Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für Integrierte Hard- und Softwaresysteme

Diplomarbeit

Konzeption und Umsetzung eines
„Remote Engineering“ Praktikums für mobile Roboter

---

eingereicht am:	19. Dezember 2007
von:	Friedemann Schmidt
Matrikelnummer:	35217
Verantwortlicher Professor:	Prof. Dr.-Ing. habil. A. Mitschele-Thiel
Hochschulbetreuer:	Dr.-Ing. K. Henke
Inventarisierungsnummer:	2007-12-19/166/II02/2235

Kurzfassung

Diese Diplomarbeit beschäftigt sich mit der Konzeption und Umsetzung eines Remote Engineering Praktikums. Dazu werden die Einsatzgebiete und der grundsätzliche Aufbau skizziert. Anhand der sich daraus ergebenen Anforderungen wird eine kurze Marktübersicht aufgestellt. Der vom Heise-Verlag entwickelte c't-Bot geht daraus als die geeignetste Plattform hervor. Neben der Beschreibung dieses mobilen Roboters, werden die Eigenschaften und Eigenheiten der verwendeten Sensoren erläutert. Zum Einstieg erfolgt noch eine kurze Einführung in die dafür verfügbare Software.

Der Hauptteil der Arbeit beschäftigt sich mit den für die Realisierung notwendigen Modifikationen und Erweiterungen. Dazu gehören unter anderem eine Akku-Überwachung, eine Ladestation sowie der Ausbau der Software. Ein Test-Applet ermöglicht die Überwachung, die Steuerung und die Programmierung des mobilen Roboters über eine Netzwerkverbindung. Eine kurze Bedienungsanleitung rundet die Arbeit ab.

Schlagwörter: Akku-Überwachung, c't-Bot, Diplomarbeit, Ladestation, Mikrocontrollerprogrammierung, mobiler Roboter, Praktikum, Remote Engineering, RemoteLab

Abstract

This diploma thesis expounds the development and realization of a remote engineering lab. As a first step, the basic structure is described and its possible fields of application are outlined. In the following part, a brief market overview is given, from which c't-Bot, developed by "Heise Verlag", emerges as the most suitable platform. The technical details of this mobile robot are then described, as well as the characteristics of the utilized sensors. A short introduction in the available software for this platform should ease the start.

The main part of this thesis describes the modifications and enhancements needed to build the remote engineering lab. These modifications include a battery monitor, a charging station and several software enhancements. A test applet is developed for the purpose of monitoring, controlling and programming the mobile robot via a network connection. A short manual for this platform rounds this work off.

Keywords: battery monitor, c't-Bot, diploma thesis, charging station, programming of microcontroller, mobile robot, lab, remote engineering, RemoteLab

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung und Zielstellung der Arbeit

Im Rahmen des RemoteLab-Projekts des Fachgebietes für Integrierte Hard- und Softwaresysteme (IHS) wurden verschiedene Praktika zur ferngesteuerten Programmierung entwickelt. Ziel dieser Arbeit ist es, durch die Konzeption eines neuen Versuches, das Praktikums-Angebot zu erweitern. Neben Grundlagenpraktika sollen auch Spezialpraktika im Bereich Remote Engineering bzw. der Entwicklung von Software-/Hardware-Lösungen zur Durchführung über das Internet angeboten werden. Auch ist ein Demonstrator für Lehrzwecke vorgesehen um den Praxisbezug in den Vorlesungen zu unterstreichen sowie Interesse an der Forschung zu wecken. Zusätzlich soll auch eine Online-Bedienbarkeit für beliebige Benutzer, die Wissbegierde schüren, und Anregungen zum Studium gegeben werden. Weiterhin wird für Forschungszwecke im Fachgebiet, insbesondere für den Aufbau des „Ad-hoc Labs“, eine Plattform benötigt.

Dies ist mit den bisherigen Praktikumsmodellen nicht zu bewerkstelligen. Diese Arbeit widmet sich daher der Konzeption und der prototypischen Implementierung einer neuen Praktikums- und Forschungsplattform.

Die dazu nötigen Anforderungen und der daraus abgeleitete grundsätzliche Aufbau werden im Kapitel 2 dargelegt. Dazu gehört neben dem reinen hardwareseitigen Aufbau und Schutzmechanismen vor allem die Netzwerk-Anbindung.

Kapitel 3 widmet sich der Suche nach einem geeigneten Modell. Es wird eine Auswahl der marktüblichen Kleinroboter vorgestellt und anhand der Anforderungen für die beabsichtigten Anwendungen eine Entscheidung getroffen.

Der gewählte Roboter wird im Kapitel 4 vorgestellt und beschrieben. Dazu zählen alle zur Programmierung erforderlichen Informationen, Charakteristik der Sensoren sowie eine kurze Einführung in die für diesen Roboter verfügbare Software.

Kapitel 5 beinhaltet die Aufzeichnung der notwendigen hardwareseitigen Veränderungen und Erweiterungen. Dazu zählen vor allem die Entwicklung einer Akku-Überwachung sowie einer Ladestation.

Die für eine Realisierung notwendigen Modifikationen und Erweiterungen der Software werden im Kapitel 6 beschrieben.

Kapitel 7 stellt eine prototypische Implementierung einer Client-Software dar. Mit ihr soll vor allem die Funktionsfähigkeit der Hard- und Software nachgewiesen werden.

Abschließend werden mit Kapitel 8 eine kurze Zusammenfassung über den Stand der Umsetzung sowie ein weiterer Ausbau aufgezeigt.

Im Anhang befinden sich Beilagen zu einzelnen Kapiteln. Dazu gehören unter anderem Installationshinweise, Softwareschnittstellen und Schaltpläne, welche den Hauptteil dieser Arbeit ergänzen.

2 Konzeption des Praktikumsplatzes

Für die vielseitigen beabsichtigten Anwendungen erschien ein kleiner mobiler Roboter als die passendste Lösung. Dieser sollte über eine großzügige Ausstattung mit Sensoren verfügen, um mit ihm unterschiedlichste Praktikumsaufgaben bewältigen zu können. Für die ebenso erwartete Nutzung dieses Modells, im Rahmen des geplanten „Ad-hoc Lab“-Projektes, des Fachgebietes IHS, muss dieser Roboter mit einem WLAN-Modul ausgestattet sein. Dies würde gleichzeitig die Realisierung einer internetbasierten Steuerung vereinfachen.

2.1 Grundstruktur

Der grundlegende Hardwareaufbau sieht folgende Konfiguration vor. Der Roboter bewegt sich in einem fest umrandeten Spielfeld. Dieses wird von einer Web-Cam überwacht. Das WLAN-Modul des Roboters verbindet sich über eine verschlüsselte Verbindung mit dem WLAN-Access Point. Ein Server übernimmt die Zugangskontrolle und Hardwarereservierung sowie die Weiterleitung der Datenströme von Kamera und Roboter an die Bediener übers Internet. Die Steuerung und Programmierung soll über ein Java-Applet erfolgen.

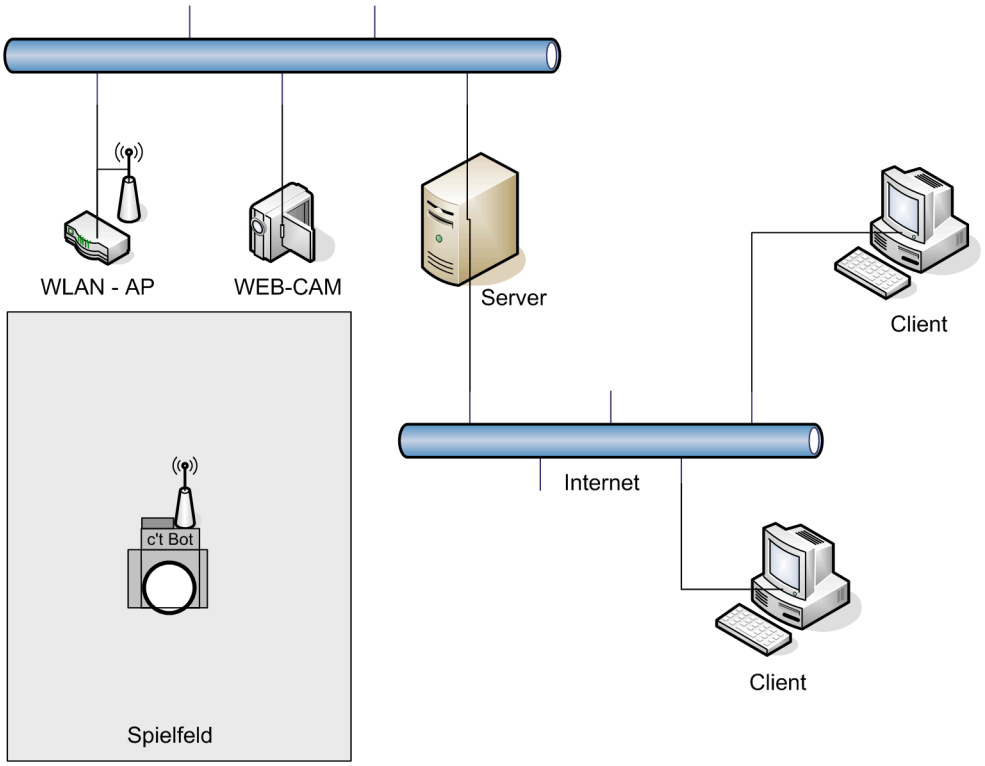


Abbildung 1: Struktur und Netzwerkanbindung des Praktikumsplatzes

2.2 Netzwerkanbindung

Der Server übernimmt in der Netzwerkanbindung eine zentrale Rolle. Neben der Weiterleitung ergeben sich noch weitere erforderliche Aufgaben. Dazu gehören eine Identifikation bzw. Zugangskontrolle mit Benutzernamen und Passwort, eine Zuteilung der Praktikumshardware zu einem Klienten, sowie auch ein Entzug der Hardware nach Beendigung der Verbindung bzw. Zeitüberschreitung. Wünschenswert wäre auch noch eine zusätzliche Überwachung der Vitalität und des Ladezustandes des Akkus vom mobilen Roboter.

Desweiteren ist auch eine Bewertungsfunktion der erfolgreichen Ausführung einer gegebenen Praktikumsaufgabe erstrebenswert, auf jeden Fall sollte eine Aufzeichnung der für eine manuelle Bewertung notwendigen Parameter nebst Web-Cam Stream erfolgen.

Als Server bietet sich daher der hier im Fachgebiet IHS entwickelte RemoteLab-Server (RLS) an.

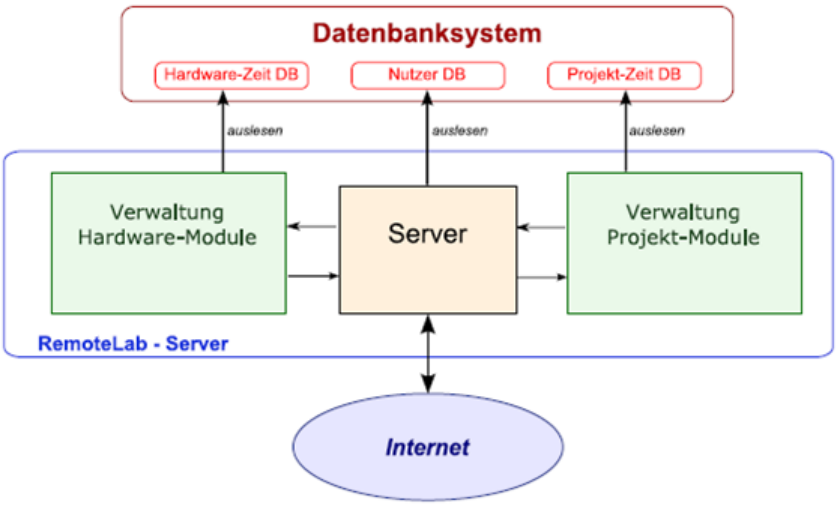


Abbildung 2: Struktur des RemoteLab-Servers

„Der RemoteLab-Server besteht im Wesentlichen aus drei Verwaltungskomponenten. Die Basisserver-Komponente „Server“ ist für die Netzwerkkommunikation und die Nutzerverwaltung verantwortlich. Die Verwaltung für die Projekt-Module (Kommandointerpreter) ist für die im System aktiven Kommandointerpreter zuständig. Die Verwaltung für die Hardware-Module dient der Reservierung von Hardware-Modellen und den dazu gehörigen Kontrollstrukturen.“ [Sch06]

Durch die dynamische Plug-In-Architektur des RemoteLab-Servers lassen sich leicht weitere Elemente integrieren. Für die konkrete Realisierung des Praktikumsplatzes und Einbindung in den RLS ist dazu ein eigenes Hardware- und Projekt-Modul zu entwickeln. Dies aber ist nicht Bestandteil dieser Arbeit.

