Schritte auf dem Blackfin Board Per Telnet auf dem Roboter ausführen:

gdbserver /dev/ttyBF1 ./programtodebug

Vom Rechner *bfin-uclinux-gdb programtodebug.gdb*

gdb) target remote <IP address of target system>:<port of the target system>

Beispielsweise:

gdb) target remote 192.168.0.5:10002 (10002 ist der Standardport der zweiten UART Schnittstelle über WLAN)

Folgender Programmstart wird erwartet:

gdb) break main

gdb) continue

Abschließend eine Übersicht der wichtigsten Befehle:

list Anzeigen des Quellcodes, hier werden auch die Zeilennummern angezeigt (wichtig um den nächsten Breakpoint zu setzen)

break ZEILE Breakpoint auf Zeilennummer setzen

cont / continue das Programm bis zum nächsten Breakpoint laufen lassen

print variable Inhalte einer Variable anzeigen

step Schrittweise das Programm ablaufen lassen, taucht in aufgerufene Funktionen ab

next Schrittweise das Programm ablaufen lassen, bleibt in Funktion

help Hilfe, Übersicht weiterer Kommandos

6 Implementierung der Software

Die Software wurde linuxtypisch in C verfasst und mit Doxygen [www-doxygen] dokumentiert. Der Output des Tools befindet sich auf der DVD, so dass in diesem Kapitel auf den generellen Aufbau eingegangen werden kann.
Der Code läuft sequentiell ab, es werden keine Nebenläufigkeiten umgesetzt.
Für eine gute Lesbarkeit wurden Arbeitsschritte so weit wie möglich auf einzelne Funktionen heruntergebrochen. Weiterhin wurden die Funktionen, entsprechend ihrer Verwendung, in mehreren Quelldateien organisiert.
Um eine gute Erweiterbarkeit zu gewährleisten wurden die Funktionsköpfe einheitlich definiert, so dass ein Konfigurationsdatentyp akzeptiert wird. Dieser Datentyp enthält alle wichtigen Parameter, die für den Betrieb des Fahrzeugs wichtig sind. (Siehe nächster Abschnitt). Neue Parameter können so einfach geändert oder hinzugefügt werden, ohne die Funktionsaufrufe zu bearbeiten.
Hardware:
Die Motoren werden von uClinux nicht unterstützt, so dass hier erst ein Treiber geschrieben werden musste, für den die standard Firmware des Roboters als Vorlage diente. Der I^2C Bus wird treiberseitig von uClinux unterstützt, so dass sich hier auf die Kommunikation zu den Busteilnehmern konzentriert werden konnte.

6.1 Softwareübersicht

Trotz der Dokumentation mit Doxygen findet sich hier zusätzliche eine Übersicht der erstellen Datentypen, Funktionen und Programme, damit in den weiteren Abschnitten ein Bezug hergestellt werden kann.

robo-header.h+c Allgemeine Funktionen / Definitionen, hier werden alle weiteren Dateien inkludiert, nur diese Datei muss in einem Programm inkludiert werden. Hier werden die folgenden Datentypen vereinbart:

6.1.1 Datentypen

struct robotdefine Konfigurationsdatentyp des Roboters

char * i2cbus Adresse I^2C Bus (/dev/i2c-0)

unsigned char i2ckompass I^2C Adresse Kompass (0x60)

unsigned char i2crfid I^2C Adresse RFID Reader (0x42)

int motor Handle des Motors, wird mit *open("/dev/motor", O_RDWR);* erstellt

int speedleft Geschwindigkeit linke Kette für Geradeausfahrt

int speedright Geschwindigkeit rechte Kette für Geradeausfahrt

int turnspeed Geschwindigkeit einer Drehung, geht an beide Ketten, da die Drehung in Pulsen durchgeführt wird ist dieser Wert immer 100

int toleranz Der Toleranzwert für Drehungen. Standard: 10, also +-1.0 Grad Toleranz

struct rfidcommand Kommando AN den RFID Reader

unsigned int length Länge des übergebenen Kommandos, wird für verarbeitende Schleifendurchläufe gebraucht

unsigned int data[20] Das Kommando selbst, darf bis zu 20 byte breit werden

struct rfiddata Antwort VON dem RFID Reader, schon vorsortiert um die Weiterverarbeitung zu vereinfachen

unsigned int length Länge eines Kommandos

unsigned int command Steuerzeichen des Kommandos auf das geantwortet wird

unsigned int checksum Checksumme (abgeschnitten auf das letzte Byte)

unsigned int data[20] Der Inhalt einer Antwort, kann bis zu 20 byte breit werden

struct rfidtag Zusammengesetzter Datentyp mit allen für einen Transponder relevanten Informationen

long serial Seriennummer

int id ID Nummer, ist besser zu lesen als die Serial, evtl. für spätere Datenbankpflege interessant

int x Koordinate X

int y Koordinate Y

int status Bei einem korrekt gelesenen Transponder sollte der Status immer 0 sein. Ist er nicht 0 dürfen die Daten nicht verarbeitet werden

6.1.2 Motoransteuerung

robo-motor.h+c Motoransteuerung, selbsterklärend

void motor_stop(robotdefine * robot)

void motor_turnleft(robotdefine * robot)

void motor_turnright(robotdefine * robot)

void motor_turn(robotdefine * robot, int direction)

void motor_forward(robotdefine * robot)

6.1.3 I²C

robo-i2c.h+c angepasste I²C Funktionen

rfiddata myi2c_sendreceive(char* dev,unsigned char clnt,rfidcommand cmd)
Funktion um mit dem RFID-Reader zu kommunizieren. Dieser akzeptiert nur Befehle nach einem bestimmten Aufbau und liefert ähnlich formatierte Antworten. Für ein anderes I²C Device kann diese Funktion nur als Vorlage dienen

6.1.4 RFID

robo-rfid.h+c RFID Funktionen

long rfid_selecttag(robotdefine * robot) Select Kommando 0x83, liefert die Seriennummer eines Transponders zurück, ein select auf einen Transponder ist immer möglich, kein Auth notwendig

int rfid_authenticate(robotdefine * robot) Authenticate Kommando 0x85, Vorher muss ein select durchgeführt werden

rfiddata rfid_readblock1(robotdefine * robot) Liest Daten aus Block1 eines Transponders aus, Kommando 0x86 = READ BLOCK, Vorher muss ein select und auth durchgeführt werden

rfidtag rfid_gettag(robotdefine * robot) Liest alle relevanten Daten eines Transponders und gibt diese sortiert zurück. Hält die Prozedur select, auth, read ein.

int rfid_writeblock(robotdefine * robot, int blocknumber, rfidtag tagtowrite) Führt Kommando 0x89 aus. Beschreibt einen Transponder. Damit ein Transponder beschrieben werden kann muss die Reihenfolge select, auth, write eingehalten werden.

void rfid_printtag(rfidtag mytag) Hilfsfunktion um die Daten eines Transponders auf dem Bildschirm auszugeben

6.1.5 Kompass

robo-kompass.h+c Kompass Funktionen

int tograd(int gradtocheck) Überprüft ob ein Zahl im Wertebereich des Kompasses zwischen 0 und 3599 liegt. Wenn nicht wird der Wert entsprechend korrigiert.

int get_kompassgrad(robotdefine *robot) Liest den Kompass aus. Ein besser lesbarer Aufruf der I^2C Funktion.

int get_kompassgrad_gemittelt(robotdefine *robot, int mittel) Liest den Kompass aus und bildet einen Mittelwert über x Werte um die Genauigkeit der Messung zu erhöhen.

int kompasskalibrieren(robotdefine *robot) Sendet das Kommando zur Kalibrierung an den Kompass. Sendet 255 an Register 15, gleiche Wirkung wie Pin 6 mit Masse zu verbinden.

int get_abstand_grad(int grad1, int grad2) Funktion gibt den Abstand zwischen zwei Gradzahlen, in normierter Form, zurück. (So das anschließend eine Drehung um die ermittelt Gradzahl durchgeführt werden kann) Negatives Ergebnis bedeutet Grad2 ist durch eine Linksdrehung von Grad1 aus am schnellsten zu erreichen, ein positives Ergebnis bedeutet eine Rechtsdrehung bedeutet den kürzesten Abstand.

int winkelberechnen(int x1, int y1, int x2, int y2) Berechnet die absolute Richtung in der Punkt2 von Punkt1 liegt. Die Gradzahl wird mit Zehn multipliziert um im Wertebereich der Kompassdaten zu bleiben.

6.1.6 Fahren / Kombinierte Funktionen

robo-fahren.h+c Funktionen des Fahrbetriebs, kombiniert die bereits genannten Funktionstypen.

void fahren_drehenzu(robotdefine * robot, int grad_ziel, long lastserial) Dreht den Roboter anhand von Kompassdaten AUF eine bestimmte Gradzahl (absolute Drehung). Wird während der Drehung ein neuer Transponder gelesen wird die Drehung abgebrochen.

void fahren_drehenum(robotdefine * robot, int gradtoturn, long lastserial) Dreht den Roboter anhand von Kompassdaten UM eine bestimmte Gradzahl (relative Drehung). Wird während der Drehung ein neuer Transponder gelesen wird die Drehung abgebrochen.

void fahren_drivetoxy(robotdefine* robo,int ziel_x,int ziel_y,int referenzgrad)

Autonome Fahrt zu einer Zielkoordinate. Fährt geradeaus, scannt dabei nach Transpondern, wenn ein Transponder gescannt wird, wird getestet ob es der Zielkoordinate entspricht. Wenn nein, neu Berechnung des Winkels, Weiterfahrt.

6.1.7 Vorlagen

i2c.c+h und i2c-dev.h Diese Dateien sind aus einem Beispielprogramm kopiert und enthalten Definitionen um auf den I^2C Bus zuzugreifen. Unter anderem sind diese elementaren Funktionen enthalten:

int i2c_write_register(char*,unsigned char,unsigned char,unsigned short)

int i2c_read_register(char*, unsigned char, unsigned char)

void i2c_dump_register(char*,unsigned char,unsigned short,unsigned short)

void i2c_scan_bus(char*)

6.1.8 Hilfsprogramme

Weiterhin wurden Hilfsprogramme verfasst um diese Funktionen auch im Einzelnen aufrufen zu können. Die Programme sind im uClinux Image eingebunden und im Ordner *myprogs* abgelegt.

drehenum Relative Drehung, akzeptiert von der Kommandozeile übergebene Werte.

drehenzu Absolute Drehung, akzeptiert von der Kommandozeile übergebene Werte.

drivetoxy Fahrt zu Zielkoordinate, akzeptiert auch mehrere Koordinatenpaare, die sequentiell abgearbeitet werden.

kompasskalibrieren Sendet das Kommando zur Kalibrierung an den Kompass.

kompasstest Gibt den aktuellen Kompasswert aus.

probefahrt Fährt 9 fest definierte Koordinaten der Teststrecke ab.

probefahrtkurz Fährt 3 fest definierte Koordinaten der Teststrecke ab.

readtag Liest einen Transponder ein.

writetag Beschreibt einen Transponder, akzeptiert von der Kommandozeile übergebene Werte.

motorsteuerung Steuerprogramm für manuelle Fahrten.