2.3 Programmierung und Validierung

Die Softwareentwicklung soll auf dem Client-PC des Nutzers erfolgen. Betriebssystemunabhängigkeit kann durch die Verwendung der Entwicklungsumgebung Eclipse erreicht werden. Dazu soll ein vorgefertigter Workspace zum Download bereitgestellt werden. Darin wird ein komplett lauffähiger C-Code zur Verfügung gestellt. Den Einstieg in die Programmierung erleichtern dabei vorgefertigte Verhaltensroutinen. In der Regel sollte es ausreichen, nur durch Änderung dieser, die gegebenen Praktikumsaufgaben zu erfüllen.

Vorteilhaft wäre, wenn für das Praktikumsmodell auch eine Simulationsumgebung zur Verfügung steht. Dann kann die Validierung der Verhaltensroutinen in zwei Schritten erfolgen. Zuerst soll dann der Praktikant seinen angepassten Code auf dem Simulator testen. Erst nach erfolgreichem Test in der Simulation ist der Einsatz auf dem realen mobilen Praktikumsroboter zu erproben. Dies würde nicht nur die Praktikumsmodelle schonen, sondern auch gleichzeitig die Einsatzzeiten pro Praktikant reduzieren. Das würde zu einer wesentlich effektiveren Laborauslastung führen.

2.4 Sicherung der Energieversorgung

Wie für alle mobilen Geräte ist die Sicherung der benötigten Energieversorgung eine entscheidende Voraussetzung der Betriebsbereitschaft. Wie bei kleinen Modellen gewöhnlich, erfolgt die Bereitstellung durch aufladbare Batterien. Der mobile Roboter muss dafür über eine eigene Akku-Überwachung verfügen bzw. muss nachträglich damit ausgestattet werden. Diese soll den Nutzer über den Ladezustand informieren, so dass dieser bei Bedarf eine Aufladung initialisieren kann. Um ohne zusätzlichen manuellen Eingriff auskommen zu können wird dazu auch eine passende Ladestation benötigt. Diese muss sich auf dem Spielfeld befinden, so dass eine Anfahrt dessen durch Steuerung über das Client-Applet oder vollautomatisch erfolgen kann.

Eine Tiefentladung des Akkus führt zur Verringerung der Lebensdauer und insbesondere der Kapazität. Dies kann bis zum vollständigen Verlust der Speicherkapazität führen. Deshalb muss die Akku-Überwachung eine Tiefentladung wirkungsvoll verhindern. Dieses Modul muss den Roboter dann auch gezwungenermaßen abschalten, falls keine Ladung erfolgt bzw. möglich ist. Eine Wiederinbetriebnahme ist dann natürlich nur durch einen Eingriff vor Ort möglich.

2.5 Anwendung in der Forschung

Ferner sollen die Ergebnisse dieser Arbeit auch dem vom Fachgebiet IHS geplanten „Ad-hoc Lab“ dienen. Diese Arbeit soll dafür wesentliche Vorarbeiten liefern. Dazu gehören die Programmier- und Steuermöglichkeiten über WLAN. Ebenso sind natürlich eine Akku-Überwachung und eine Lademöglichkeit essentiell.

„Das „Ad-hoc Lab“ soll aus mobilen Teilnehmern bestehen, welche durch kleine Roboter mit Wireless LAN-Karten realisiert werden. Sie sollen sich im Praktikumsbereich zufällig bewegen oder einfache, vorab programmierte Wege abfahren. Mit Hilfe von IP-Kameras soll eine Fernüberwachung und Aufzeichnung der Bewegungen realisiert werden. Ein stationärer Rechner mit WLAN Access Point-Funktionalität soll als Access Point für die Kommunikation und als Überwachungs- und Steuerungseinheit für die mobilen Roboter dienen.

Durch die Reduzierung des Senderadius der mobilen Roboter ist die Emulation eines größeren Gebietes möglich. Durch die Reduzierung der Sendeleistung des Access Points ist die Emulation eines Gebietes mit geringer Funkabdeckung möglich. Dies erlaubt die Durchführung von Multi-Hop- (mehrmalige Weiterleitung der Daten unter den Mobilteilnehmern bis der Access Point erreicht wird) und Ad-hoc- (Kommunikation ohne zentralen Access Point bzw. ohne Infrastruktur) Praktikumsversuchen.“[Mit07]

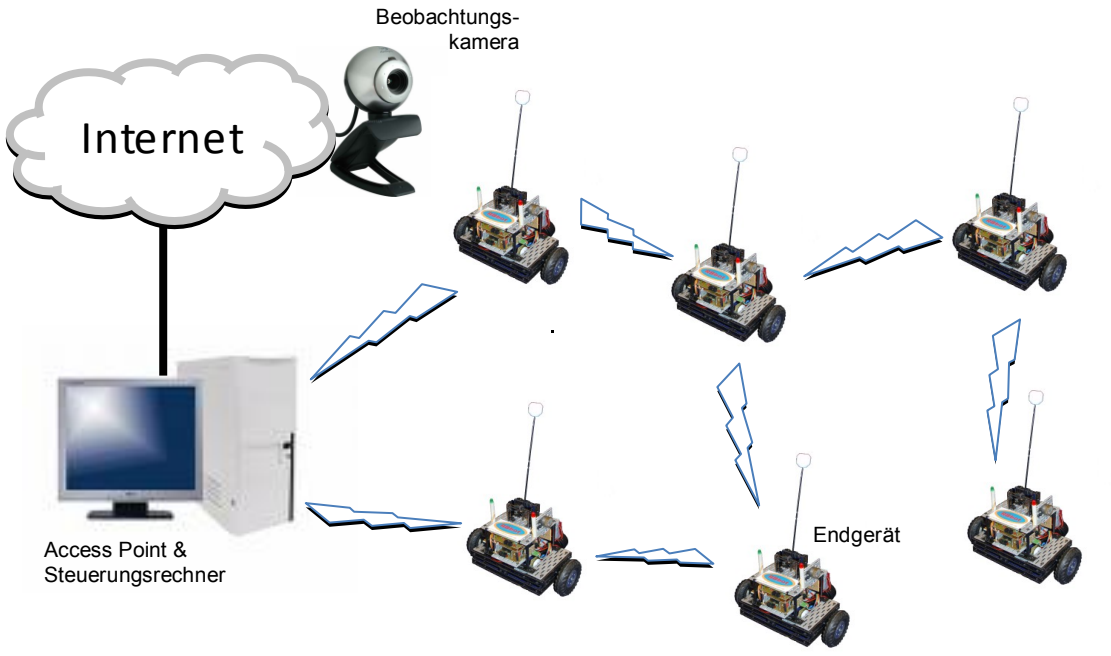


Abbildung 3: Aufbau eines Ad-Hoc Labs

Die Abbildung 3 verdeutlicht den prinzipiell angestrebten Aufbau des Ad-Hoc Labs. Im Vergleich mit der angedachten Gestaltung des Praktikumsplatzes in Abbildung 1 ergeben sich viele Gemeinsamkeiten, welche eine beidseitige Nutzung ermöglichen.

3 State of the Art mobiler Roboter

Für den speziellen Einsatz als mobiler Praktikumsroboter ergeben sich zahlreiche notwendige bzw. wünschenswerte Eigenschaften. Unter anderem sind hierbei zu nennen:

- Robustheit
- Ausstattung, Sensoren, Mobilität
- Verfügbarkeit von WLAN für Integration in „Ad hoc Lab“
- Steuer- und Programmiermöglichkeiten
- Aufwand für Aufbau und Programmierung
- Anforderungen an Größe wegen beschränkten Platzangebotes
- Preis
- Verfügbarkeit, Beschaffung von Ersatzteilen

Da mittlerweile doch recht viele kleine Roboterplattformen erhältlich sind, folgt eine kurze Marktübersicht der verbreitesten Modelle. Eine Vorauswahl wurde dabei anhand der oben genannten Anforderungen getroffen.

3.1 Asuro

Asuro ist ein kleiner, mobiler Roboter. Er wurde am Deutschen Zentrum für Luft- und Raumfahrt (DLR) für die Ausbildung und Lehre entwickelt.

Asuro besitzt zwei Motoren, die unabhängig voneinander angesteuert werden können. Mittels zweier Drehzahlsensoren für die Räder lässt sich die Geschwindigkeit und Position bestimmen. Desweiteren verfügt er über eine optische Linienfolgeeinheit, sechs Kollisionstaster, drei optische Anzeigen sowie eine Infrarot-Kommunikationseinheit, welche die Programmierung und auch eine Fernsteuerung über einen PC ermöglicht. Erweitern lässt sich dieser Bausatz noch mit einem Ultraschallabstandssensor sowie einem LCD-Display.

Als Mikrocontroller wird ein Atmel ATmega8 verwendet. Dafür existieren auch verschiedene Freeware-Entwicklungsumgebungen. Als Programmiersprache wird ANSI-C verwendet, wie bei den meisten Mikrocontrollern üblich.

Der Aufbau ist auch für den Elektronikeinsteiger ohne weiteres durchzuführen. Es werden nur handelsübliche, mit normaler Feinmotorik handhabbare und leicht zu beschaffende Bauteile verwendet.

Abbildung 4: Robotermodell Asuro

ASURO eignet sich daher ausgezeichnet für Hobbybastler, welche den Einstieg in prozessorgesteuerte Schaltungen wagen wollen, für Schüler und Studentenprojekte, Fortbildungen oder Volkshochschulkurse [AREXX_ASURO].

3.2 qfix Crash-Bobby

Neben dem Asuro fiel der Crash-Bobby in die nähere Auswahl. Dieser Bausatz wurde von qfix robotics speziell für die Ausbildung in Schule und Studium entwickelt.

Dieser Roboter wird von zwei Rädern mit unabhängigen Motoren angetrieben und durch ein drittes Rad am hinteren Ende in der Mitte gestützt. Als Energieversorgung stehen zwei Akku-Packs zur Verfügung.

Er wird von einem Atmel ATmega32 Mikrocontroller angesteuert. Vier digitale und vier analoge Eingänge sowie Ausgänge erlauben eine Erweiterung durch externe Senso-

ren. Desweiteren lassen sich über den integrierten I²C-Bus[1] noch weitere Sensoren oder sonstige Elektronik anbinden.

Der Roboter selbst bringt drei Abstandssensoren mit. Als optionale Erweiterungen stehen ein LCD Display, ein Bumper-Set, ein Liniensensor-Set sowie Rad-Encoder zur Geschwindigkeitsmessung zur Verfügung.

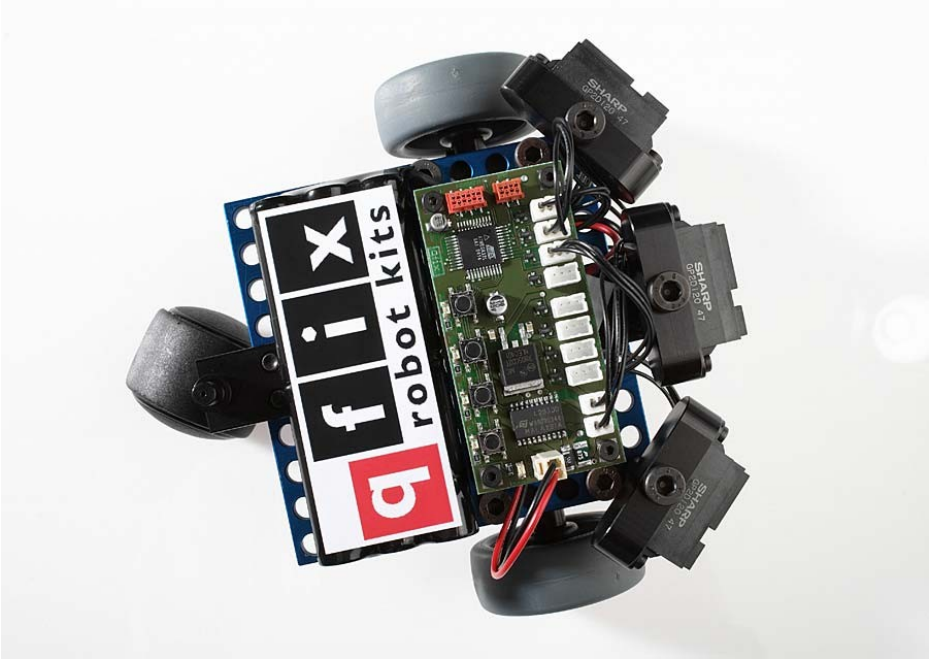


Abbildung 5: Robotermodell qfix Crash-Bobby

Programmiert wird der Crash-Bobby über einen GNU C/C++ Compiler. Zusätzlich steht auch eine grafische Programmierumgebung zur Verfügung. Diese ermöglicht eine C++ Programmierung anhand von Programmablaufplänen.

Ausgeliefert wird dieser Roboter als Modulbausatz mit fertig bestückter Platine [Ras06] [qfi07].

3.3 c't–Bot

Als dritter Kandidat fiel der c't-Bot der Computerzeitschrift c't in die nähere Auswahl. Diese Roboterplattform zeichnet sich durch eine Kombination aus einer Hardware-Plattform und einem Software-Simulator aus, auf dem Bewegungs- und Sensorenvorgänge zuvor getestet werden können, bevor diese dann auf dem Roboter ihren Einsatz finden. Die Plattform bietet, wie die beiden vorher beschriebenen, ebenfalls eine Vielzahl von Schnittstellen digitaler sowie analoger Art. Seine kompakte Bauweise ermöglicht einen hohen Grad an Mobilität. Sensoren nach vorn und unten verhindern eine

[1] I²C (Inter-Integrated Circuit) von Philips entwickelter serieller² Datenbus

Kollision oder den Absturz. Zwei 6 Volt Gleichstrommotoren treiben die beiden Antriebsräder an.

Gesteuert wird der c't-Bot durch einen Atmel ATmega32, der in C unter Windows oder Linux programmiert werden kann.

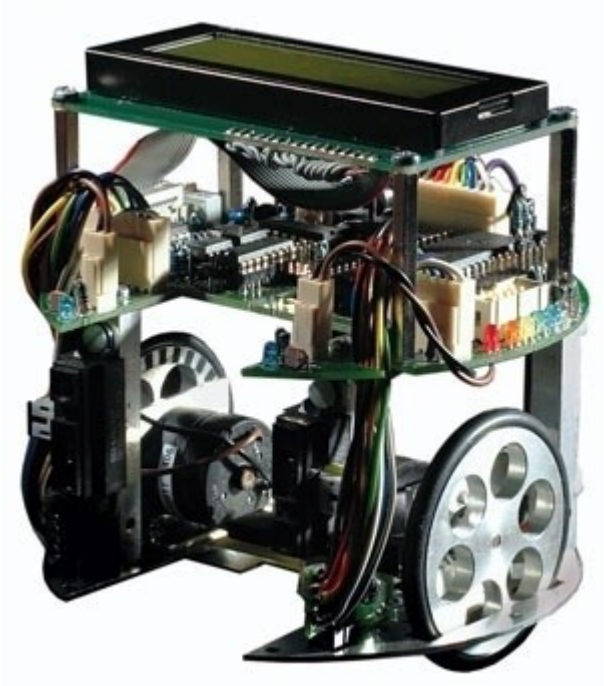


Abbildung 6: Robotermodell c't-Bot

Mit einem zusätzlich erhältlichen Erweiterungsboard stehen dem Bot neben einem SD/MMC Karten Slot ein vollwertiges WLAN-Modul mit eigenem Webserver zur Verfügung. Dieses gestattet eine komfortable Steuerung vom PC aus. Auf dem Webserver lassen sich eigene Webseiten oder Applets für die Steuerung ablegen. Auf einer handelsüblichen Flash-Speicherkarte können zusätzlich Log-Daten sowie Karten der Umgebung aufgezeichnet werden.

Eine gute Dokumentation und eine große Community erleichtern den Zusammenbau, der von Hand vollständig selbst erledigt werden muss, sowie die spätere Programmierung [Ras06] [Hei06].

3.4 Lego Mindstorms TriBot

Mit dem Mindstorm NXT Kit bietet Lego eine interessante Möglichkeit sich einen fahrenden Roboter selbst aufbauen zu können. Dieser dreirädrige Roboter lässt sich mit den im Kit enthaltenen Modulen einfach und schnell aufbauen.

Die gesamte Elektronik befindet sich im NXT–Baustein. NXT bietet mit den zwei eingebauten Prozessoren, einem 32-Bit ARM7 Mikrocontroller sowie einem 8-Bit AVR Mik-

rocontroller, reichlich Rechenleistung. Ein LCD Grafik-Display mit 100x64 Pixeln bietet komfortable Anzeige- und Bedienungsmöglichkeiten.

Über ein integriertes Soundmodul mit Lautsprecher und externem Soundsensor lassen sich auch akustische Ausgaben und Steuerrungen verwirklichen.

Neben einer USB-Schnittstelle verfügt NXT noch über eine Bluetooth-Schnittstelle. Mit dieser ist es nicht nur möglich den TriBot vom PC aus neu zu programmieren, sondern ihn auch von anderen Bluetooth-Geräten zu steuern.

Abbildung 7: Robotermodell Lego Mindstorms TriBot

Die Motoren sowie die Sensoren lassen sich einfach an diesem Modul anschließen. Es verfügt dazu über drei Ausgangsports zur Ansteuerung von Motoren und vier Eingangsports zum Auslesen der Sensoren. Im Mindstorm NXT Kit stehen dazu ein Berührungssensoren (Taster), ein Soundsensor, ein Lichtsensor und ein Ultraschallentfernungssensor zur Verfügung. Optional erhältlich sind ein Kompass, ein Beschleunigungssensor sowie ein Gyroskop-Sensor.

Programmiert wird NXT standardmäßig mit LabVIEW. Dieses graphische Programmiersystem arbeitet mit verschachtelbaren Funktionsblöcken. Datenflüsse sind durch Verbindungslinien gekennzeichnet. Entwicklungsumgebungen von Drittanbietern erlauben aber auch eine Programmierung in gebräuchlicheren Sprachen wie C/C++ oder Java [Lego].

3.5 Lynxmotion 4WD2

Dieses Modell ist noch kein Roboter, kann aber dazu ausgebaut werden. Deshalb soll eine solche Variante nicht von vornherein unbeachtet bleiben, auch wenn hierfür ein hoher Entwicklungsaufwand aufzubringen wäre.

Abbildung 8: Plattform Lynxmotion 4WD2

Der 4WD2 ist eine äußerst robuste und solide Plattform zum Aufbau von autonomen Robotern etc. Sie besitzt keinerlei Elektronik, lediglich der Antrieb (die Räder und Motoren) ist bereits integriert. Im Inneren des Gehäuses bietet sich aber noch viel Platz um eigene Elektronik unterzubringen. Durch die flache Bauweise (Räder überragen Chassis) gibt es keine feste Ober- bzw. Unterseite. Überschläge sind somit kein Problem. Daher ist diese Plattform auch sehr gut für Outdoor-Anwendungen geeignet.

Mit einer maximalen Geschwindigkeit von 116 cm/s ist er sehr schnell, bleibt nur die Frage, ob eine zu entwickelnde Steuerelektronik und Software mit dieser Geschwindigkeit Schritt halten könnte.

Die Traglast von 2.5 kg dürfte für die meisten Fälle mehr als genug sein. Wenn man die Überschlagfunktion nicht nutzen möchte bietet sich auch auf der Oberfläche viel Platz für Elektronik, Sensoren und Anzeigen oder sonstige Aufbauten [Act_4WD2].

3.6 Robby RP6

Interessant ist ebenso diese von AREXX Engineering entwickelte und von Conrad vertriebene Roboterplattform. Dies ist das Nachfolgermodell des komplett überarbeiteten erfolgreichen „C-Control Robby RP5“ und wird als fertig montiertes Modell ausgeliefert.

Wie auch der Vorgänger benutzt Robbie RP6 das fast baugleiche, robuste Raupenfahrgestell. Die zwei unabhängigen Antriebsmotoren sowie die Elektronik werden von sechs AA-Standardbatterien versorgt, welche unterhalb der Hauptplatine untergebracht sind.

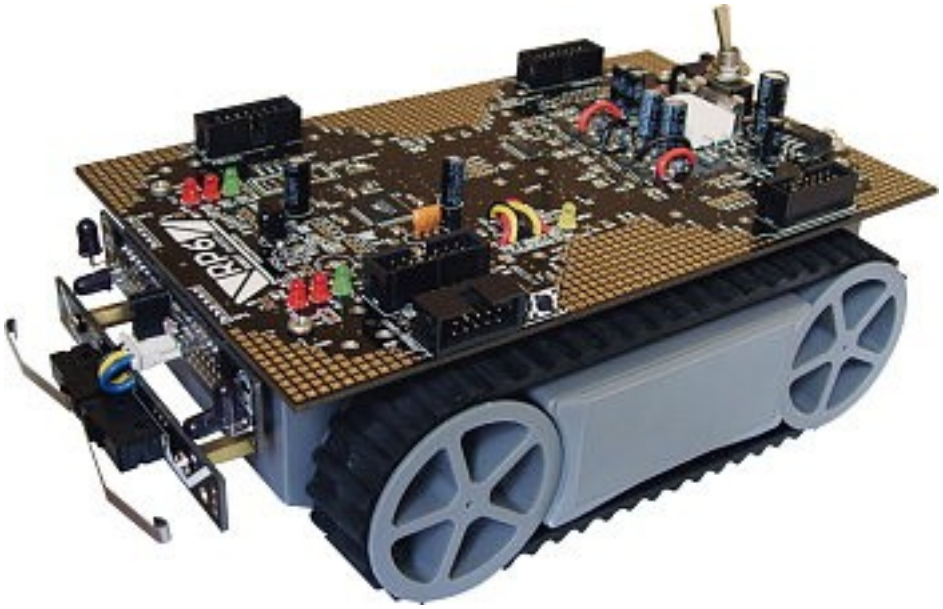


Abbildung 9: Robotermodell Robby PR6

Er besitzt eine Stoßstange mit zwei Tastsensoren um Kollisionen erkennen zu können. Zwei Lichtsensoren sowie zwei IR-Antikollisionssensoren sind ebenfalls am vorderen Ende angebracht. Desweiteren besitzt der Robby einen IR-Sender sowie einen dazu passenden IR-Fernbedienungssensor. Damit ist dann sowohl eine Steuerung mittels handelsüblicher Fernbedienung als auch eine Kommunikation mehrerer RP6 untereinander möglich. Durch die vielen Erweiterungsflächen und Stecksockel lassen sich auch noch viele weitere Sensoren über den verwendeten I²C-Bus anschließen. Hervorzuheben sind auch die hochauflösenden Drehzahlgeber, die für eine präzise Geschwindigkeitsregelung und Wegmessung unabdingbar sind. Desweiteren verfügt dieses Modell über eine Überwachung der Akkuspannung sowie der Motorstromaufnahme. Sechs LEDs und ein zusätzlich angebotenes LCD-Text Display erleichtern die Kommunikation und Überwachung und ermöglichen ein einfaches Debugging in laufendem Betrieb.

Gesteuert wird dieser Roboter von einem Atmel ATmega32. Zur Programmierung wird das Open Source Tool avr-gcc empfohlen. Dazu werden auch umfangreiche Bibliotheken und viele Beispielprogramme geliefert. Die am PC entwickelten Programme werden über ein beigelegtes USB-Interface auf den Mikrocontroller übertragen. Eine große Internet-Community sowie zahlreiche Foren erleichtern den Einstieg und bieten Gelegenheit für einen intensiven Erfahrungsaustausch [AREXX_PR6].

3.7 Surveyor SRV-1

Der SRV-1 wurde als ferngesteuerte Webcam bzw. als selbstnavigierender autonomer Roboter entwickelt. Mehrere SRV1 können von einer einzigen Basisstation betrieben werden. Die Roboterbasis ist aus stabilem Aluminium aufgebaut. Er besitzt Kettenantrieb, der von zwei Getriebemotoren angetrieben wird. Mit dem beigelegten 7.2 V 2 Ah Li-Ionen Akku ergeben sich Betriebszeiten von bis zu 4 h. Die maximale Geschwindigkeit beträgt 40 cm/s.