6.2 Ansteuerung der Motoren

6.2.1 Linux Treibergundlagen

Die im Folgenden genannten Grundlagen sind an [plate07] Kapitel 3.4 "Treibergrundlagen" angelehnt.
Unter Linux übernimmt der Betriebssystemkern die Steuerung der Hardware. Dies geschieht unter Verwendung von Software-Komponenten, die jeweils für die Arbeit mit exakt einem der Geräte zuständig sind. Diese Software Komponenten werden als Kernel-Treiber (oder Device-Treiber, Gerätetreiber) bezeichnet. Sie sind eine Software-Ebene zwischen Anwendungen und der eigentlichen Hardware. Sie stellen einheitliche Hardware-unabhängige Zugriffsmöglichkeiten bereit, die die Details der Hardware vor den oberen Software-Schichten verbergen. Unter Linux stellt der Kern für die Zugriffe auf die Geräte dieselben Möglichkeiten bereit wie beim Zugriff auf Dateien. Zugriffe (beispielsweise open, read, write, close) erfolgen über eine spezielle virtuelle Datei im Dateisystem (Unterhalb von /dev/).

6.2.1.1 Kernel-Kontext, User-Kontext

Die CPU unterstützt verschiedene Modi in denen Prozesse auf Ressourcen zugreifen können. Mit diesen ist eine Zugriffskontrolle möglich. Die im Kernel-Kontext (die CPU befindet sich im Supervisor Mode) ablaufenden Prozesse haben vollen Zugriff auf alle Ressourcen. Das Gegenteil bildet der User-Kontext (der User-Mode der CPU). Hier werden die Zugriffe auf die Hardware und auf andere Ressourcen von der CPU kontrolliert und

gesteuert. Aus dieser Unterteilung wird klar, dass die Treiber stets im Kernel-Kontext laufen müssen, da sie direkten Zugriff auf die Hardware haben.

6.2.1.2 Anlegen eines Devices

Es gilt die Maxime "Alles ist eine Datei". Es gibt allerdings einen wesentlichen Unterschied zwischen Gerätedateien und "normalen" Dateien. Normale Dateien können einfach vom Benutzer angelegt werden (Per Editor, oder mit dem Kommando *touch*). Gerätedateien werden dagegen vom Superuser, meist unter Verwendung von *mknod* (make block or character special files), erstellt. Mknod wird der Name der zu erzeugenden Datei, ihr Typ (zeichenorientiert oder blockorientiert), sowie die Major- und die Minor-Nummer übergeben.

- char-Devices, zeichenorientierte Geräte wie beispielsweise Treiber für Tastatur, Maus, an der seriellen Schnittstelle angeschlossene Geräte
- block-Devices, blockorientierte Geräte können ein Dateisystem enthalten. Festplatten, CD-ROMs, USB-Sticks ...
- Major-Nummer steht für die Klasse eines Devices, beispielsweise ein Controller.
- Minor-Nummer ist die Nummer eines Gerätes, beispielsweise ein Terminal an einem Controller.

6.2.1.3 Statischer oder ladbarer Treiber

Ein Gerätetreiber kann fest in den Kernel eingebunden werden oder als ladbares Kernel-Modul umgesetzt werden. Aufgrund der höheren Flexibilität wurde der Motortreiber als ladbares Modul realisiert. So lässt sich ein Modul leichter Debuggen, da der Treiber schneller geladen und entladen werden kann, ohne den Kernel jeweils neu zu starten.

6.2.2 Erstellung des Treibers

6.2.2.1 Firmware als Quelle

In der mitgelieferten Firmware des Roboters ist bereits eine Motorsteuerung mitsamt PWM (Pulsweiten Modulierung) in C-Code realisiert. Diese arbeitet mit direkten Zugriffen auf Speicheradressen. Als Parameter benötigt die Steuerung jeweils den Geschwindigkeitswert für die linke und rechte Kette im Wertebereich von -100 bis +100, wobei der Zahlenwert für die Geschwindigkeit und das Vorzeichen für die Richtung steht. Der vorhandene Code wurde in das Linux Modell eines Treibers portiert.
Die eigentlichen Speicherzugriffe zur Steuerung der Hardware sollen im Kernel-Kontext ablaufen, die Parametrierung im User-Kontext.

6.2.2.2 Portierung in einen Linux Treiber

Jeder Treiber muss über eine Init und eine Exit Funktion verfügen. (Werden beim Laden oder Entladen des Moduls aufgerufen). Diese beiden Funktionen reichen aus um Code im Kernel-Kontext auszuführen. Um aber einen Datenaustausch zu Steuerprogrammen zu ermöglichen müssen weitere Funktionen hinzugefügt werden.
Auch hier gibt es, je nach Einsatzgebiet, verschiedene Ansätze und Möglichkeiten. Faktoren zur Auswahl der entsprechenden Funktionen (open, read, write, lock... - um einige Beispiele zu nennen) können sein:

- Müssen mehrere Prozesse gleichzeitig auf das Device zugreifen?
- Müssen Daten zurückgelesen werden?
- Berechtigungen (Lesend, Schreibend)

Im Fall der Motorsteuerung soll jeweils nur ein Prozess auf das Device zugreifen und es sollen keine Daten zurückgelesen werden.
Auf dieser Grundlage wurde die Funktion ioctl (input output control) in abgewandelter Form implementiert.

6.2.2.3 Die Treiberschnittstelle ioctl()

IO-Controls stellen eine universelle Schnittstelle zum Treiber dar. Der Standard-Aufruf der Anwendungsschnittstelle ist wie folgt definiert:
*int ioctl(struct inode *geraetedatei, struct file *instanz, unsigned int cmd, unsigned long args)*
Genutzt wird die Tatsache, dass zwei Variablen an die Funktion übergeben werden (cmd und args). Normalerweise wird *cmd* in einer Switch-Case Anweisung ausgewertet um so mehrere unterschiedliche Befehle interpretieren zu können. In *args* finden sich die jeweiligen Argumente zum Kommando.

Der Aufruf wird dahingehend abgewandelt, dass in cmd und args die beiden Geschwindigkeitswerte übergeben und, nach einem Typecast, an die entsprechend portierten Funktionen aus der Firmware übergeben werden.

6.2.2.4 Laden des Treibers

Geladen wird das Modul mit zwei Befehlen:
mknod /dev/motor c 241 0
Das Character-Device wird im Dateisystem unter /dev/motor angelegt. Major-Number: 241, Minor-Number: 0
insmod /myprogs/motor.ko
das Treibermodul wird geladen.
Um den Treiber beim Start des Roboters automatisch zu laden wurden obige Aufrufe der Datei */etc/rc* hinzugefügt, die bei jedem Systemstart aufgerufen wird.

6.2.2.5 Entladen des Treibers

Um den Treiber wieder zu entladen dient der Befehl:
rmmod motor

6.2.3 Benutzung aus dem User-Kontext

In einem normalen User Programm kann nun mit den folgenden Kommandos im Quelltext mit den Motoren gearbeitet werden.
int fd = open("/dev/motor", O_RDWR);
Mit dem Open File Befehl wird die Datei / das Device "/dev/motor" im Read/Write Modus geöffnet.
Anschließend können mit dem Aufruf
ioctl(fd, speedleft, speedright);
die Geschwindigkeitswerte für die Motoren übergeben werden. So ist es möglich mit normalen Berechtigungen die Motoren zu steuern. Es existiert eine klare Trennung zwischen den Hardwarezugriffen sowie der Parametrierung.

6.3 Motor Funktionen

Es wurden Funktionen für die wesentlichen Fahrbefehle erstellt. Diese stellen die unter 6.2.3 "Benutzung aus dem User-Kontext" genannten ioctl() Aufrufe unter aussagekräftigen Funktionsnamen zur Verfügung.

6.4 I^{2}C Funktionen

In den uClinux Quellen findet man Beispiele wie man Daten mit Geräten am I^{2}C Bus austauschen kann. Im Programm *ppifcd-test* (zu finden in den uClinux Quellen unter */user/blkfin-test/ppifcd-test*), einem Testprogramm für einen Kamerasensor, finden sich in den Dateien *i2c-dev.h* und *i2c.c* Definitionen und elementare Funktionen.
Mit diesen Funktionen konnte bereits der Kompass ausgelesen werden. Um ein Register auszulesen wird dessen Nummer an die Adresse des Busteilnehmers gesendet und dieser sendet als Antwort den Inhalt des gewünschten Registers. Die Kommunikation mit dem RFID-Reader gestaltet sich dagegen anders. Die Besonderheiten werden im entsprechenden Abschnitt beschrieben.

6.5 Kompass Funktionen

Der Kompass liefert Werte im Bereich von 0 bis 3599, Gradzahlen mit einer Nachkommastelle Genauigkeit die, wie in Abbildung 6.1 gezeigt, zu interpretieren sind. Die Kompassdaten werden genutzt um den Fortschritt von Drehungen des Fahrzeugs zu bewerten. Die bei der autonomen Fahrt verwendete Funktion *winkelberechnen()* bestimmt die Richtung zwischen zwei Koordinaten. Dies soll an einem Beispiel erläutert werden. Punkt A habe die Koordinaten (9/9), Punkt B habe (3/3). Erstellt man nun ein rechtwinkliges Dreieck mit A und B als Endpunkten der Hypotenuse, so ergibt sich der Winkel zwischen Ankathete und Hypotenuse zu 45 Grad. Die Funktion betrachtet nun zusätzlich die Lage von A und B zueinander. Da Punkt B süd-westlich von A liegt berechnet die Funktion die Richtung zu 225 Grad (der errechnet Winkele von 45 Grad + 180 Grad für den süd-westlichen Sektor). Stünde also das Fahrzeug auf Punkt A, so müsste es sich in Richtung von 225 Grad ausrichten um auf Punkt B zu zeigen.

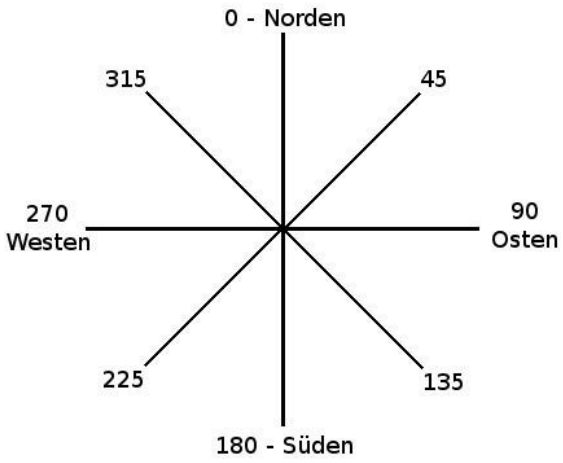


Abbildung 6.1: Navigationsbereiche

6.6 RFID Funktionen

Um mit dem RFID Reader zu Kommunizieren müssen gewisse Konventionen eingehalten werden.
Ein Befehl besteht hier aus den Teilen:
- Gesamtlänge der gesendeten Daten, ohne Checksumme, 1 Byte
- Befehl, 1 Byte
- Data, n Byte
- Checksumme, 1 Byte (abgeschnitten)

Die Antwort erfolgt in ähnlicher Konvention:
- Gesamtlänge der Antwort, ohne Checksumme, 1 Byte
- Befehl auf den geantwortet wird, 1 Byte
- Antwort, n Byte
- Checksumme, abgeschnitten auf den letzten Byte, 1 Byte

Bevor allerdings eine Antwort gelesen werden kann muss eine Wartezeit im Bereich von 15 ms eingehalten werden (die sogenannte *15ms Rule*). Weiterhin werden Antworten nur für die Dauer von 50ms bereitgestellt. Werden die Daten nicht in diesem Zeitfenster ausgelesen sind sie verloren.
In der vorliegenden Implementation wird der RFID Reader im Polling-Verfahren abgefragt. Die 15ms Rule wird durch den Einsatz einer sleep Funktion umgesetzt.

Problematisch war die variable Länge von Befehlen und Antworten, die von 3 bis 20 Byte reichen. Um mit diesem Problem umgehen zu können wurden entsprechende Datentypen definiert (*rfidcommand* - Kommando, *rfiddata* - Vom Reader gelesene Antwort).
Die Funktion *myi2c_sendreceive()* verarbeitet diese variablen Datentypen und hält auch die 15ms Rule ein.
Im folgenden wird der Ablauf der Lese und Schreibvorgänge der Transponder erläutert. Dieser Funktionsumfang erlaubt die Benutzung der elementaren Funktionen des RFID-Moduls über den I^2C Bus. Somit ist man von der Windows Software und der seriellen Schnittstelle unabhängig. Diese wird nur noch zu Wartungszwecken (Einspielen neuer

Firmware) oder zu Testzwecken benötigt.

6.6.1 Transponder Lesen

Um den Inhalt eines Transponders auszulesen ist folgende Reihenfolge einzuhalten:
select - Ein Transponder befindet sich in Reichweite, die Seriennummer wurde gelesen. Die Seriennummer kann immer gelesen werden, diese ist nicht verschlüsselt.
auth - Authentifizierung, man macht bekannt welcher Block des Transponders gelesen werden soll und stellt den Schlüssel zur Verfügung, vorher muss ein select erfolgreich durchgeführt worden sein.
read - Erst jetzt kann der Inhalt des zuvor authentifizierten Blocks gelesen werden. Es werden immer die gesamten 16 Byte des gewünschten Blocks ausgelesen.
Die Funktion *rfid_gettag()* hält diese Reihenfolge ein und liefert die Informationen des gelesenen Transponders formatiert im Datentyp *rfidtag* zurück. Zur Ver- und Entschlüsselung wird der unveränderte standard Schlüssel des Readers verwendet.
Jeder Select Vorgang, der prüft ob sich ein Transponder in Reichweite befindet, dauert wenigstens 15ms (Absenden des select Kommandos, 15ms Warten, Antwort abholen). Befindet sich ein Transponder in Reichweite erhöht sich die Wartezeit auf wenigstens 45ms (select, auth, read jeweils 15ms).

6.6.2 Transponder Beschreiben

Um einen Transponder zu Beschreiben ist die schon unter 6.6.1 “Transponder Lesen” beschriebene Reihenfolge einzuhalten, nur das ein *write* auf *select* und *auth* folgt. Der zu diesem Zweck verfassten Funktion *rfid_writeblock()* wird die gewünschte Blocknummer sowie die zu schreibenden Informationen in Form des Datentyps *rfidtag* übergeben. Die Speichernutzung erfolgt wie unter 4.2.5.2 “Speichereinteilung” genannt.

6.7 Richtungsänderungen

Um eine Richtungsänderung durchzuführen werden die Funktionen von Kompass, Motor und RFID kombiniert eingesetzt.

Der erste Ansatz eine Drehung durchzuführen war so gestaltet, dass das Fahrzeug in eine gleichförmige Drehbewegung durch gegenläufige Kettenbewegung versetzt wird, wie sie für ein Kettenfahrzeug typisch ist. Der Fortschritt der Drehung wird dann anhand von Kompassdaten, in Form eines Vergleichs von Startwert, Sollwert und Istwert, interpretiert. Sobald die gewünschte Gradzahl erreicht ist werden die Motoren abgeschaltet. Durch die Trägheit der Motoren dreht das Fahrzeug aber noch 10 bis 30 Grad nach. Verschiedene Drehungen im Uhrzeigersinn, gegen den Uhrzeigersinn oder um unterschiedliche Gradzahlen erzeugen verschiedene Störungen. So war es nicht möglich diese Totzeit zu bestimmen und herauszurechnen. Es zeigte sich allerdings, dass der Fehler mit Herabsenken der Drehgeschwindigkeit abnahm. Allerdings führte selbst die langsamste mögliche Drehgeschwindigkeit noch zu hohen Abweichungen im besagten Bereich. Weiterhin ist eine Störung des Kompasses durch das von den Motoren erzeugte Magnetfeld wahrscheinlich (allerdings nicht praktikabel messbar).
Es wurde eine alternative Steuerung der Drehung erarbeitet. Die Drehung wird nun in Form von kurzen Pulsen durchgeführt. Die Motoren werden mit voller Kraft gegenläufig für 10ms angeschaltet und wieder gestoppt. Nach diesem Puls wird 100 ms lang gewartet. Erst dann wird die Gradmessung durchgeführt. Der Roboter befindet sich zur Gradmessung idealerweise im Stillstand und der Motor wird als Störquelle minimiert. (Weiterhin wird an diesem Punkt auch der RFID-Reader abgefragt. Sollte während der Drehung ein neuer Transponder in der Reichweite der Antenne auftauchen wird das Fahrzeug ein kleines Stück zurückgesetzt und die Drehung abgebrochen, damit auf den neu gelesenen Transponder eingegangen werden kann.)
Softwareintern kann so mit einer Toleranz von 2.0 Grad (+- 1.0 Grad) gearbeitet werden. Die Toleranz weiter zu verringern ist nicht sinnvoll, da die Feinheit der Bewegungsimpulse nicht mehr ausreicht um diese einzuhalten.

6.8 Autonomes Fahren

Der autonomen Fahrt liegt das folgende Gedankenmodell zu Grunde: Das Fahrzeug bewegt sich in einem zweidimensionalen kartesischen Koordinatensystem und versucht eine gegebene Zielkoordinate zu erreichen. Die Koordinaten des Koordinatensystems sind auf entsprechend positionierten RFID-Transpondern gespeichert. Bei der Fahrt auf der Strecke werden die gelesenen Koordinaten mit den Zielkoordinaten verglichen. Anhand ihrer Lage zueinander richtet sich das Fahrzeug in Richtung des Ziels aus und versucht dieses auf direkten Wege anzufahren. Die Autonomie entsteht also durch das kontinuierliche Bewerten der eigenen Position in Bezug auf das Ziel und dem entsprechenden Versuch dieses zu erreichen.

Die Funktion *fahren_ drivetoxy()* setzt dies wie folgt um: Der Roboter fragt während der Geradeausfahrt permanent ab, ob sich ein RFID Transponder in Reichweite befindet. Ist dies der Fall so werden die auf dem Transponder gespeicherten Koordinaten mit den Zielkoordinaten verglichen. Entsprechen die gelesenen Koordinaten den Zielkoordinaten so ist die Fahrt beendet. Ist dies nicht der Fall so wird anhand geometrischer Funktionen die Richtung bestimmt in der sich das Ziel befindet. Das Fahrzeug richtet sich in die entsprechende Richtung aus und setzt seine Fahrt in gerader Richtung fort. Diese Prozedur wird bei jedem gelesenen Transponder wiederholt.

Um die Richtung zu bestimmen wird die unter 6.5 “Kompass Funktionen” erklärte Funktion *winkelberechnen()* genutzt. Diese bestimmt, entsprechend der nach Abbildung 6.1 zu interpretierenden Wertebereiche, in welcher Richtung Koordinate 2 (soll) von Koordinate 1 (ist) liegt.

Damit diese Werte auch nutzbar sind muss davon ausgegangen werden können, dass die y-Achse in Richtung Norden (0 Grad) zeigt. Da aber eine solche Ausrichtung der Teststrecke aufwendig und wenig praxisnah ist, wird die Ausrichtung des Koordinatensystem vom Roboter bestimmt.

Dafür muss er sich zum Programmstart in einer definierten Position zur Teststrecke befinden. Es wurde sich für die Ausrichtung parallel zur x-Achse mit der Vorderseite (Die Seite mit der Kamera) nach rechts zeigend entschieden. Mit dem Kontextwissen

über die Stellung des Fahrzeugs auf der Teststrecke lässt sich nun ein Offset bestimmen, der zu allen berechneten Richtungen hinzuaddiert wird.
So ist gewährleistet, dass sich das Fahrzeug bei 0 Grad parallel zur y-Achse mit der Vorderseite nach oben zeigend ausrichtet und die Richtungsberechnung sinnvoll genutzt werden kann.

6.9 Kopplung mit Steuerungssoftware von Dirk Mödrath

Die in [moedrath09] erstellte Steuerungssoftware ist in die Elemente Bedienstation, Server und Roboter unterteilt. Sie unterscheidet zwischen den Betriebsmodi manuelle Fahrt und autonome Fahrt.
Die manuelle Fahrt ist durch Aufrufe der unter *robo-motor.c+h* bereitgestellten Funktionen möglich. Im Roboter Teil der Steuerungssoftware sind diese unter *roboter_main.c* ab Zeile 793 (Mit dem Inhalt: *else if(strcmp(lesen.art,"Fahranweisung im Manuellbetrieb") == 0){*) einzufügen. Hier ist zu beachten, dass die dort verwendete Variable *Weite* im jetzigen Ausbauzustand des Fahrzeugs nur mit entsprechenden Wartezeiten realisiert werden kann.
Damit autonome Fahrten durchgeführt werden können, müssen noch Vorarbeiten geleistet werden. Dazu gehören die Pflege der RFID-Transponder-Datenbank und eine Routerberechnung. Eine Route ist in Form einer verketteten Liste vom Datentyp *Tag_Reihen_Liste* organisiert. Diese enthält Elemente des Typs *RFID_Routendaten*, welche wiederum die eigentlichen Transponder Daten im Datentyp *RFID_DATEN* enthalten.

In den Programmen *probefahrt* und *probefahrtkurz* wird gezeigt wie die Verarbeitung des kleinsten Elemts einer Liste geschehen kann. Sie enthalten die Funktion *fahren_drivetoRFIDDATEN()*, eine Abwandlung von *fahren_drivetoxy()*, welche den von Dirk Mödrath entworfenen Datentyp *RFID_DATEN* verarbeiten kann.

7 Vorstellung der Teststrecke

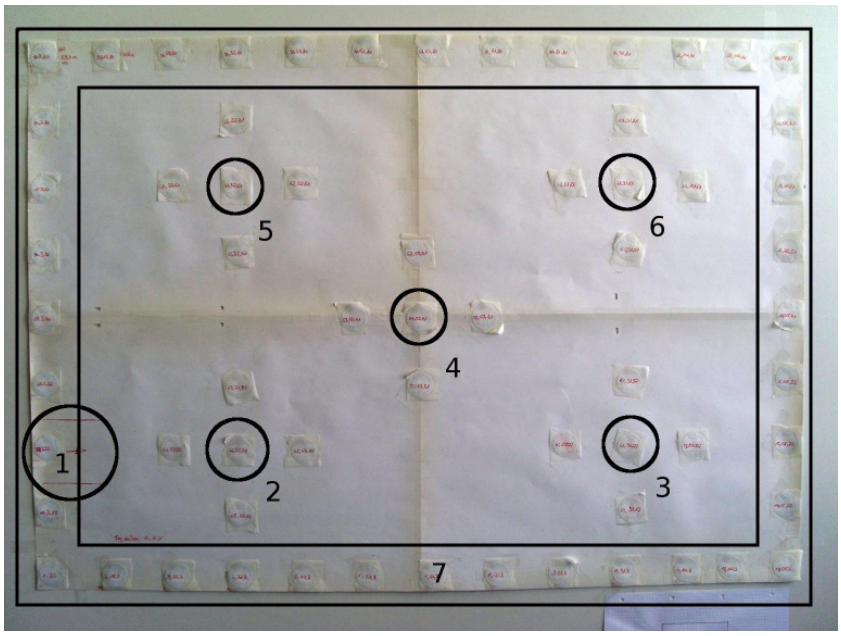


Abbildung 7.1: Teststrecke

Abbildung 7.1 zeigt die Teststrecke. Sie umfasst eine Größe von 1.20m x 0.84m.