Abbildung 10: Robotermodell Surveyor SRV-1

Die eingebaute Kamera überträgt die Bilder zu der Java basierten Steuer-Software. Die Bildübertragung und Steuerung läuft über eine Zigbee-Funkübertragung, die eine Reichweite von bis zu 100 m innerhalb von Gebäuden ermöglicht. Der SRV1 hat ein Array von Infrarotsendern und Empfängern zur Abstandserkennung und zur Kommunikation mit anderen Robotern.

Programmiert wird dieser Bot in ANSI-C. Die Software für autonomes Verhalten, inklusive Bewegungserkennung, Objektverfolgung, Hindernisvermeidung sowie einfache Navigation ist noch in Entwicklung und wird dann als Open Source (GPL) zur Verfügung stehen [Act_SRV].

3.8 iRobot Create

Mit dem iRobot Create steht eine robuste Plattform für eigene Roboter zur Verfügung. Grundlage ist der autonome Staubsaugerroboter iRobot Roomba, welcher seit Jahren, vor allem in den USA, vertrieben wird. Dadurch wurde die verwendete Mechanik dieses Modells schon langjährig getestet. Dieser Roboter beinhaltet alle zur häuslichen Navigation notwendigen Sensoren. Es besitzt zehn eingebaute Demo-Programme, welche die Möglichkeiten des Einsatzes kurz vorstellen. Die Plattform erlaubt eine Verhaltensprogrammierung ohne Kenntnisse der Low-Level Programmierung.

Abbildung 11: Roboterplattform iRobot Create

Angesteuert wird dieser Bot über die Softwareschnittstelle „iRobot Create's Open Interface". Diese beinhaltet komfortable Kommandos für Bewegung, Sound, Displaydarstellung, Sensorüberwachung usw. Dafür kann entweder eine serielle Verbindung mit dem PC genutzt werden oder das iRobot Command Modul (siehe Abbildung 11: grünes Steckmodul). Dieses mit einem Atmel ATmega168 ausgestattete Command Module erlaubt es auch weitere Sensoren anzubauen oder andere Erweiterungen anzusteuern.

Platz für verschiedene Erweiterungen, ob zusätzliche Leiterplatten oder mechanische Aufbauten, bietet die „Cargo Bay". Eine Ladestation ermöglicht das automatische Laden des Roboters. Mit Hilfe der virtuellen Wände lässt sich der Arbeitsbereich begrenzen. Ein zusätzliches Erweiterungsmodul gestattet auch eine Steuerung über Bluetooth. Allerdings blockiert dieses Modul dann den Erweiterungsport, an dem sonst das Command Modul angeschlossen ist [iRo07].

3.9 Auswahl und Wertung

In der Tabelle sind noch einmal die wichtigsten Merkmale der hier vorgestellten Systeme dargestellt.

Tabelle 1: Übersicht marktüblicher mobiler Kleinroboter

Name	Größe (mm)	Mikrocontroller	Funk / WLAN	Sensoren	Link	Preis (EUR)
Asuro	117 x 122	ATmega8	IR / n	Linie	www.arexx.com	50
Crash-Bobby	150 x 170	ATmega32	n / n	Abstand, Linie	www.qfix.de	259
c't-Bot mit Erweiterung	120 x 120	ATmega644	IR / ja	Abstand, Abgrund, Licht, Linie	www.heise.de/ct/projekte/ct-bot/	420
Lego Mindstorm TriBot	k.A.	ARM7	Bluetooth / n	Abstand, Tastsensor, Licht, Linie, Sound	http://mindstorms.lego.com	279
Lynxmotion 4WD2	300 x 300	-	- / -	nur Chassis, Motor und Räder	www.active-robots.com	230
Robby RP6	172 x 128	ATmega32	IR / n	Abstand, Licht, Tastsensor	www.conrad.de	129
Surveyor SRV-1	120 x 100	ARM7	Zigbee / n	Abstand, IR, Webcam	www.active-robots.com	492
iRobot Create Premium Package	330 x 330	ATmega168 (Command M.)	IR / n	Abgrund, Abstand, Berührungssensor	www.irobot.com	253

Als wichtigste Kriterien stellten sich der erforderliche Platzbedarf sowie die Verfügbarkeit von WLAN heraus. Leider konnte nur ein Modell damit dienen. Im Prinzip ließen sich auch die anderen Modelle noch mit einem teilweise selbst zu entwickelnden WLAN-Modul nachrüsten, was aber zu erheblich höherem zusätzlichen Aufwand führen würde. Deshalb fiel die Wahl zu Gunsten des c't-Bots, da er neben dem WLAN-Modul auch über eine sehr gute Grundausstattung verfügt, aber auch noch Raum für eigene Erweiterungen lässt.

4 Beschreibung des c’t–Bot Roboters

Der Roboter wurde von der Zeitschrift c’t des Heise Zeitschriften Verlages im Jahre 2006 entwickelt und vorgestellt. Er hat seitdem viele begeisterte Fans und Anwender gefunden. Einige beteiligen sich auch an der weiteren Programmierung und Verbesserung der Software. Diese wird zentral in einem SVN-Repository[2] des Verlags verwaltet [Heise_Wiki]. Dort befindet sich auch ein Wiki mit Anleitungen, Fragen und Antworten sowie Dokumentationen zum Bot.

4.1 Überblick

Der Roboter baut auf einer runden Grundplatte mit 120 mm Durchmesser auf. An Ihr sind alle Komponenten stabil mit Aluminiumträgern befestigt. Zwei Motoren ermöglichen die Fortbewegung. Die Achse geht durch den Mittelpunkt der Grundplatte. Dadurch kann sich der Bot auf der Stelle drehen. Ein Gleitpin verhindert das Abkippen des Bots nach hinten. Die Masseverteilung vor allem des Akku-Packs sorgt dafür, dass der Bot nicht nach vorne kippt.

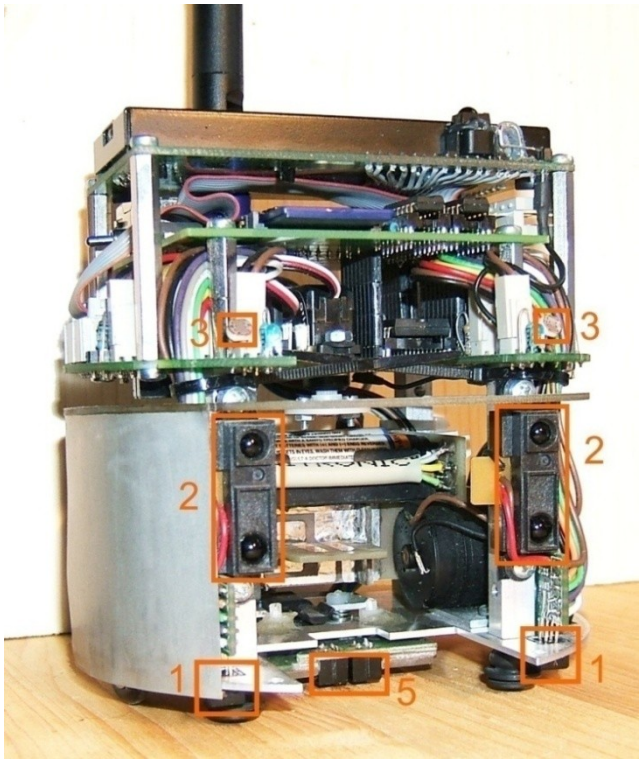


Abbildung 12: Sensoren des c't-Bot
1 – Abgrundsensoren; 2 – Abstandssensoren; 3 – Lichtsensoren; 5 - Liniensensoren

[2] Subversion Projektarchiv; zentrale Versionsverwaltung von Dateien und Verzeichnissen anhand einer Revisionsnummer

Vorn befindet sich ein rechteckiger Ausschnitt, mit dem Gegenstände eingefangen und transportiert werden können. Eine Transportklappe verhindert ein unbeabsichtigtes entweichen eingesammelter Gegenstände.

Die beiden 6 V Gleichstrommotoren (2619-006SR) von der Firma FTB Faulhaber sind stark genug um auch zusätzliche Aufbauten bewegen zu können und ermöglichen eine maximale Geschwindigkeit von 450 mm/s.

Auf der Innenseite der Räder sind Encoder-Scheiben mit abwechselnd schwarzen und weißen Feldern angebracht. Diese werden mit zwei Reflexlichtschranken abgetastet. Damit lässt sich die Drehgeschwindigkeit der Räder ermitteln, der Geradeauslauf des Bots kontrollieren sowie die Position ermitteln. Zusätzlich ist unten ein optischer Maussensor angebracht, mit dem sich noch höher aufgelöste Geschwindigkeitsmessungen und Positionsbestimmungen durchführen lassen.

Als Prozessor dient in dieser Version ein Atmel ATmega644 Prozessor. Dieser wird mit 16 MHz getaktet und besitzt 64 KiB[3] Flash Programmspeicher, 4 KiB Ram und 2 KiB EEPROM. In der Originalversion wird ein ATmega32 benutzt der, im Gegensatz zum Verwendeten, nur die halbe Speichergröße besitzt.

Zur besseren Navigation verfügt der c't-Bot über eine Vielzahl weiterer Sensoren. Auf der Unterseite angebrachte Reflexlichtschranken ermöglichen eine Abgrunderkennung (1) und die Fahrt entlang einer dunklen Linie (5) (siehe Abbildung 12). Zwei Abstandssensoren (2) mit einem Erfassungsbereich von 10 bis 80 cm ermöglichen eine Kollisionsvermeidung mit Hindernissen oder auch eine Kartierung der Umgebung. Zwei Fotosensoren (3) ermöglichen eine lichtgesteuerte Navigation. So lassen sich z. B. lichtsuchende Roboter bauen [Sch07].

Darüber hinaus verfügt der Bot über einen IR-Fernbedienungssensor, so dass er sich auch mit einer normalen TV-Fernbedienung steuern lässt. Neben acht Status-LEDs ermöglicht ein 4x20 Zeichen Text-Display die Programmüberwachung sowie Ausgabe verschiedener Sensorwerte. Das WLAN–Modul [Lantronix] ermöglicht eine komfortable Steuerung und Überwachung auf einem PC. In Tabelle 12 im Anhang E.1 werden dazu die technischen Daten aufgeführt.

[3] Kibibyte; 2^{10} Byte = 1024 Byte

4.2 Antrieb

Für eine „intelligente“ Antriebsregelung sind genaue Angaben über den Antrieb insbesondere der verwendeten Motoren erforderlich. Deshalb bedarf es einer ausführlicheren Abhandlung dieses Aspektes.

Wie bereits erwähnt, wird der Bot von zwei 6 V Gleichstrommotoren von FTB Faulhaber angetrieben. Diese besitzen eisenlose Rotoren, welche das Trägheitsmoment der Motoren deutlich reduzieren. Ein eingebautes Getriebe untersetzt die Drehzahl des Motors und liefert dadurch ein höheres Drehmoment. Mit der angegebenen Drehzahl von 151 U/min erreicht der Roboter bei einem Raddurchmesser von 57 mm eine maximale Geschwindigkeit von 45 cm/s.

Die beiden Motoren sind über eine integrierte H-Brücke[4] angeschlossen. So lassen sich leicht Rechts- und Linkslauf gewährleisten. Die Drehzahlregelung bzw. die Einstellung der dem Motor zugeführten effektiven Spannung erfolgt durch Pulsweitenmodulation. Somit ist eine quasi stufenlose, verlustarme Anpassung der Motorspannung möglich.

Die wichtigsten Parameter sind im Anhang E.2 aufgeführt (Siehe auch Kapitel 6.9).

[4] Elektronische Schaltung bestehend aus vier Schaltern. Diese kann zur Ansteuerung und Wahl der Drehrichtung von Gleichstrommotoren verwendet werden.

4.3 Sensorik

Die richtige Auswertung der Sensoren ist essentiell für die erfolgreiche Programmierung. Deshalb widmet sich dieser Abschnitt der ausführlichen Erläuterung aller Sensoren. Damit soll es den Studenten ermöglicht werden, auf eventuelle Schwachpunkte und Besonderheiten der Sensoren bzw. der Anbringung einzugehen. Wenn nicht anders beschrieben, erfolgt die Digitalisierung der analogen Messwerte im internen 10 Bit ADC[5] des Atmel Mikrocontrollers. Somit steht ein Wertebereich von 0 bis 1023 für die digitalisierten Sensorsignale zur Verfügung.