Die Transponder sind in einem zweidimensionalen kartesischen Koordinatensystem angeordnet. Die Koordinaten werden in ganzen Zentimetern angegeben. Der Koordinatenursprung (0 / 0) befindet sich in der unteren linke Ecke. Jeder Transponder enthält die Attribute Id, Koordinate-X und Koordinate-Y.

Transponder Abstand Die Transponder werden im Abstand von 6 cm positioniert. Dieser Abstand wurde wie im folgenden beschrieben ermittelt:

Der Roboter wurde mit einer Software ausgestattet, die die Fahrt nach Lesen eines Transponders abbricht. Der Abstand der Transponder wurde stetig vergrößert. Nach jedem Lesen wurde das Fahrzeug wieder an die Ausgangsposition zurückbewegt. Dieser Vorgang wurde solange wiederholt bis das Fahrzeug zwischen den Transpondern hindurchfuhr ohne dabei einen der Transponder zu lesen. Dies geschah bei ca. 7 cm Abstand. Dieser Wert wurde zur Sicherheit um einen cm verringert, so dass sich 6 cm als geeigneter Abstand ergab. Abbildung 7.2 zeigt diesen Aufbau.

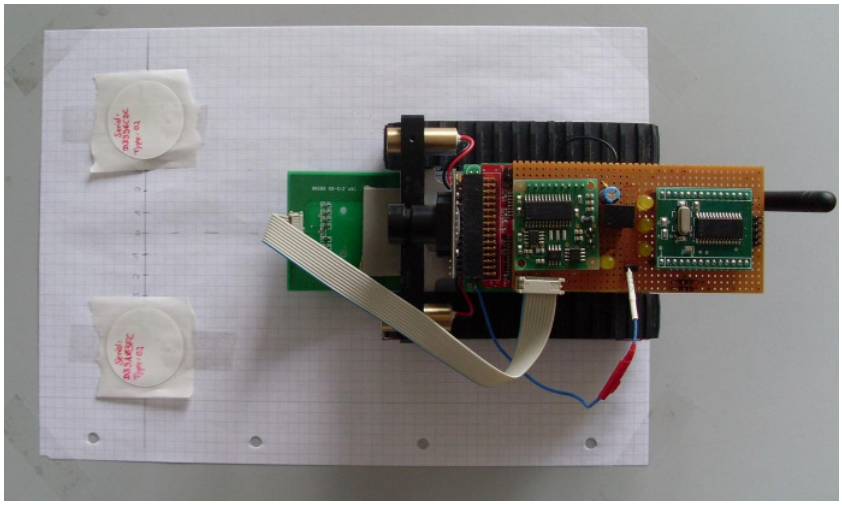


Abbildung 7.2: Bestimmung des maximalen Abstands zwischen zwei Transpondern

Äußere Umrandung (7) Die Strecke wird von 40 Transponder nach Außen abgegrenzt. Sie sollen sicherstellen, dass der Roboter die Teststrecke nicht verlassen kann ohne einen Transponder zu lesen.

Zielpunkte (2-6) Insgesamt sind 5 Zielpunkte vorhanden. Jeder Ziel-Transponder wird von 4 Transpondern umgeben. Diese vergrößern dessen Einzugsbereich. Für den Fall, dass das Fahrzeug am Ziel vorbeifährt, soll so verhindert werden, dass der Fehler erst am Rand der Strecke bemerkt wird. Generell ist aber jede auf einem Transponder gespeicherte Koordinate als Ziel möglich.

Kalibrierstelle (1) Weiterhin gibt es eine Kalibrierstelle, an der sich das Fahrzeug zum

Programmstart befinden muss. (Dieser Punkt ist willkürlich gewählt, der Roboter muss nur exakt parallel zur x-Achse, mit der Kamera nach rechts zeigend, ausgerichtet sein) Dass das Fahrzeug nach dem Start direkt einen Transponder liest ist für die Kalibrierung nicht von Belang.

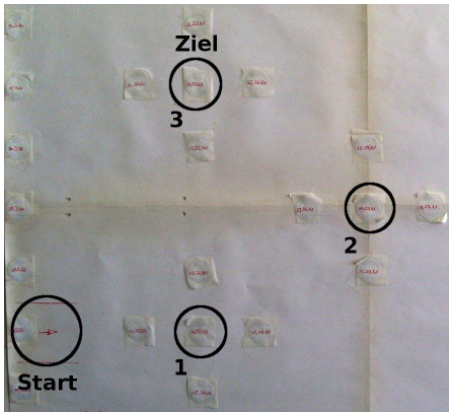
Hier wird die Ausrichtung der Teststrecke festgestellt. Diese Messung ist nötig, da die Richtungswechsel gestützt auf Kompassdaten durchgeführt werden können. (Siehe 6.8 “Autonomes Fahren”)

8 Auswertung mit Ausblick

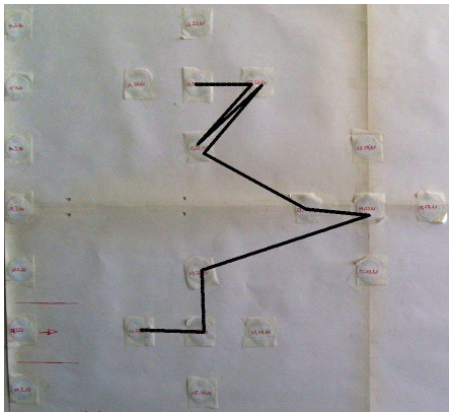
In diesem Kapitel sollen die einzelnen Teile dieser Arbeit kritisch betrachtet werden. Die Abschnitte enthalten teilweise einen Ausblick um diese Arbeit in einen größeren Kontext einzubetten.

8.1 Autonomes Fahren

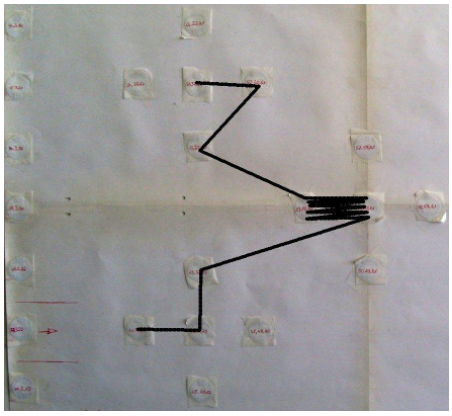
Die autonomen Fahrten auf der Teststrecke verlaufen erfolgreich. Das Fahrzeug findet gegebene Ziele verlässlich. Die Abbildung 8.1 zeigt exemplarisch die gelesenen Transponder bei der Ausführung des Programms *probefahrtkurz*. In *probefahrtkurz* werden drei Transponder mit den Koordinaten (32,22), (59,41) und (32,61) nacheinander angefahren. (In der Abbildung als 1, 2 und 3 gekennzeichnet)
Die Abbildung zeigt nicht den tatsächlich gefahrenen Weg, sondern die auf der Fahrt gelesenen Transponder. Hieran kann man erkennen, dass keine Aussage über die Position des Fahrzeugs getroffen werden kann. Klar ist nur, welcher Transponder als letztes gelesen wurde. Aus welchem Winkel ein Transponder angefahren wurde lässt sich nur anhand des davor gelesenen Transponders vermuten. Eine Angabe über den Abstand des Fahrzeugs zum Transponder ist ebenfalls nicht möglich.
Weiterhin ist gut sichtbar wie sich die Transponder teilweise überlappen und der Reader abwechselnd nebeneinander liegende Transponder liest ("Zickzack"Muster). Die kompletten Logfiles von 5 Testfahrten befinden sich im Anhang unter 12.4.
Der gewählte Ansatz ein Ziel zu erreichen, sowie der Aufbau der Teststrecke, ermöglichen es, dass das Fahrzeug auch mit Störungen umgehen kann. So ist es möglich den Roboter während der Fahrt auf eine beliebige andere Stelle der Strecke zu stellen. Sobald der



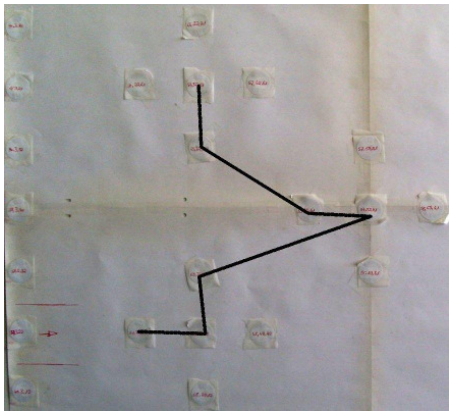
(a) Ablauf



(b) Fahrt 1



(c) Fahrt 2



(d) Fahrt 3

Abbildung 8.1: Exemplarische autonome Fahrten

nächste Transponder gelesen wird, wird sich wieder in Richtung des Ziels ausgerichtet und die Fahrt in die korrekte Richtung fortgesetzt.
In den folgenden Abschnitten werden nun wichtige Bestandteile der autonomen Fahrt betrachtet.

8.1.1 Geradeausfahrt

Im Rahmen dieser Arbeit findet keine Kontrolle der Geradeausfahrt statt. Auf der verwendeten Teststrecke fällt dies allerdings nicht stark ins Gewicht, da keine langen Strecken zurückgelegt werden und die Richtung an jedem gelesenen Transponder neu berechnet wird.
Warum der Geradeauslauf korrigiert werden muss und warum dies in dieser Arbeit nicht möglich war wird im Folgenden erklärt:
Im Ausgangszustand des Roboters ist bei der Geradeausfahrt schon nach wenigen Metern ein Versatz von der Ideallinie zu erkennen. (Beide Ketten werden mit den gleichen Werten angesteuert). Dieses Verhalten ist in der Kettenkonstruktion begründet. Die beiden Gummi-Ketten sind unterschiedlich stark gespannt und bieten keine Möglichkeit dies zu korrigieren. Weiterhin ist ein Schlupf der Ketten auf deren Aufhängung zu erkennen. Ebenfalls ist vorstellbar, dass die Motoren nicht gleich stark sind und dadurch über ein unterschiedliches Ansprechverhalten verfügen. Der Faktor der Kettenspannung fällt allerdings stärker ins Gewicht.
In dieser Arbeit sollte nun die Möglichkeit betrachtet werden die Geradeausfahrt anhand von Kompassdaten zu bewerten und die Fahrt entsprechend zu korrigieren.
Da sich der Kompass am Erdmagnetfeld orientiert ist er anfällig für Störungen durch andere Magnetfelder oder Metall.
Dies beginnt bei Verfälschungen um wenige Grad (Metallteile von Möbeln) und kann beliebige Werte annehmen (Träger in Böden und Wänden / Magnetfelder von Netzteilen). Diese Anfälligkeit ist keine Besonderheit die nur auf das verwendete Kompassmodul zutrifft. Ein zum Vergleich herangezogener Kartenkompass ließ sich auf die gleiche Weise stören.