Tabelle 2: Übersicht der Sensoren des c't-Bot

Sensoren	Typ	Reichweite & Bemerkung	Verwendung
Abgrundsensoren	Reflexlichtschranke CNY70	1..6 mm Position unten - rechts und links vorn	Abgrunderkennung
Liniensensoren	Reflexlichtschranke CNY70	1..6 mm Position unten Mitte	Linienfolger
Abstandssensoren	Sharp GP2D12 Entfernungsmesser	10 .. 80 cm Position rechts und links vorn	Hindernisvermeidung
Lichtsensoren	lichtempfindlicher Widerstand	Position oben - rechts und links vorn	lichtabhängiges Verhalten
Maussensor	Agilent ADNS-2610 optische Maus	400 DPI, max. Geschwindigkeit 30 cm/s	Positions- und Geschwindigkeitsbestimmung
Rad-Encoder	Reflexlichtschranke CNY70	3 mm Auflösung	Positions- und Geschwindigkeitsbestimmung
Sensor Klappe	Reflexlichtschranke CNY70	nur 0/1	Transportklappe geschlossen?
Sensor Ladebucht	IR-Lichtschranke	nur 0/1	Gegenstand in Ladebucht?
IR-Empfänger	TSOP 34836	ca. 4 m	Fernsteuerung

[5] Analog-to-Digital-Converter

4.3.1 Abgrundsensoren

Zwei Abgrundsensoren sind in den vorderen Ecken angebracht. Sie sind dazu gedacht, Abgründe, Tischkanten, Treppenstufen oder ähnliches zu erkennen.

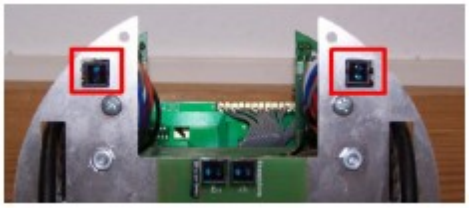


Abbildung 13: Position der Abgrundsensoren

Verwendet wurden hier zwei Reflexlichtschranken CNY70 von Vishay Semiconductors. Der Sensor besteht aus einem Licht-Emitter und einem Licht–Detektor (verpackt in einem kompakten Gehäuse). Die Leuchtdiode sendet IR-Licht mit einer Wellenlänge von 950 nm aus. Das an einer Oberfläche reflektierte Licht wird vom eingebauten Fototransistor empfangen und in einen lichtabhängigen Strom gewandelt.

Der Kollektorstrom hängt dabei stark vom Reflexionsgrad und Abstand der Oberfläche ab. Gute Ergebnisse sind bei Abständen zwischen 0,3 mm und 8 mm möglich.

Der Detektor besitzt zwar einen optischen Filter, aber der Einfluss von Streulicht ist besonders bei höheren Abständen nicht zu vernachlässigen. So ist mitunter bei einer hohen Umgebungsbeleuchtung eine präzise Detektion von Kanten nicht möglich.

Da sich die Infrarot LEDs auf dem Bot per Software an- und ausschalten lassen (siehe Schaltplan *ENA_KANTLED*) wäre es möglich, dieses Problem mit einer weiteren Messung zu umgehen. Mit einer Messung ohne aktivierte LED könnte die Umgebungshelligkeit bestimmt werden. Mit einer zweiten Messung mit aktivierter LED könnte dann die Veränderung bestimmt werden. Als Schwellenwert würde somit nicht ein absoluter Wert sondern ein differenzieller oder prozentualer Wert dienen.

4.3.2 Abstandssensoren

An der Vorderseite des Roboters befinden sich zwei Distanz-Mess-Sensoren von Sharp (GP2D12) [SHARP]. Durch die Verwendung von moduliertem IR-Licht arbeiten diese Sensoren sehr robust, unabhängig von einfallender Beleuchtung und Oberflächenbeschaffenheit. Diese liefern ein, dem Abstand entsprechendes, analoges Ausgangssignal. Dieses ist allerdings nicht linear (siehe Abbildung 14).

Deshalb wird dieses Signal im c't-Bot mithilfe von LookUp-Tabellen linearisiert und in ein Abstandsmaß umgerechnet. Allerdings ist der Sensor für Abstände unter 10 cm nicht geeignet, da, wie in Abbildung 14 zu erkennen ist, die Ausgangsspannung bei kleiner werdenden Abständen sinkt.

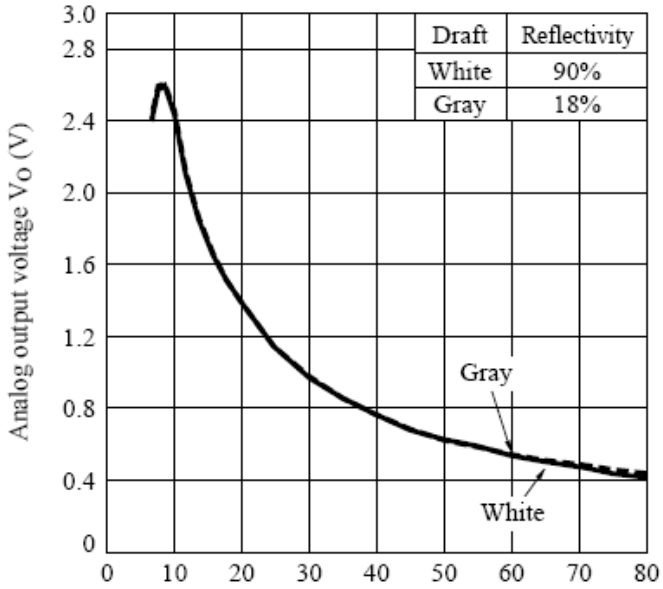


Abbildung 14: Kennlinie des Abstandssensors

Wie in Abbildung 15 zu erkennen ist, besteht in der Breite nur ein recht schmaler Erfassungsbereich. So beträgt z. B. bei einem Abstand von 40 cm die Sichtkegelbreite nur 4 cm. Gegenstände außerhalb dieses Bereiches werden nicht erfasst. Beim Geradeausfahren könnte dies zu Problemen führen. So ist es möglich, dass der Bot leicht schräg auf eine Wand zufährt, ohne diese detektieren zu können.

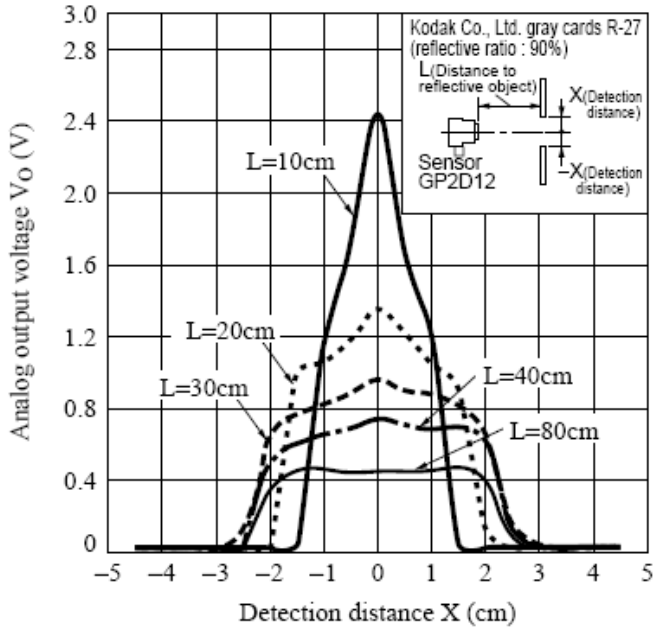


Abbildung 15: Erfassungsbereich des Abstandssensors

Die Abbildungen dieses Kapitels sind dem Datenblatt des Sensors entnommen worden. [SHARP]

4.3.3 Lichtsensoren

Vorn auf der Oberseite der Hauptplatine befinden sich rechts und links jeweils ein Fotowiderstand. Mit ihnen lässt sich die Umgebungshelligkeit messen. Außerdem kann sich der Bot mithilfe der Lichtsensoren auf die Suche nach Lichtquellen begeben, bzw. lassen sich lichtabhängige Verhalten[6] erstellen. In der Praxis hat sich aber gezeigt, dass ein recht hoher Kontrast zur Umgebung notwendig ist, um Lichtquellen detektieren zu können. Dies ist normalerweise nur in abgedunkelten bzw. künstlich beleuchteten Umgebungen zu erreichen.

4.3.4 Maussensor

Auf der Unterseite ist ein optischer Maussensor montiert. Dieser arbeitet nach dem Prinzip einer optischen Computermaus. Eine LED sendet ein Licht zur Beleuchtung der abzutastenden Oberfläche. Eine kleine schwarz/weiß Kamera nimmt ein 19 x 19 Pixel Bild auf. Die Elektronik in dem Sensor ermittelt anhand der Verschiebung markanter Punkte die Bewegung in x- und y-Richtung. Dadurch, dass der Maussensor nicht in Achsmitte liegt, lassen sich Translation und Rotation des Bots bestimmen. Ein Vergleich mit den Geschwindigkeitswerten der Rad-Encoder kann ein Schlupf der Räder erkennen.

Da der Abstand des Sensors sehr exakt stimmen muss, war hier eine kleine Modifikation erforderlich (siehe Abschnitt 5.1.2). Der Erfolg der Maßnahmen lässt sich gut anhand des Sensorbildes zeigen, welches sich im Applet darstellen lässt. Ein gutes Bild sollte starke Kontraste zeigen. Allerdings wird dazu auch eine geeignete Unterlage benötigt, wie z. B. ein Mauspad für eine optische Maus, ein Geldschein oder ein sehr feiner Ausdruck. Das Blickfeld ist bei 400 DPI etwas klein (es sind demnach nur 1,2 mm x 1,2 mm).

Die maximal zulässige Geschwindigkeit ist mit rund 300 mm/s angegeben, was unterhalb der Robotergeschwindigkeit liegt. Dies muss bei der Programmierung berücksichtigt werden!

4.3.5 Liniensensoren

Auf der Maussensorplatine sind zusätzlich zwei Liniensensoren angebracht. Verwendung fanden auch hier die bereits beschriebenen Reflexlichtschranken CNY70. Allerdings sind diese Sensoren deutlich dichter zur Oberfläche montiert, was die Erkennung positiv beeinflusst. Die Erkennung von dunklen Linien erfolgt in der verwendeten Software anhand eines Schwellenwertes. Der Roboter fährt im „Linienfolger-Modus“ (siehe Anhang D.2) mit dem linken Sensor auf der dunklen Linie und mit dem Rechten daneben. Die Linienbreite sollte demnach mindestens der Breite eines Sensors entsprechen, also mindestens 7 mm. Eventuell ist der Schwellenwert der Erkennung anzupas-

[6] In Anlehnung zur Bezeichnung in der c't werden mit „Verhalten“ die Funktionen und Routinen zur Koordination der Bewegung beschrieben

sen (*LINE_SENSE* in *bot-local.h*). Gute Erfahrungen konnten mit der Verwendung von 15 mm breitem, schwarzem Klebeband gemacht werden. Dieses lässt sich auch problemlos entfernen und wiederverwenden.

4.3.6 Rad-Encoder

Zur Bewegungserkennung sind an der Innenseite der Räder Inkremental-Scheiben mit 30 schwarzen und 30 hellen Feldern angebracht. Diese werden mit den bereits erwähnten Reflexlichtschranken abgetastet -- werden hier aber hardwareseitig digitalisiert. Der Schwellenwert ist dabei durch einen Spannungsteiler fest eingestellt. Bei ungünstiger Wahl des Spannungsteilers ergeben sich unterschiedliche Intervallzeiten. Dies bedeutet, dass bei konstanter Drehzahl die high und low Zeiten eines Intervalls unterschiedlich sind, obwohl die Felder der Inkremental-Scheiben alle gleich groß sind. (Siehe auch Kapitel 5.1.1).

Die Rad-Encoder bieten eine maximale Auflösung von 3 mm bei Verwendung beider Flanken zur Detektion. Dies ist vor allem bei langsamen Geschwindigkeiten recht wenig. In der Praxis hat sich aber dennoch gezeigt, dass sie gegenüber dem Maussensor verlässlichere Werte liefern. Deshalb arbeitet die implementierte Geschwindigkeitsregelung mit den Werten der Rad-Encoder.

4.3.7 Sensor-Klappe

Eine Klappe wird zum Schließen des Transportfaches verwendet. Die Position dieser Klappe wird mit einer Reflexlichtschranke am Tragarm überprüft. Das Ausgangssignal wird ebenfalls hardwareseitig digitalisiert. Dieser Zustand wird im Applet unter „*Door-Sens*“ angezeigt. Der Ausgangswert ist 0, wenn die Klappe vollständig geschlossen ist. Der Ausgangswert ist 1 bei allen anderen Öffnungswinkeln, egal ob ganz offen oder nur fünf mm geöffnet. Nutzen ließe sich dieser Wert z. B. zum Erkennen, ob der Schließvorgang erfolgreich war oder ob eventuell ein Gegenstand den Schließvorgang behindert hat.

4.3.8 Sensor-Ladebucht

In der Ladebucht befindet sich ca. 8 mm vom Ende der Bucht in 17 mm Höhe eine Lichtschranke. Diese ist als Lichtschranke mit getrenntem Emitter und Detektor aufgebaut. Auf der einen Seite der Ladebucht befindet sich eine IR-Diode und auf der anderen Seite ein IR-Empfänger. Der Zustand wird im Applet unter „*Trans*“ angezeigt. Wenn sich kein Gegenstand in der Ladebucht befindet (bzw. Lichtschranke nicht unterbrochen ist), so ist dieser Wert 0. Verwendet wird diese Information z. B. beim Andocken an die Ladestation. Nützlich ist der Sensor auch für die Detektion von eingefangenen Gegenständen.

4.3.9 IR-Empfänger

Zur Steuerung des c't-Bots über eine übliche Standard-Fernbedienung gibt es einen passenden Empfänger. Verwendung findet hier ein TSOP 34836. Dieser detektiert Infrarot-Licht, welches mit 36 kHz moduliert ist. Dies wird in dem RC5-Standard der meisten TV-Fernbedienungen verwendet. Hierzu müssen die zu verwendenden Codes der Fernbedienung im Quellcode für den Bot eingetragen werden. Eine kleine Auswahl ist bereits implementiert und muss nur noch aktiviert werden (siehe *rc5-codes.h*). Für diesen Praktikumsplatz und für die Steuerung des Roboters mittels Applet wird der Sensor nicht verwendet. Die im Applet integrierte Fernbedienungskonsole generiert und übermittelt dem Bot die hinterlegten Codes, die dann genauso wie die über den Sensor empfangenen Signale behandelt und abgearbeitet werden können.

4.4 Verfügbare Software

Neben der Hardware wurde auch die Software für den c't-Bot vom Heise Verlag entwickelt. Die Weiterentwicklung wird allerdings von vielen Nutzern vorangetrieben. Der Code dafür wird im SVN-Repository des Verlages zentral verwaltet und zum Download angeboten [Heise_Wiki]. Der gesamte Quellcode ist unter der „General Public License" veröffentlicht und zur freien Verwendung freigegeben.

Zum einen wird der Code für die grundlegende Programmierung des Mikrocontrollers zur Verfügung gestellt. Diese übernimmt vor allem die Low-Level Programmierung der Hardware, das Auslesen der Sensoren, die Ansteuerung der Motoren, die Kommunikation usw. Der Anwender kann sich somit auf die High-Level Programmierung, dem Schreiben von Verhaltensroutinen, beschränken. Zum anderen wird eine Simulationsumgebung (c't-Sim) zum Testen und Simulieren der Programme auf dem PC angeboten.

Die Software für den c't-Bot ist in ANSI-C geschrieben, einige Funktionen wurden auch direkt in Assembler implementiert. Der Simulator ist im Gegensatz dazu eine reine Java–Anwendung.

Als Entwicklungsumgebung wurde Eclipse gewählt. Diese plattformunabhängige und frei erhältliche Umgebung bietet viele Vorteile. Mit ihr lassen sich nicht nur Programme für den Mikrocontroller erzeugen, sondern auch für den PC. Die PC-Version wird dabei für die Verwendung des Simulators benötigt. Dadurch lassen sich alle Programmteile in einer Entwicklungsumgebung bearbeiten. Desweiteren gestattet Eclipse den komfortablen Import aus dem SVN-Repository.

4.4.1 Grundstruktur der Software

Die grundlegende Struktur lässt sich am besten anhand des Programmablaufplanes der Main-Routine nachvollziehen.

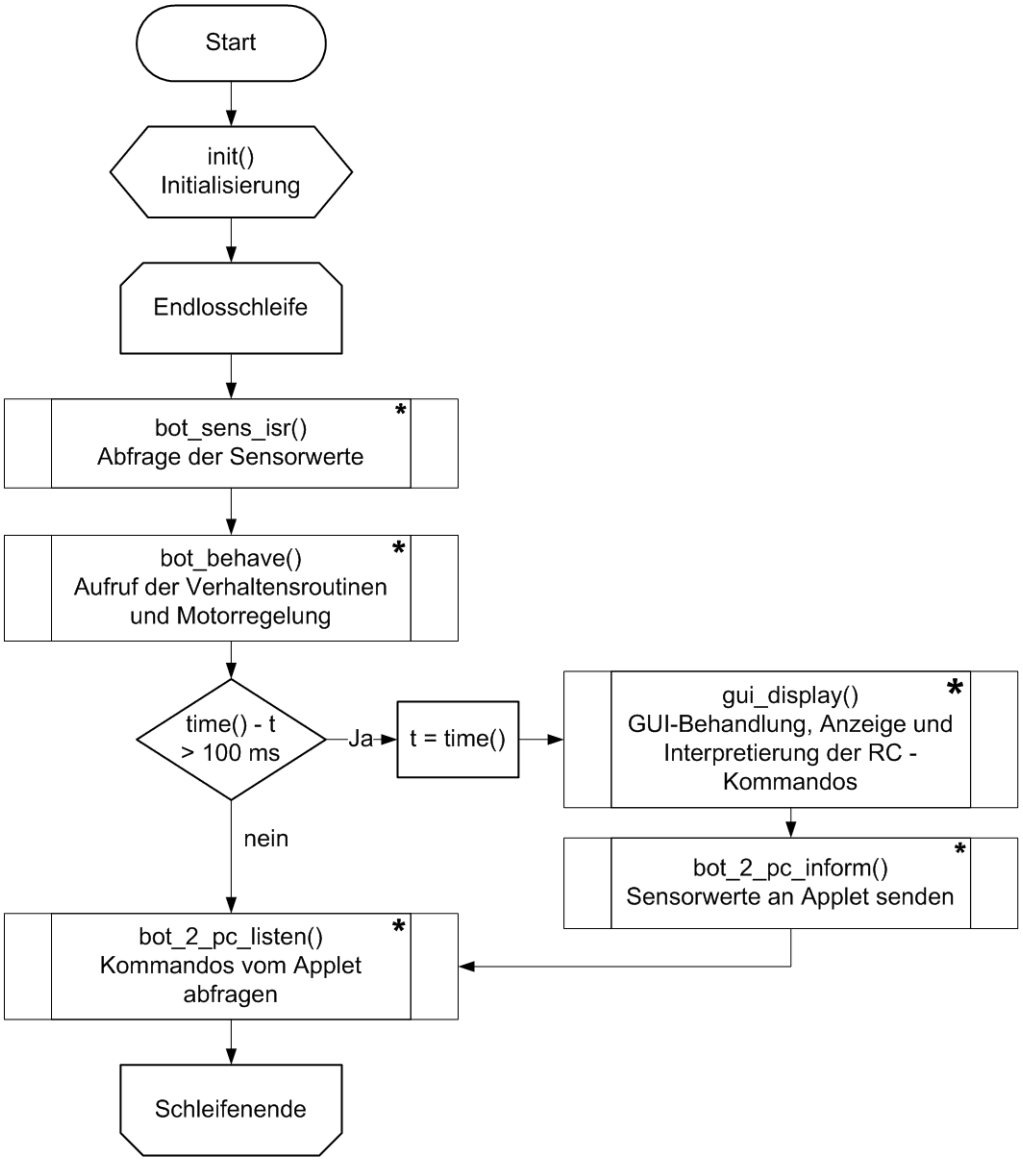


Abbildung 16: Programmablaufplan der Main-Routine

Nach dem Start werden alle Hard- und Softwarekomponenten initialisiert. Die eigentliche Programmausführung startet danach in der Endlosschleife. Zuerst erfolgen die Abfrage der Sensorwerte, die Abarbeitung der Verhaltensroutinen und das Setzen der Aktoren (wie Motor und Klappe). Danach erfolgt die Kommunikation mit dem Nutzer. Die Anzeige des Displays sowie die Übermittlung der Sensorwerte und Zustände an das Test-Applet erfolgen aus Zeitgründen nur aller 100 ms. Nach der Abfrage und Auswertung möglicher Kommandos vom Nutzer beginnt der Schleifendurchlauf von

vorn. Eigene Verhalten werden in *bot_behave* eingebunden und von dort aus aufgerufen.

Damit soll der prinzipielle Ablauf des Programmes deutlich werden. Dies soll den Einstieg in die Programmierung dieses Roboters erleichtern, da der Umfang des Codes und die Anzahl der Dateien, für den begrenzten Programmspeicherplatz der MCU[7], doch beachtlich sind.

4.4.2 Verhaltensprogrammierung

Die Hauptaufgabe der Studenten wäre es, eigene Verhaltensroutinen zu programmieren, zu simulieren und am realen Bot zu testen. Zur leichten Integration unterschiedlichster Verhaltensmuster wurde dazu eine Verwaltungsstruktur geschaffen. Die einzelnen Routinen werden, geordnet nach vorgegebener Priorität, in einer verketteten Liste eingetragen. Die Abarbeitung erfolgt dann der Reihe nach.

Im jeweiligen Verhalten werden die Geschwindigkeitswünsche für die Motoren angegeben. Dazu sind die globalen Variablen *speedWishLeft* sowie *speedWishRight* zu belegen. Durch Setzten der Variablen *faktorWishLeft* und *faktorWishRight* sind auch relative Geschwindigkeitsangaben möglich. Die konkrete Sollgeschwindigkeit wird dann durch eine multiplikative Verknüpfung bestimmt.

So wird die Verhaltensliste nach und nach abgearbeitet, bis ein Verhalten die Geschwindigkeit direkt setzten möchte. Diese Geschwindigkeitsangaben für die beiden Räder werden dann jeweils mit den relativen Wünschen der vorherigen Verhalten multipliziert und der Motorsteuerung übergeben. Somit kann sich der Multiplikator auch aus den relativen Angaben mehrerer Verhaltensroutinen zusammensetzten.

Die Verhalten werden in der Regel als Zustandsautomat implementiert. Ein Wechsel des Zustandes erfolgt z. B. durch einfache Bedingungsabfragen. Die Verhaltensroutinen werden periodisch aufgerufen. Die Abarbeitungszeit des Prozessors für einen Durchlauf sollte demnach sehr kurz gehalten werden, insbesondere dürfen keine Wartepausen eingelegt werden. Hier erfolgt nur die Anweisung, was der Bot im nächsten Intervall tun soll.

Nachfolgendes Beispiel soll zeigen, wie sich eigene Verhaltensroutinen erstellen lassen. Es stellt dabei nochmals die verschiedenen Geschwindigkeitsangaben dar. Außerdem demonstriert es den Aufruf anderer Verhaltensroutinen. Somit lassen sich dann auch komplexe hierarchische Verhalten realisieren.

[7] MCU (engl. Microcontroller unit); Mikrocontroller

```
void bot_example_behaviour(Behaviour_t *data) {

  switch (state) {
    case START:   // direktes Setzen einer Wunschgeschwindigkeit
      speedWishLeft = BOT_SPEED_NORMAL;
      speedWishRight = BOT_SPEED_NORMAL;
      if (sensDistL < 250 || sensDistR < 250) state=STATE1;
    break;

    case STATE1:     // relatives Setzen einer Wunschgeschwindigkeit
      faktorWishLeft = 0.5;
      faktorWishRight = 0.5;
      state=STATE2; // if (??) state=??;
    break;

    case STATE2: // Aufruf vorhandener Verhaltensmuster
      bot_turn( data, 180);
      state=END;
    break;

    case END:
    default:  // beenden der Ausführung
      state=START;
      return_from_behaviour(data);
    break;
  }
}
```

Dieses Beispiel lässt den Roboter solange geradeausfahren, bis ein Hindernis im Abstand von weniger als 250 mm detektiert wird. Anschließend wird eine Drehung um 180° ausgeführt.

In *STATE1* wird ohne Abfrage ein Zustandswechsel vorgenommen. Somit ist dieser Zustand nur für einen Durchlauf aktiv. Von der Ausführung würde man daher nichts mitbekommen.