Eine Bewertung der Geradeausfahrt des Fahrzeuges nur Anhand von Veränderungen der Kompassdaten ist deshalb nicht praktikabel. In einem Gebäude sind dafür zu viele Störeinflüsse vorhanden.
Um den Effekt der ungleichmäßig gespannten Ketten zu kompensieren wurde sich mit der Anpassung der Geschwindigkeitswerte beholfen, so dass die schwächer gespannte Kette schneller angetrieben wird. Dies ist aber keine allgemeingültige Lösung, da sich das Verhalten mit sinkender Batteriespannung verändert und stark vom Untergrund abhängig ist.

Als Ausblick für folgende Arbeiten seien hier einige Möglichkeiten genannt den Geradeauslauf zu überprüfen:
Mit der vorhandenen Kamera kann man das Fahrzeug um die Fähigkeit der Linienverfolgung erweitern. So kann das Fahrzeug entsprechend am Boden platzierte Linien zur Orientierung nutzen. Die Überbrückung von langen Gängen kann so gewährleistet werden.
Auch kann man mit entsprechender Bildverarbeitung Kanten im Sichtbereich der Kamera nutzbar machen. Durch die Erkennung des Übergangs zwischen Boden und Wand kann die Breite eines Ganges bestimmt werden und entsprechend mittig darin gefahren werden.
Die Möglichkeiten von Erweiterungen der Hardware des Roboters sind vielfältig. Über Radimpulsgeber kann sichergestellt werden, dass sich beide Ketten gleichschnell drehen. In Kombination mit gleich stark gespannten Ketten ist der Geradeauslauf bereits verlässlicher, auch wenn so zum Beispiel Untergrundwechsel und die daraus resultierenden Abweichungen nicht bemerkt werden können. Auch ist dann eine Aussage über die zurückgelegte Strecke möglich.
Im Internet finden sich auch Ideen zum Einsatz eher unüblicher Sensoren. So wurde ein Bristlebot mit dem optischen Sensor einer Maus ausgestattet. Durch die Interpretation der Werte für die X-Achse wird das Fahrzeug gelenkt. (siehe [www-bstlctrl]).

8.1.2 Richtungsänderungen

Zur Kontrolle der Geradeausfahrt lies sich der Kompass nicht nutzen. Inwieweit er für die Durchführung von Richtungswechseln tauglich ist beschreibt dieser Abschnitt.
Soll sich das Fahrzeug anhand von Kompassdaten absolut in eine bestimmte Richtung ausrichten, so müssen die unter 8.1.1 beschriebenen Störeinflüsse ausgeschlossen werden. Im Rahmen der auf der Teststrecke durchgeführten autonomen Fahrten ist dies möglich. Soll das Fahrzeug aber in einem größeren Maßstab eingesetzt werden, so können Störeinflüsse nicht mehr ausgeschlossen werden. Richtungswechsel in eine absolute Richtung sind dann nicht mehr vertrauenswürdig, da eine mögliche Störung nicht festgestellt werden kann.
Eine relative Drehung des Fahrzeuges anhand der Kompassdaten ist dagegen, trotz der angesprochenen Störanfälligkeit, gut möglich. Da sich das Fahrzeug auf der Stelle um seine eigene Achse dreht, bleiben für die Dauer der Drehung die Störeinflüsse gleich und fallen nicht ins Gewicht. (Auch wenn eine vom Modul gemessene Himmelsrichtung nicht der tatsächlichen entspricht, so bedeutet eine Drehung um 180 Grad dennoch eine Drehung des Fahrzeugs in die entgegengesetzte Richtung)
Wie genau eine Richtungsänderung durchgeführt werden kann hängt von mehreren Faktoren ab. Als erstes sei hier die Genauigkeit des Kompass Moduls genannt, die laut Datenblatt 3.0 bis 4.0 Grad beträgt. Zu dieser Ungenauigkeit addiert sich die softwareinterne Toleranz von 2.0 Grad. Ebenfalls könnten die von den Motoren erzeugten Magnetfelder einen Einfluss auf den Kompass haben. (Auch wenn bei der Drehung nach einem Puls die Motoren 100 ms abgeschaltet sind, bevor die Messung durchgeführt wird, so ist es denkbar, dass doch noch Störeinflüsse vorhanden sind.)
Um eine Aussage über die Genauigkeit einer relativen Drehung treffen zu können wurde eine Messreihe angefertigt. Gemessen wurde die Abweichung des Fahrzeugs nach einer Drehung um 180 Grad (der größten im Programmablauf möglichen Drehung) als Winkel zur X-Achse, sowie die Verschiebung in X- und Y-Richtung. Abbildung 8.2 zeigt die Schablone auf der die Messung durchgeführt wurde, Abbildung 8.3 zeigt die Ausgangsposition des Fahrzeugs, sowie eine beispielhafte Endposition nach einer Drehung.

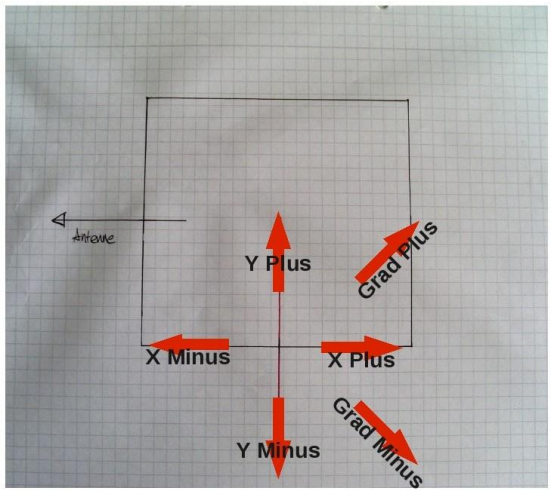
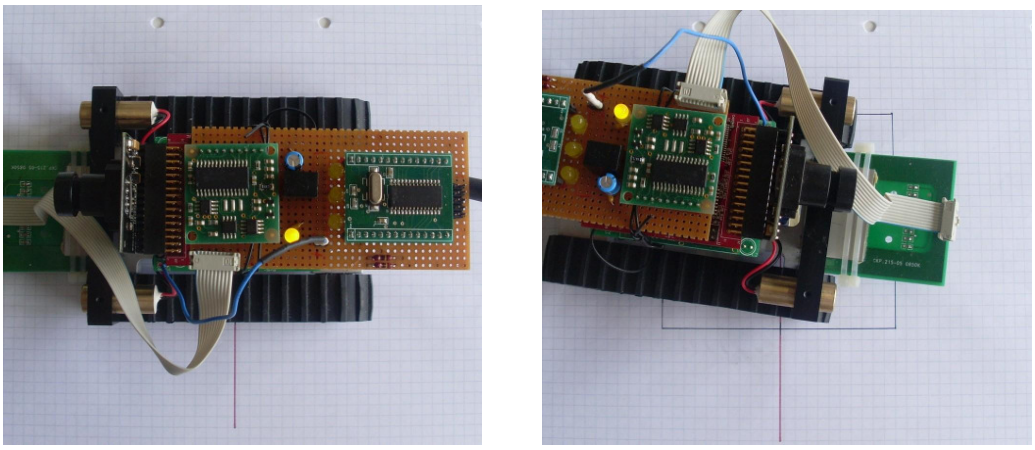


Abbildung 8.2: Messung Richtungsänderung Aufbau



(a) Ausgangsposition (b) Endposition Beispiel

Abbildung 8.3: Messung Richtungsänderung Beispiel

Die folgende Tabelle zeigt die aufgenommenen Messwerte, sowie die sich daraus ergebenden Mittelwerte:

Messung	Abweichung X (cm)	Abweichung Y (cm)	Winkel (Grad)
1	-4	+1,25	-8
2	-3	+1,5	-8
3	-3	+1,25	-9
4	-3	+1,25	-7
5	-3	+1,0	-9
6	-3,5	+1,25	-8
7	-3,5	+2,0	-9
8	-3,5	+1,75	-8
9	-3,0	+1,25	-6
10	-3,0	+1,25	-6
Mittelwert:	**-3,25**	**+1,375**	**-7,8**

Die Abweichung in x und y Richtung ist durch die bereits angesprochenen Schwächen des Antriebs zu erklären, weshalb sich das Fahrzeug nicht exakt um seine eigene Achse dreht.

Die Abweichung des Winkels zur X-Achse von -7,8 Grad im Mittel ist überraschend. Da die Drehung jeweils im Uhrzeigersinn durchgeführt wurde, war die Vermutung, dass die Trägheit der Motoren für das konstante Übersteuern der Drehung verantwortlich ist. Einige Gegentests mit Drehungen gegen den Uhrzeigersinn zeigten allerdings sehr ähnliche Ergebnisse.

Betrachtet man die Differenz zwischen höchster und niedrigster Abweichung so beträgt diese nur 3 Grad, was, in Anbetracht der genannten Einflüsse, einen guten Wert darstellt. Die Grundabweichung von wenigstens 6 Grad (absolut) könnte in der Kalibrierung des Kompasses begründet sein. Bevor also die Arbeit mit dem Kompass fortgesetzt wird sollte eine erneute Kalibrierung mit anschließender Messung durchgeführt werden.

Einen verlässlichen Geradeauslauf vorausgesetzt, wäre eine absolute Abweichung von 7,8

Grad nach einer Richtungsänderung tragbar. Eine Abweichung von 7,8 Grad bedeutet auf eine Strecke von 10 Metern eine Abweichung von 1,36m. In einem Szenario, in dem das Fahrzeug durch entsprechend breite Flure fährt, wäre, befänden sich alle 10 Meter Transponderreihen zur Orientierung, diese Abweichung akzeptabel. Nach erreichen der Transponderreihen könnte sich das Fahrzeug anhand dieser wieder mittig im Gang ausrichten und seine Fahrt fortsetzen.

8.1.3 RFID-Modul, RFID-Transponder

Das Vorhandensein eines Transponders in Reichweite der Antenne wird durch Polling des RFID-Readers überprüft. Trotz der Verzögerungen beim Lesen (ein select dauert aufgrund der 15ms Rule wenigstens 15ms; ein kompletter Read wenigstens 45ms) funktioniert dies auf der Teststrecke verlässlich. Es kommt nicht dazu, dass ein Transponder überfahren wird ohne dabei gelesen zu werden.
Es kommt allerdings vor, dass sich das Fahrzeug vor Abschluss des Lesevorgangs zu weit vom Transponder entfernt hat, so dass der dreigeteilte Lesevorgang nicht bis zu Ende durchgeführt werden kann. Dies wird erkannt und durch Zurücksetzen des Fahrzeugs ein Neueinlesen des Transponders eingeleitet.
Erhöht man allerdings die Geschwindigkeit des Fahrzeugs, so nimmt die Fehlerrate zu und es kann dazu kommen, dass ein Transponder überfahren wird, ohne dabei gelesen zu werden.
Hier ist ein Interrupt wünschenswert. Das RFID-Modul verfügt mit *Pin21 OUTPUT1 / DREADY* über eine Interrupt Leitung. Diese ist zur Zeit an einer LED angeschlossen. Nach dem jetzigen Stand gibt es allerdings keine Möglichkeit die Interrupt-Leitung an das Blackfin Board anzuschließen. Dieses verfügt nur über general purpose IO Anschlüsse. Hier ist zu prüfen ob diese als Interrupt genutzt werden können. Das Modul könnte dann im Autoseek Modus betrieben werden und nach Scannen eines Transponders einen Interrupt beim Betriebssystem auslösen. Dies würde das Lesen der Transponder verlässlicher machen und höhere Fahrgeschwindigkeiten erlauben.
Sollte sich allerdings die Anzahl der benötigten Transponder in folgenden Projekten deut-

lich steigern, so ist der Preis der Mifare Transponder zu beachten. Deren Preis stellt mit 1.35 Euro pro Stück (im vorliegenden Fall) einen nicht unerheblichen Faktor dar. Andere Transponder, die ebenfalls im 13.56MHz Bereich arbeiten, aber einen unverschlüsselten ISO Standard verwenden, haben dagegen einen Stückpreis von 0.25 Euro. Hier muss ebenfalls auf einen anderen Reader umgestiegen werden, da der SM130 exklusiv zu Transpondern mit Mifare Code kompatibel ist. (Alternativen zum verwendeten RFID-Modul werden unter 4.2.2 “Auswahl” angeführt)

8.2 Debugging / WLAN

Die WLAN Verbindung birgt ein gewisses Fehlerpotential. Bei Telnet Verbindungen oder Dateiübertragungen kommt es in unregelmäßigen Abständen zu Verbindungsabbrüchen obwohl sich das Fahrzeug in wenigen Metern Nähe zum Access-Point befindet.
Eine Telnet Verbindung kann nach einem Verbindungsabbruch fortgesetzt werden, diese Probleme in der Kommunikation machen dagegen die getestete debugging Methode unpraktikabel. Fehler in der Datenübertragung führen zu halbminütigen Verzögerungen die die debugging Session mit einem Checksum Fehler abbrechen lassen. Die Möglichkeit der Kombination von GDBserver (Blackfin) und GDB (Entwicklungs-PC) wäre sonst durchaus vielversprechend. Schlussendlich musste sich doch mit der Ausgabe der Inhalte von Variablen auf dem Bildschirm per *printf()* beholfen werden.
In dieser Arbeit konnte aus zeitlichen Gründen nicht untersucht werden, ob die beobachteten Fehler ein generelles Problem bei der Benutzung von GDB über WLAN darstellen oder in der Umsetzung von WLAN auf seriell durch den Matchport begründet sind. Es bleibt zu prüfen ob eine Veränderung der Parameter der UART Schnittstelle eine Verbesserung in diesem Verhalten bringt.

8.3 Akkulaufzeit

Im voll geladenen Zustand, welcher spätestens nach ca. 3 Stunden Ladezeit erreicht sein sollte, sind 7,6 Volt Spannung an der Batterie zu messen. Ein Spannungsabfall unter 7

Volt macht sich bereits in einem schwächeren Ansprechverhalten der Motoren bemerkbar. Sinkt die Batteriespannung unter 6,8 Volt schaltet das Blackfin Board ab. Im Testbetrieb geschah dies nach ca. 20 großen Probefahrten (Programm *probefahrt* in dem 9 verschiedene Punkte angefahren werden). Negativ an der Spannungsversorgung des Roboters fällt auf, dass keine Ladekontrolle vorhanden ist, so dass dieser nur unter Aufsicht geladen werden sollte.

9 Abschluss

9.1 Bekannte Fehler und Verbesserungsvorschläge

Zu den schon unter 8. “Auswertung der Teilkomponenten” genannten Vorschlägen sind noch folgende Punkte zu erwähnen:
Im jetzigen Zustand verfügt das Fahrzeug über keine Hinderniserkennung. Um es in einem größeren Maßstab einsetzen zu können ist eine Hinderniserkennung und Umfahrung wünschenswert. Auch hierfür ist die Integration der Kamera denkbar. Als Zusatzmodul biete sich der Anbau eines Ultraschallmoduls an.
Im Falle einer Überarbeitung oder eines Neuentwurfs des Erweiterungsboards sollte der Kompass umpositioniert werden. Die nördliche Seite des Kompasses sollte in Fahrtrichtung zeigen. So muss man nicht zusätzlich zum gemessenen Wert die Position des Kompasses auf dem Roboter betrachten und die Werte werden transparenter.
Weiterhin bedarf die Telnet Verbindung der Konfiguration. Die Syntax-Vervollständigung funktioniert zwar, wird aber nicht korrekt angezeigt. Programme die mit *getch()* auf Benutzereingaben reagieren interpretieren genannte Eingaben doppelt.

9.2 Fazit

Die gesteckten Ziele dieser Bachelorarbeit wurden größtenteils erfüllt.
Das Betriebssystem uClinux biete eine gute Basis für weitere Softwareerweiterungen. Mit der Anbindung des I^2C Bus ist eine gute Grundlage für weitere Hardwareerweiterungen gegeben. Allerdings fehlt weiterhin ein Weg diese Plattform zu Debuggen. So wurde mit dem GDBserver zwar eine vielversprechende Möglichkeit gefunden, welche aber aufgrund

der Qualität der WLAN Verbindung nicht nutzbar ist.
Der Roboter demonstriert auf der Teststrecke verlässlich, dass es möglich ist einem Fahrzeug, mit Hilfe von nur zwei Sensoren und künstlichen Landmarken, zur Autonomie zu verhelfen.
Der Kompass zeigt aber auch Grenzen auf. Seine Störanfälligkeit fällt im Rahmen der Teststrecke nicht stark ins Gewicht. Störeinflüsse können hier minimiert oder gar ausgeschlossen werden. Auch hat ein unkorrigierter Geradeauslauf auf kurzen Strecken keine große Auswirkung.
Bei dem Einsatz in einem größeren Maßstab allerdings können Störeinflüsse nicht mehr ausgeschlossen werden. Längere Strecken machen ein korrektes Geradeauslaufverhalten zwingend erforderlich. Der Roboter muss hierfür um weitere Sensoren erweitert werden (Anregungen hierzu wurden im Kapitel 8.“Auswertung der Teilkomponenten” im Abschnitt “8.1.1 Geradeausfahrt” genannt). Der Kompass wird deshalb von seiner zentralen Rolle weiter in den Hintergrund treten.
Die RFID Technologie zeigt einen simplen und zuverlässigen Weg künstliche Landmarken zu realisieren.
Die Kopplung mit der Steuerungssoftware in Verbindung mit der Pflege einer RFID-Transponder-Datenbank und dem Entwurf einer passenden Wegfindung bietet Potential für weitere Arbeiten.

10 Quellenverzeichnis

[brinker07] *Embedded Linux - Praktische Umsetzung mit uClinux*, Brinker, Degenhardt, Kupris, VDE-Verlag 2007

[finkenzeller06] *RFID Handbuch - Grundlagen und praktische Anwendungen induktiver Funkanlagen, Transponder und kontaktloser Chipkarten*, 4. Auflage, Klaus Finkenzeller, Hanser Verlag 2006

[moedrath09] *Orientierung eines vernetzten autonomen Fahrzeugs im Zusammenspiel mit einer globalen Datenbank und RFID Positionsmarken*, Diplomarbeit, Dirk Mödrath, 2009

[plate07] *Linux Hardware Hackz - Messen, Steuern und Sensorik mit Linux*, Jürgen Plate, Hanser Verlag 2007

[www-bstlctrl] `http://spritesmods.com/?art=bristlebot`, Webseite, Controllable BristleBot, letzter Abruf: 12.07.2009

[www-doxygen] `http://www.doxygen.org`, Webseite, Doxygen Doku Tool, letzter Abruf: 12.07.2009

[www-forum1] `https://blackfin.uclinux.org/gf/project/uclinux-dist/forum/?_forum_action=ForumMessageBrowse&thread_id=33336&action=ForumBrowse&forum_id=39`, uClinux Forum, Beitrag zu debugging, letzter Abruf: 12.07.2009

[www-gdb] `http://www.gnu.org/software/gdb/`, Webseite, GDB: The GNU Project Debugger, letzter Abruf: 12.07.2009

[www-gnu] http://www.gnu.org/, Webseite, The GNU Project, letzter Abruf: 12.07.2009

[www-mct] http://www.mct.net/product/if232lp.html,Webseite, IF232LV RS232 Interface Module der Firma MMCT Paul & Scherer Mikrocomputertechnik GmbH, letzter Abruf: 12.07.2009

[www-raflatac] http://www.rfidplaza.com/pages/page.aspx?c=productdetail&aid=3056, Webseite, Bezugsquelle der Transponder, letzter Abruf: 12.07.2009

[www-section5] http://www.section5.ch, Webseite, Homepage des ICEBear Herstellers, letzter Abruf: 12.07.2009

[www-sonmicro] http://www.sonmicro.com, Webseite, SonMicro Electronics, letzter Abruf: 12.07.2009

[www-surveyor] http://www.surveyor.com/SRV_info.html, Webseite, Surveyor SRV-1, letzter Abruf: 12.07.2009

[www-uclinux] http://blackfin.uclinux.org, Webseite, das uClinux Projekt , letzter Abruf: 12.07.2009

11 Abbildungsverzeichnis

12 Anhang

12.1 Grundkonfiguration des Roboters

Anbei eine Kopie der wichtigsten Abschnitte des dritten Kapitels aus [moedrath09].

Diese Liste enthält die wichtigsten Hardwarebausteine des Roboterfahrzeugs, die in dieser Arbeit und in den weiteren Arbeiten zu diesem Projekt gebraucht werden. Wird eine genauere Liste, Schaltpläne oder Pinbelegungspläne zu Hardware gebraucht, können diese unter [survey] angesehen und herunter geladen werden.

3.2 Die UART Schnittstellen

Auf dem Roboterfahrzeug befinden sich, wie bereits in Kapitel 3.1 erwähnt, zwei UART Schnittstellen. Diese beiden Schnittstellen verbinden den Blackfin Prozessor und den Matchport (s. Kap. 3.3). Im Auslieferungszustand ist allerdings nur der erste UART(UART0) durchgeschaltet und funktionsbereit. Um den zweiten UART(UART1) benutzen zu können, müssen auf der untersten Platine des Roboterfahrzeugs zwei Widerstände eingelötet werden. *Abb. 3.2* zeigt das Platinenlayout. Die Stelle, an der die Widerstände eingelötet werden müssen, ist mit einem roten Rechteck markiert.