4.4.3 Simulationsumgebung c't Sim

Zum Test der Software, vor allem selbstentwickelter Verhaltensroutinen, dient die Simulationsumgebung c't Sim [Hei06].

Sie ist dafür ausgelegt, den für den c't-Bot geschriebenen Steuerungscode testen zu können. Der simulierte Bot erhält von der Simulationsumgebung die Werte der einzelnen Sensoren. Diese ermittelt unter Auswertung der Steuerung der virtuellen Motoren die Position in der virtuellen Welt, die wiederum Basis für die dem Bot gelieferten Sensorwerte sind. Kollidiert der Bot mit einem Hindernis, bleibt der simulierte Bot stehen.

Der Simulator ist komplett in Java geschrieben, um den Einsatz plattformunabhängig auf verschiedenen Betriebssystemen zu ermöglichen. Benutzt wird zusätzlich die Java3D Bibliothek zur Darstellung der Welt und der Modellierung. Hinweise zur Installati-

on und Benutzung sowie den kompletten Quellcode dazu sind im Wiki [Heise_Wiki] verfügbar.

Für die Nutzung des Simulators stehen im Code für den Roboter zahlreiche Precompiler–Anweisungen, um zwischen dem Code für den Mikrocontroller und dem Code für den PC auswählen zu können. Zur Nutzung des Simulators muss das Projekt mit anderen Einstellungen neu kompiliert werden. Dazu ist in Eclipse die Konfiguration Debug-W32 bzw. Debug-Linux auszuwählen. Die Kompilation erzeugt ein auf dem PC ausführbares Programm. Die Kommunikation zwischen Bot-Programm und Simulator erfolgt intern über eine TCP/IP Verbindung. Der Simulator beinhaltet dafür einen TCP–Client und auch einen einfachen TCP–Server.

An dieser Stelle sei nur die grundlegende Vorgehensweise angedeutet:

- den Simulator c't Sim starten
- in Eclipse Projekt/Build Configurations/Set Active/Debug-W32 bzw. Debug-Linux auswählen
- Projekt neu kompilieren (es wird eine ct-Bot.exe bzw. ct-Bot.elf Datei erzeugt)
- ct-Bot.exe bzw. ct-Bot.elf starten
- im Simulator Play anwählen und der virtuelle Bot beginnt zu arbeiten

Aufgrund der simulierten Hardware sind nicht alle Funktionen verfügbar. Auch werden für die Simulation andere Fernbedienungscodes verwendet. Das sollte im Prinzip nicht weiter stören, die Bedienung ist dadurch allerdings etwas eingeschränkt.

Nach erfolgreicher Simulation kann die Praxistauglichkeit der Programmierung auf den realen c't-Bot getestet werden. Dabei ist zu beachten, dass der reale Roboter keine genauen Sensorwerte liefert und Bewegungsbefehle nicht so exakt ausgeführt werden wie in der Simulation. Eine gewisse Fehlertoleranz ist also erforderlich.

5 Hardwaremodifikationen und Erweiterungen

Die speziellen Anwendungen und die damit verbundenen Anforderungen an den Praktikumsplatz, insbesondere die Sicherstellung der Energieversorgung, haben es erforderlich gemacht, nicht nur die vorhandene Hardware anzupassen und zu verändern, sondern auch die bestehende Hardware zu ergänzen.

5.1 Modifikationen

Aufgrund der nicht ausgereiften Fahrweise und vor allem wegen den störungsanfälligen Sensoren wurden die nachfolgend beschriebenen Veränderungen durchgeführt. Diese sollen die Zuverlässigkeit der Sensoren steigern.

5.1.1 Rad-Encoder

Die CNY70-Sensoren der Rad-Encoder tasten die hell/dunkel Flächen der Encoder-Scheiben ab. Der Phototransistor reagiert darauf mit einem lichtabhängigen Strom, der durch einen Spannungsteiler in eine abgreifbare Spannung umgesetzt wird. Diese wird im Schmitt-Trigger in einen binären Wert gewandelt. Zusätzlich wurde noch eine Entprellung per Software realisiert. Trotz dieser Maßnahmen kam es beim langsamen Drehen des Rades und bei einem hell/dunkel Übergang manchmal zu mehreren Schaltvorgängen je Übergang. Eigentlich sollte dies durch die Hysterese des Schmitt-Triggers unterbunden werden.

Um diese Signalfehler zu unterdrücken, wurde nachträglich eine RC-Tiefpass-Filterung eingebaut. Dazu wurden jeweils 100nF parallel zu den Widerständen (*R17* und *R18*) geschaltet. Dies ließ sich durch Auflöten von SMD 0805 Bauteilen auf der Unterseite der Platine erreichen. Messungen mit einem Oszilloskop bestätigten das gewünschte Tiefpass-Verhalten. Der c't-Bot konnte nun die Encoder-Schritte zuverlässig detektieren.

Durch Anpassen der Vorwiderstände zu den IR-Dioden des Sensors konnte der Stromverbrauch um ca. 19 mA reduziert werden. Dazu wurden *R23* und *R24* durch 330 Ω Widerstände ersetzt (siehe Schaltplan im Anhang F.3).

Desweiteren wurde der Spannungsteiler so angepasst, dass einfallendes Streulicht weniger Einfluss auf die Sensoren hat, d. h. der Schaltpunkt in Richtung hell verschoben. Dazu wurden die Widerstände *R19* und *R20* durch 9,1 kΩ Typen ersetzt.

5.1.2 Maussensorplatine

Wie dem Datenblatt zu entnehmen ist, liegt der optimale Abstand des Maussensors im Bereich von 2,3 mm bis 2,5 mm (siehe [SHARP]). Bei anderen Abständen liefert er keine befriedigenden Messergebnisse.

Die mit auf der Platine angebrachten Liniensensoren überragen den Maussensor noch um 2 mm. Bei einer Montage der Maussensorplatine im richtigen Abstand zur abzutastenden Oberfläche beträgt daher die Bodenfreiheit des Roboters nur noch 0,5 mm. Durch eine Modifikation wurde dieses Manko behoben. Der vordere Teil der Platine, auf dem sich die Liniensensoren befinden, wurde abgetrennt. Dieser Teil wird, um eine Platinenstärke versetzt, wieder an der Platine mit Epoxidharz befestigt. Eine zusätzliche kleine Leiterplatte stabilisiert den Aufbau. Die dabei unterbrochenen Leiterbahnen wurden mit dünnen Drähten wieder verbunden. Wenn auch diese Lösung für den Nachbau nur bedingt zu empfehlen ist, wird nun eine Bodenfreiheit von über 2 mm erreicht (siehe Abbildung 17).

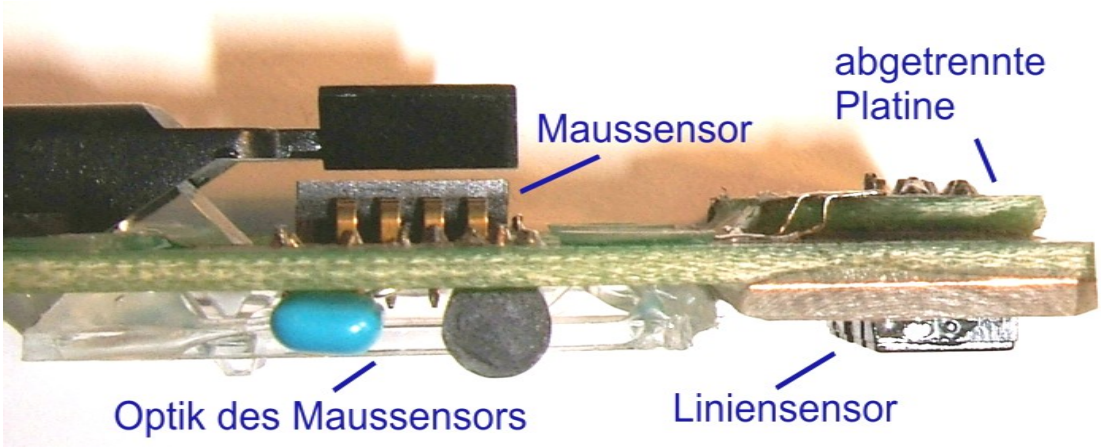


Abbildung 17: Veränderungen an der Maussensorplatine

Die Messergebnisse des Maussensors konnten somit wesentlich verbessert werden, ohne die Bodenfreiheit oder die Funktion der Liniensensoren zu beschränken. Für den weiteren Ausbau ist eventuell ein Neuentwurf dieser Leiterplatte in Erwägung zu ziehen, um die Bauteile in einer passenden Höhe montieren zu können.

Zum leichteren Justieren des Abstandes wurde die Leiterplatte nicht direkt mit Abstandshaltern montiert, sondern diese durch Federn ersetzt. Außerdem wurden die Muttern auf der Oberseite der Grundplatte des Bots angeklebt. So kann die Justage des Abstandes einfach und schnell durch Drehen an den vier Befestigungsschrauben erfolgen.

5.1.3 Abstandssensoren

An den Abstandssensoren waren nur geringfügige Änderungen notwendig, welche auch im Wiki zum c't-Bot empfohlen wurden [Heise_Wiki]. Das zyklische Messen der Sensoren verursachte ein periodisches Einbrechen der Versorgungsspannung. Dies liegt vor allem an dem recht hohen Widerstand des, in der Versorgungsspannungsleitung liegenden und zum An- bzw. Ausschalten der Sensoren verwendeten, MOSFETs *TR1*. Durch Pufferung mit einem 330 µF Kondensator zwischen *Pin 3* und *Pin 10* von *ST8* auf der Hauptplatine, sowie jeweils einen 100 nF Kondensator direkt am Sensor, konnte dieser Effekt stark reduziert werden (siehe Schaltplan im Anhang F.3).

Außerdem wurde der linke Sensor um 180 Grad gedreht, so dass jetzt beide Sensoren in gleicher Ausrichtung montiert sind. Dadurch werden Messunterschiede durch unsymmetrisches Verhalten vermieden. Für den Aufbau weiterer c't-Bots ist es empfehlenswert, den MOSFET durch einen Geeigneteren zu ersetzten oder aber auf die Abschaltbarkeit dieser Sensoren zu verzichten. In diesem Fall kann der MOSFET *TR1* durch eine Drahtbrücke zwischen Drain und Source ersetzt werden.

5.2 Erweiterungen

Wie bereits erwähnt, ist die Sicherstellung der Energieversorgung speziell für den ferngesteuerten Einsatz essentiell. Der c't-Bot verfügt standardmäßig nur über eine rudimentäre Spannungsüberwachung. Weder ein Ladegerät noch eine Ladestation sind für den Roboter erhältlich. Daher mussten hierfür spezielle Erweiterungen vorgenommen werden.

5.2.1 Akku–Halterung

Ursprünglich wurde der Akkupack nur provisorisch mit Klettband an der unteren Trägerplatte fixiert. Mitunter veränderte sich durch abrupte Fahrmanöver die Lage und beeinflusste dadurch auch das Fahrverhalten negativ. Es musste hier eine bessere Lösung gefunden werden. Desweiteren sollte es möglich sein, den Akkupack ohne großen Aufwand an ein Ladegerät anschließen und aufladen zu können.

Der Akkupack wurde mit Leiterplatten teilweise ummantelt. Diese dienen als Träger sowie beinhalten die benötigten elektrischen Verbindungen. Auf der Hinterseite des Akkupacks befinden sich die Steckverbinder zur Aufnahme der Akku-Überwachungsplatine. Auf der Vorderseite, in die Ladebucht hineinragend, befinden sich die Kontaktflächen für die Verbindung mit der Ladestation (siehe Abbildung 18 und 20).