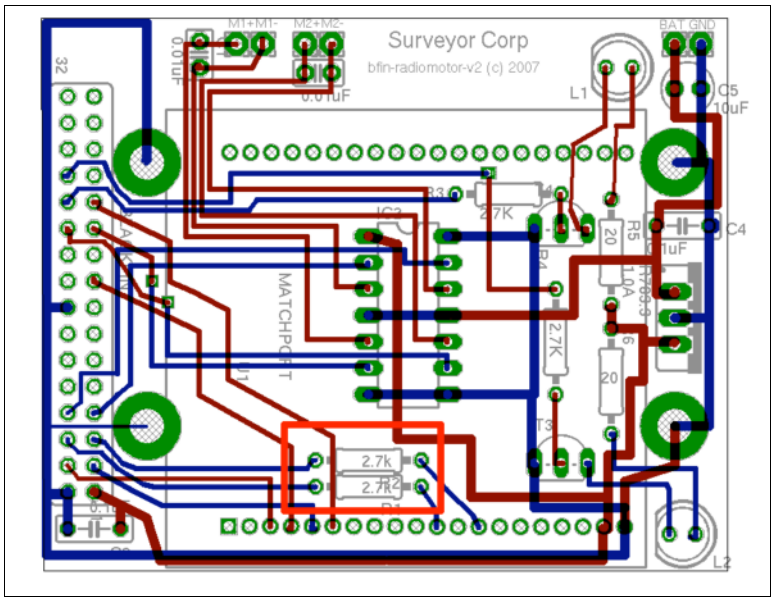


Abb. 3.2 Platinenlayout unterste Platine [survey]

Zusätzlich ist zu beachten, dass nur UART0 die Möglichkeit der Hardware-flusskontrolle bietet. Da es diese Möglichkeit für UART1 nicht gibt, empfiehlt es sich bei dieser Schnittstelle eine Softwareflusskontrolle einzustellen. Damit sich die UART Schnittstellen korrekt verhalten, müssen sie auf beiden Seiten, am Matchport und am Prozessor (im Betriebssystem), gleichermaßen eingestellt werden. Wie diese Einstellungen am Matchport vorgenommen werden, wird in Kapitel 3.3 behandelt.

3.3 Der Matchport

Der Matchport ist eine kostengünstige WLAN Lösung für den eingebetteten Bereich. Er bietet die Möglichkeit der Umwandlung von seriellen Signalen in WLAN Signale. Hieraus ergibt sich die Möglichkeit, trotz nicht vorhandener Netzwerkschnittstelle das zu entwickelnden System netzwerkfähig zu machen. In *Abb. 3.3* wird dargestellt, wie der Matchport aufgebaut ist.

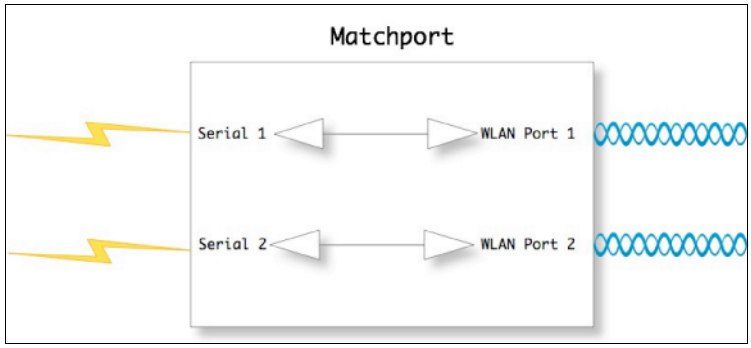


Abb. 3.3 Struktur des Matchports

Unterstützt werden auf der WLAN Seite die Standards IEEE 802.11b; 802.11g und die Verschlüsselungsprotokolle WEP (Wired Equivalent Privacy), WPA-PSK (Wi-Fi Protected Access – Pre-Shared Keys) und WPA2.

Auf der seriellen Seite werden Geschwindigkeiten von 300 bps (bits per second) bis zu 921000 bps, verschiedene Paritätsmodi, serielle Formate und Flusskontrollenmodi unterstützt. Eine komplette Liste der Eigenschaften, Software und weitere Dokumentationen des Matchport findet man unter [lan_mat].

3.3.1 Arten der Konfiguration

Der Matchport wird über den internen Web Server, Telnet, Serielle Leitung, SNMP (Simple Management Network Protocol / allerdings nur lesen), oder eine für Windows erhältliche DeviceInstaller Software verwaltet.

Die letzten beiden Varianten wurden in dieser Arbeit nicht getestet und können daher auch nicht bewertet werden. Das Konfigurieren über den internen Web Server ist nicht zu empfehlen, da hier nicht möglich ist nur einzelne Parameter des Matchports zu ändern, sondern es müssen immer alle Parameter eingestellt werden. Ein weiteres Problem bei dieser Konfigurationsmöglichkeit ist es, das beim Verbinden mit internen Web Server nicht die aktuellen Parameterwerte des Matchports geladen werden, sondern nur voreingestellte Standardwerte. Wegen dieser Probleme, die sich im Test ergeben haben, wird auf diese Art der Konfiguration nicht näher eingegangen.

Das Konfigurieren über Telnet und die Serielle Schnittstelle haben sich als die besten Varianten der Einstellung erwiesen. Daher werden sie in den folgenden Kapiteln näher erläutert.

Konfiguration über Telnet

Für die Konfiguration über Telnet benötigt man einen Telnet Klient, bei dem sich der Standardport für Telnet (23) ändern lässt. Der Port muss auf den Wert *9999* geändert werden, da dies der Standardkonfigurationsport des Matchports ist. Dafür bietet sich unter Linux und unter Windows der Befehl *telnet* an. Hat das Roboterfahrzeug beispielsweise die IP-Adresse 192.168.2.100 so sieht der Befehl wie folgt aus:

telnet 192.168.2.100 9999

Nachdem die Verbindung aufgebaut ist wird man aufgefordert mit *Enter* zu bestätigen. *Abb. 3.6* zeigt den Bildschirm nach der Bestätigung.

3.3.1.1 Konfiguration über die Serielle Schnittstelle

Um die Konfiguration über die Serielle Schnittstelle vorzunehmen, werden ein USB to Serial Umsetzer, Treiber und Softwaretools benötigt. Der Umsetzer enthält einen Silabs CP210x USB to UART Chip und wird von der Firma Surveyor kostenlos zur Verfügung gestellt. *Abb. 3.4* zeigt diesen Umsetzter.

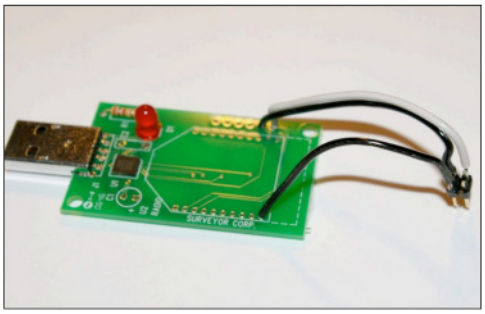


Abb. 3.4 USB to Serial Platine [sur_ser]

Allerdings muss der Umsetzer nach Erhalt erst einmal, wie in der Beschreibung unter [sur_ser] dokumentiert, modifiziert werden.

Die Treiber für diesen Umsetzter erhält man unter [sur_dri]. In *Abb. 3.5* wird gezeigt, wie der Umsetzer am Roboter angeschlossen werden muss.

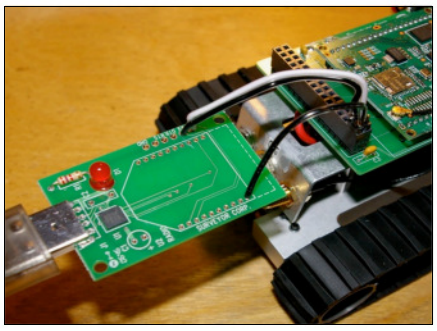


Abb. 3.5 Anschluss des Umsetzers am Roboterfahrzeug [sur_ser]

Als Softwaretools wurden in dieser Arbeit *Putty* für Windows und das Terminalprogramm *Kermit* für Linux getestet. Wie man diese Tools benutzt und sie einstellt, wird in Anhang A Schritt für Schritt erklärt.

Abb. 3.6 zeigt wie das Fenster nach vollständig aufgebauter Verbindung aussehen sollte.

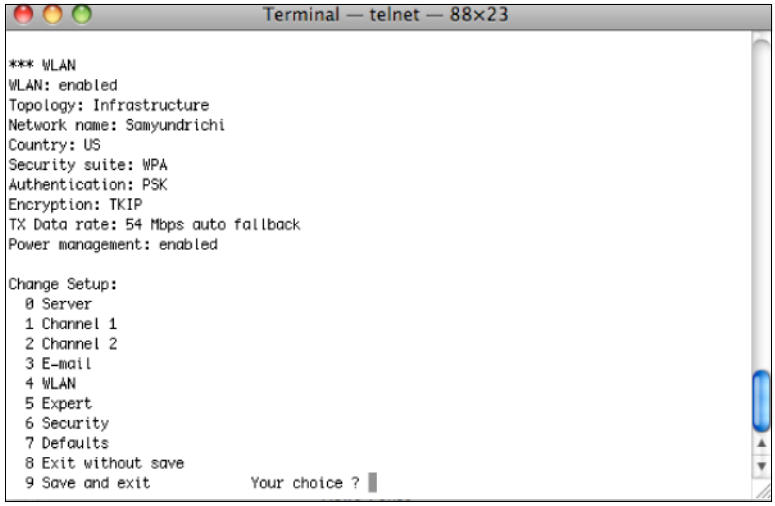


Abb. 3.6 Konfigurationsfenster Matchport

3.3.2 Wichtige Einstellparameter des Matchports

Abb. 3.6 zeigt die Auswahl der Einstellungsmöglichkeiten des Matchports. Diese werden in neun Unterpunkte gegliedert. Dabei werden mit den ersten sieben Menüpunkten die Parameter des Matchports verändert. In diesem Abschnitt soll auf die, für diese Arbeit, wichtigsten Teile näher eingegangen werden. Dabei kann es sein, dass einzelne Unterpunkte komplett ausgelassen werden, da sie keine Relevanz für die Arbeit haben. Falls weitere Erklärungen nötig sind, befindet sich unter [lan_mat] eine komplette Dokumentation. Die Konfiguration die in dieser Arbeit verwendet wurde befindet sich in Anhang B.

3.3.2.1 Server

Parameter:

- *Network Mode*: Hier muss zwingend 1 für drahtlose Verbindung eingestellt werden.
- *IP-Adress*: Wenn das Roboterfahrzeug seine Adresse von einem DHCP-Server bekommt. wird hier 0.0.0.0 eingestellt werden. Ansonsten wird hier die IP-Adresse eingestellt, die für das Roboterfahrzeug vorgesehen ist.
- *Set Gateway*: Gibt es ein Gateway (typischerweise ein Router) im Netzwerk wird hier Y eingestellt. N wird eingestellt, falls kein Gateway vorhanden ist
- *Gateway IP addr*: Hier wird die IP-Adresse des Gateways eingestellt
- *Netmask*: Mit diesem Parameter wird die Subnetmaske des Netzwerks eingestellt. Dabei bedeudet 8 -> 255.255.255.0 und 16 -> 255.255.0.0 usw.

3.3.2.2 Channel 1 & Channel 2

Parameter:

- *Baudrate:* Einstellung der Geschwindigkeit der Seriellen Schnittstellen.
- *I/F Mode:* Mit diesem Parameter wird die Schnittstellenart, die Paritätsart, die Anzahl der Datenbits und der Stopbits eingestellt.
- *Flow:* Hier wird die Art der Flusskontrolle eingestellt. Dabei gibt es folgende Möglichkeiten: 0 -> keine Parität, 1 -> Softwareflusskontrolle und 2 - > Hardwarflusskonstrolle
- *Port:* Hier legt man die Portnummer fest, unter der diese Serielle Schnittstelle von der WLAN Seite aus angesprochen werden kann

- *Connect Mode:* Hier werden die allgemeinen Verbindungs-eigenschaften und der Transportschichttyp (UDP/TCP) der WLAN Seite eingestellt. Die Verbindungseigenschaften sagen etwas darüber aus, wer dafür sorgt, dass eine Netzwerkverbindung entsteht. Die Art der Verbindungseinleitung wird ebenfalls mit diesem Parameter eingestellt.

  Dabei gibt es folgende Standardeinstellungen:
  CO -> TCP, Verbindungsaufbauten werden ausschließlich von der WLAN Seite her eingeleitet. Dabei werden alle an der WLAN Seite eingehenden Verbindungen akzeptiert.

  C1 -> TCP, Verbindungsaufbauten werden durch beide Seiten, Seriell und WLAN, eingeleitet. Dabei werden alle eingehenden Verbindungen der WLAN Seite akzeptiert und die Serielle Seite kann durch ein beliebiges Zeichen einen Verbindungs-aufbau einleiten.

  01 -> TCP, Verbindungsaufbauten werden ausschließlich durch die Serielle Seite eingeleitet. Der Verbindungsaufbau wird durch ein beliebiges Zeichen von der Seriellen Seite eingeleitet.

- *Remote IP Address:* Dieser Parameter ist sehr wichtig, für den Fall, dass unter *Connect Mode* ein Verbindungsaufbau der Seriellen Seite gewählt wird. Bei dieser Einstellung des Connect Mode, wird hier die IP Adresse des Netzwerkteilnehmers, mit dem sich der Matchport verbinden soll, eingestellt. Baut die WLAN-Seite die Verbindung auf, bekommt dieser Parameter den Wert 0.0.0.0.

- *Remote Port:* Hier wird der zu *Remote IP Address* dazugehörige Port des Netzwerkteilnehmers festgelegt. Bei Ein-leitung des Verbindsaufbaus durch die WLAN Seite wird hier 0 eingestellt.

3.3.2.3 WLAN

Parameter:

- *Topology:* Hier wird eingestellt um welche Netzwerktopologie es sich handelt.
- *Network Name:* Dieser Parameter enthält die Einstellung der SSID (Service Set Identifier) des Netzwerks.
- *Security Suite:* Hier wird das Verschlüsselungsprotokoll der WLAN Verbindung eingestellt.
- *Change Key:* Durch diesen Parameter wird festgelegt, ob der aktuell eingestellte Schlüssel geändert wird. Bei Eingabe von Y erfolgt die Aufforderung zur Festlegung eines neuen Schlüssels im nächsten Parameter.
- *Encryption:* Hier werden die Verschlüsselungseigenschaften des gewählten Protokolls eingestellt

3.3.2.4 Defaults, Exit without save, Save and exit

Mit diesen Parametern wird der Matchport auf seine Standardeinstellung zurückgestellt, die vorher eingegebenen Parameter gespeichert, oder diese verworfen.

Mifare® 1K Memory Organization

		Byte Number within Block																
Sector	**Block**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**	**Description**
0	0																	Manufacturer Block
	1																	Data Block
	2																	Data Block
	3	Key A						Access Bits				Key B						Sector Trailer 0
1	0																	Data Block
	1																	Data Block
	2																	Data Block
	3	Key A						Access Bits				Key B						Sector Trailer 1
...	...																	Data Block
...	...																	Data Block
...	...																	Data Block
...	...	Key A						Access Bits				Key B						Sector Trailer ...
14	0																	Data Block
	1																	Data Block
	2																	Data Block
	3	Key A						Access Bits				Key B						Sector Trailer 14
15	0																	Data Block
	1																	Data Block
	2																	Data Block
	3	Key A						Access Bits				Key B						Sector Trailer 15

Abbildung 12.1: Schaubild Mifare Speichereinteilung

12.2 Schaubild Mifare Speichereinteilung

12.3 Belegung des Expansion Headers des Blackfin Boards

PinBFin	Signal	Description of Expansion Board Signal
1	3.3V	3.3V voltage regulator
2	GND	ground
3	TX0	Matchport RX1 (pin 9)
4	RX0	Matchport TX1 (pin 5)
5	TX1/TMR6	2nd UART TX or timer 6 pwm/ppm
6	RX1/TMR7	2nd UART RX or timer 7 pwm/ppm
7	TMR2	h-bridge CE1 (pin 2) - L motor/servo pwm/ppm
8	TMR3	h-bridge CE2 (pin 13) - R motor/servo pwm/ppm
9	SPI_MOSI	
10	SPI_MISO	
11	SPI_SCK	
12	SPI_SEL	spi_ssel3 / spi_slave (set by jumper on CPU card)
13	SPI_SEL2	
14	**I2C SCL**	i2c scl - Clock
15	**I2C SDA**	i2c sda - Data
16	**GND**	ground
17	GPIO-H0	serial flow ctrl in - Matchport RTS0 (pin 7) ***
18	GPIO-H1	ultrasonic module trigger (out)
19	GPIO-H2	battery low-voltage detect (bat $<$ 6V == hi)
20	GPIO-H3	master/slave processor detect (master == low)
21	GPIO-H4	h-bridge IN1 (pin 6) - L motor direction
22	GPIO-H5	h-bridge IN2 (pin 9) - R motor direction
23	GPIO-H6	serial flow ctrl out - Matchport CTS0 (pin 11) ***
24	GPIO-H7	laser 1 (transistor switch)
25	GPIO-H8	microSD SPI select on RCM board
26	GPIO-H9	laser 2 (transistor switch)
27	GPIO-H10	ultrasonic module #1 input
28	GPIO-H11	ultrasonic module #2 input
29	GPIO-H12	ultrasonic module #3 input
30	GPIO-H13	ultrasonic module #4 input
31	GPIO-H14	
32	GPIO-H15	

12.4 Logfiles probefahrtkurz

probefahrtkurz 1:
referenzgrad gemessen: 2617
referenzgrad korrigiert: 1717

tag wurde gelesen id: 41 x: 22 y: 22
tag wurde gelesen id: 42 x: 32 y: 22
Zieltag erreicht! 1 / 3 ++++++++++++++++++++++++++
tag wurde gelesen id: 42 x: 32 y: 22
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 59 x: 32 y: 32
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 54 x: 59 y: 41
Zieltag erreicht! 2 / 3 ++++++++++++++++++++++++++
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 65 x: 32 y: 51
tag wurde gelesen id: 52 x: 42 y: 61
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 65 x: 32 y: 51
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 52 x: 42 y: 61
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 51 x: 32 y: 61
Zieltag erreicht! 3 / 3 ++++++++++++++++++++++++++

probefahrtkurz 2:
referenzgrad gemessen: 2600
referenzgrad korrigiert: 1700
tag wurde gelesen id: 41 x: 22 y: 22
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 42 x: 32 y: 22
Zieltag erreicht! 1 / 3 ++++++++++++++++++++++++++
tag wurde gelesen id: 42 x: 32 y: 22
tag wurde gelesen id: 59 x: 32 y: 32
tag wurde gelesen id: 53 x: 49 y: 41
tag wurde gelesen id: 56 x: 59 y: 31
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
Zieltag erreicht! 2 / 3 ++++++++++++++++++++++++++
tag wurde gelesen id: 56 x: 59 y: 31
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 56 x: 59 y: 31
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41

tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 56 x: 59 y: 31
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 56 x: 59 y: 31
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 56 x: 59 y: 31
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 56 x: 59 y: 31
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 56 x: 59 y: 31
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 65 x: 32 y: 51
tag wurde gelesen id: 52 x: 42 y: 61
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 51 x: 32 y: 61
Zieltag erreicht! 3 / 3 +++++++++++++++++++++++++

probefahrtkurz 3:
referenzgrad gemessen: 2615

referenzgrad korrigiert: 1715
tag wurde gelesen id: 41 x: 22 y: 22
Fehler bei Auth, Zurueckfahren!
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 42 x: 32 y: 22
Zieltag erreicht! 1 / 3 +++++++++++++++++++++++++
tag wurde gelesen id: 42 x: 32 y: 22
tag wurde gelesen id: 59 x: 32 y: 32
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 54 x: 59 y: 41
Zieltag erreicht! 2 / 3 +++++++++++++++++++++++++
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 65 x: 32 y: 51
tag wurde gelesen id: 52 x: 42 y: 61
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 51 x: 32 y: 61
Zieltag erreicht! 3 / 3 +++++++++++++++++++++++++

probefahrtkurz 4:
referenzgrad gemessen: 2615
referenzgrad korrigiert: 1715
tag wurde gelesen id: 41 x: 22 y: 22
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 42 x: 32 y: 22

Zieltag erreicht! 1 / 3 ++++++++++++++++++++++++++
tag wurde gelesen id: 42 x: 32 y: 22
tag wurde gelesen id: 59 x: 32 y: 32
tag wurde gelesen id: 54 x: 59 y: 41
Zieltag erreicht! 2 / 3 ++++++++++++++++++++++++++
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
zu viele ausrichtungsversuche, drehung wird abgebrochen
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 65 x: 32 y: 51
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 51 x: 32 y: 61
Zieltag erreicht! 3 / 3 ++++++++++++++++++++++++++

probefahrtkurz 5:
referenzgrad gemessen: 2610
referenzgrad korrigiert: 1710
tag wurde gelesen id: 41 x: 22 y: 22
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 42 x: 32 y: 22
Zieltag erreicht! 1 / 3 ++++++++++++++++++++++++++
tag wurde gelesen id: 42 x: 32 y: 22
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 59 x: 32 y: 32
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 54 x: 59 y: 41
Zieltag erreicht! 2 / 3 ++++++++++++++++++++++++++
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41
tag bei drehung gelesen, drehung abgebrochen
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 54 x: 59 y: 41
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 53 x: 49 y: 41

zu viele ausrichtungsversuche, drehung wird abgebrochen
Fehler bei Auth, Zurueckfahren!
tag wurde gelesen id: 65 x: 32 y: 51
tag wurde gelesen id: 52 x: 42 y: 61
tag bei drehung gelesen, drehung abgebrochen
tag wurde gelesen id: 51 x: 32 y: 61
Zieltag erreicht! 3 / 3 ++++++++++++++++++++++++++

12.5 Inhaltsverzeichnis der DVD

Auf der beiligenden DVD sind folgende Ordner enthalten:

dokumente Enthält dieses Dokument als PDF Datei sowie Bildmaterial

eagle entwurfsdateien Enthält die Quelldaten für den Schaltplan des Erweiterungsboards, mit dem Layout Programm "Eagle" erstellt.

kompass cmps03 Datenblatt des Kompass Moduls

quellcode Quellcode aller in dieser Arbeit verfassten Software, inklusive den kompilierten Programmen. Enthält auch das Motormodul.

quellen enthält die Dateien:

blackfin-toolchain-uclinux-SVN.i386.tar.gz - Toolchain

blackfin-toolchain-linux-uclibc-SVN.i386.tar.gz - Toolchain Librarys

u-boot-1.1.6-2008R1.5.tar.gz - Quellen des Bootloaders

uclinux-dist-trunk-svn-7974.src.tar.gz - uClinux Quellen

uclinux-dist-trunk-svn-7974.src_konfiguriert.tar.gz - uClinux Quellen bereits konfiguriert, so wie sie in dieser Arbeit verwendet wurden

rfid modul sm130 Application Note, Data Sheet und User Manual des verwendeten RFID Moduls, sowie die Windows Tools inklusive der verwendeten Firmware. Enthält auch das Angebot der Mifare Etiketten.

spannungswandler Datenblatt des Spannungswandlers

programmdoku doxygen Die mit Doxygen erstellte Programmdoku. Mit index.html starten.

uclinux image mit flashtools und bootloader Die letzte Version des uClinux Images (enthält alle Programme), sowie der gepatchte Bootloader inklusive den Tools um alles auf den Roboter zu flashen