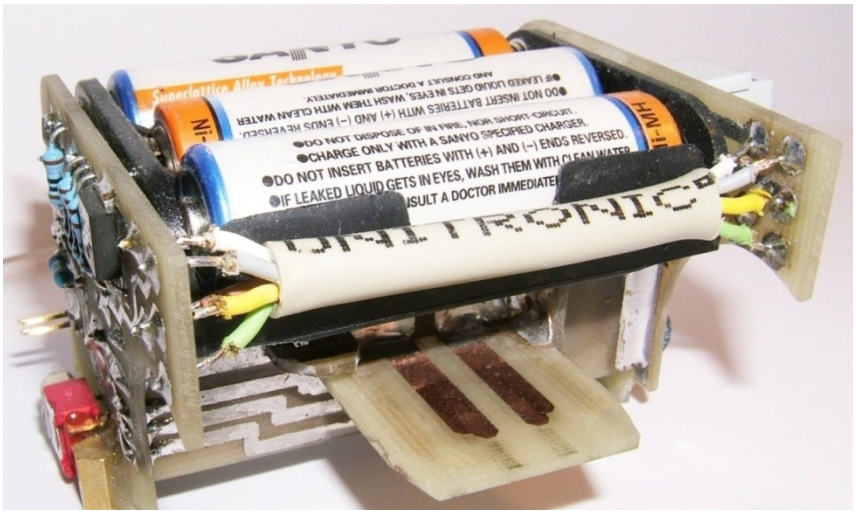


Abbildung 18: Akku-Halterung mit Konnektor, Ansicht oben/vorn

Abbildung 19 stellt den Schaltplan der Akku-Halterung dar. Jumper *JP1* und *JP2* dienen als Steckverbinder zur Aufnahme des Akku-Überwachungsmoduls. *JP1* führt, neben der Versorgungsspannung an *Pin 6,* den Spannungsabgriff über den Strommesswiderständen *R1* bis *R6 (Pin 1).* Die Anschlüsse der *Pins 2* bis *4* werden über *JP3* zur Ladestation weitergeleitet. *JP2* führt die zur Messung verwendeten Spannungsabgriffe aller Zellen. *Pin 1* und *Pin 6 (CHARGER)* sind die äußeren Kontakte des Akkupacks. Zum Laden werden diese über den Konnektor mit der Ladestation verbunden. Die Anschlüsse der einzelnen Zellen an *JP2* sind im Schaltplan nicht eingezeichnet, wurden aber im Layout der Leiterplatten berücksichtigt.

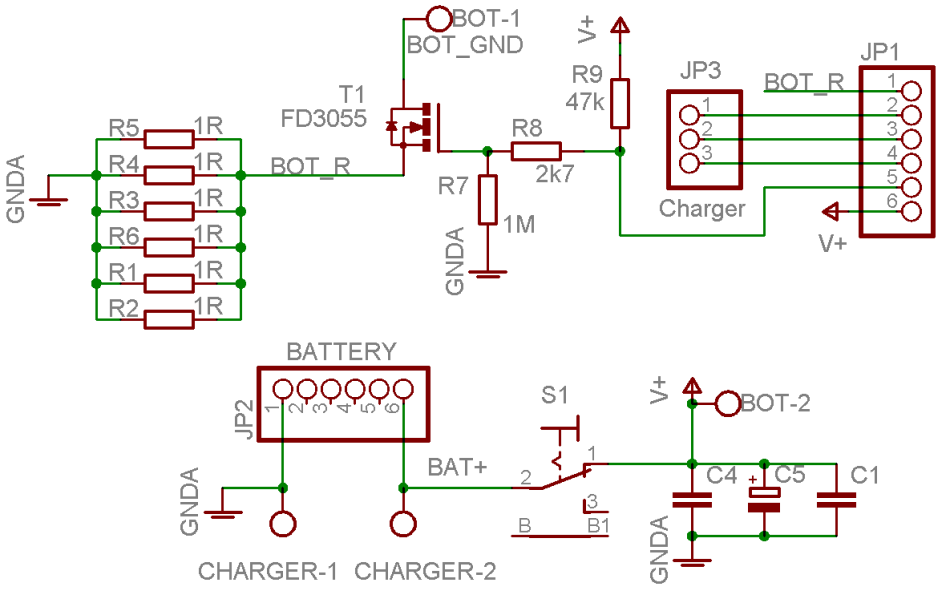


Abbildung 19: Schaltplan der Akku-Halterung

Der Schalter *S1* ist der Hauptschalter des Bots. Der MOSFET *T1* stellt einen zusätzlichen Notschalter dar, der von der Akku-Überwachung gesteuert wird. Standardmäßig wird er durch *R9* immer voll durchgeschaltet. Dadurch kann der Bot auch ohne aufgesteckte Akku-Überwachung betrieben werden. Zum Abschalten durch die Akku-Überwachung wird *Pin 5* von *JP1* auf Masse *(GNDA)* gesetzt.

Die Widerstände *R1* bis *R6* dienen zur Strommessung. Der Spannungsabfall lässt sich über das Ohmsche Gesetz berechnen. Demnach ist $U = RI = \frac{1\,\mathrm{V}}{6\,\mathrm{A}} I$. Bei einem Strom von 600 mA ergibt sich somit ein Spannungsabfall von 100 mV. Dieser ist groß genug um ihn vernünftig messen zu können, aber auch noch klein genug, um nicht die dem Bot zu Verfügung stehende Spannung unnötig zu reduzieren.

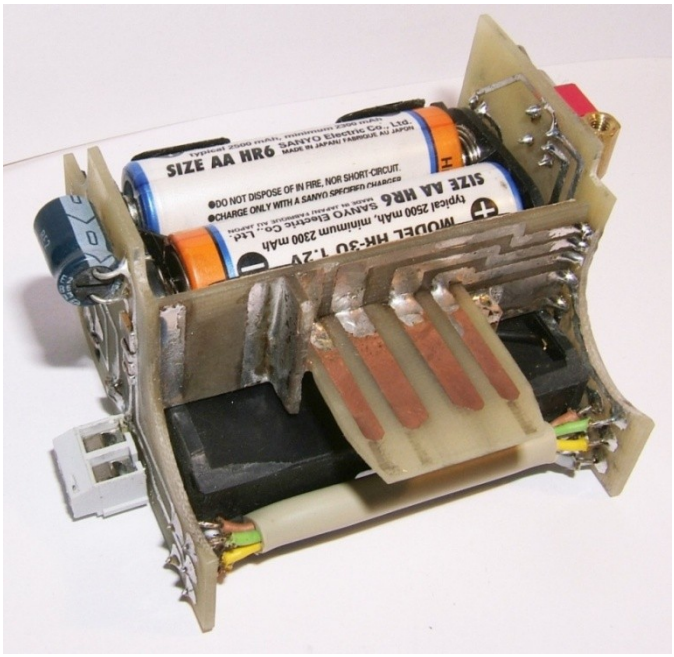


Abbildung 20: Akkupack mit Konnektor, Ansicht unten/vorn

Dieser modifizierte Akkupack lässt sich durch Schraubverbindungen stabil am Roboter befestigen. Weiterhin bietet er viele Zusatzfunktionen. Neben den Kontakten zur Verbindung mit der Ladestation gestattet er die Aufnahme eines Akku-Überwachungsmoduls, ermöglicht die Messung der einzelnen Zellenspannungen und der Stromentnahme des Bots und beinhaltet den, zur Abschaltung des Roboters benötigten, MOSFET.

5.2.2 Akku-Überwachung

Als spezielle Erweiterung für die Sicherung der Energieversorgung des Roboters wurde eine Akku-Überwachung entwickelt und realisiert.

Ein ATmega48 übernimmt die Messung, Logik und Kommunikation. Die Verbindung mit dem Mikrocontroller des Bots, dem ATmega644, erfolgt mit dem I²C–Bus. Der ATmega48 arbeitet dabei als Slave-Device (siehe auch Abbildung 22).

Abbildung 21 stellt den Schaltplan dar. Die Jumper *JP1* und *JP2* sind die Steckverbinder zur Akku-Halterung. Die an *JP1* anliegenden Batteriespannungen werden über Spannungsteiler reduziert und im Mikrocontroller digitalisiert. Dazu verfügt der ATmega48 über acht Analog-Digital-Wandler mit einer Auflösung von 10 Bit. Als Vergleichsspannung wird die interne 1,1 V Referenzspannung verwendet. Deshalb war der Spannungsteiler so auszulegen, dass alle Spannungen sicher unterhalb dieses Wertes bleiben.

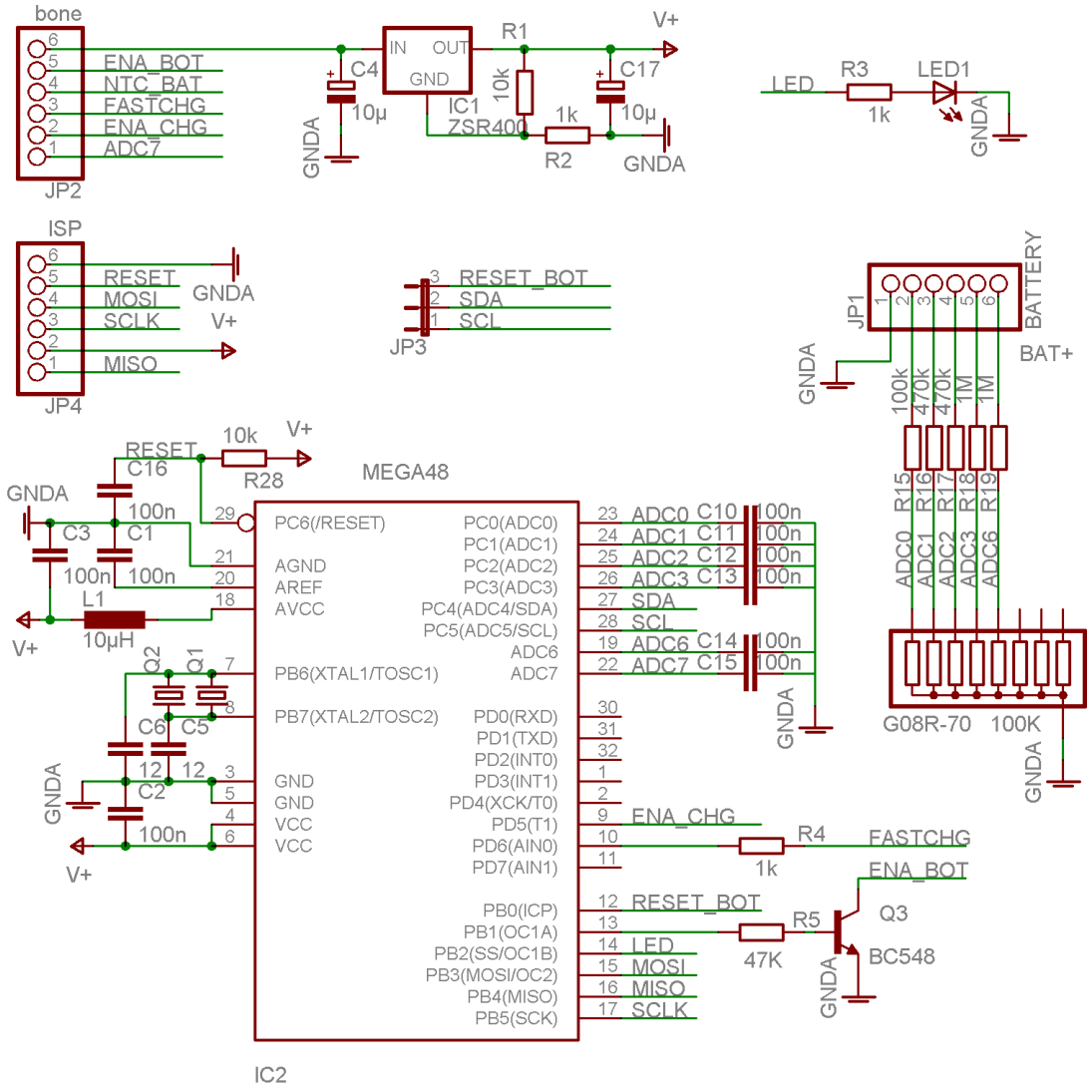


Abbildung 21: Schaltplan der Akku-Überwachung

JP4 ist der benötigte ISP-Programmieranschluss. Aus Platzgründen wurde nicht der standardmäßige zehnpolige Steckverbinder verwendet. Zum Flashen des Controllers wurde ein zusätzliches Adapterkabel angefertigt.

Der Reset–Ausgang ist vorgesehen, um den Bot zum Neustart zu zwingen, ohne dabei das WLAN–Modul auszuschalten. Ein Reset beider Komponenten ist durch Aus- und Einschalten des MOSFETs *T1* der Akku-Halterung möglich (siehe Abbildung 19).

Eine LED dient als optische Kontrolle der Programmausführung dieses Moduls. Sie blinkt in Abhängigkeit von der Batteriespannung bei niedriger Spannung mit ca. 1 Hz, bei höherer Spannung mit entsprechend niedrigerer Frequenz.

Der Mikrocontroller überwacht die Spannungen jeder Ni-MH Akkuzelle. Bei Unterschreiten einer Schwellenspannung von 0,8 V einer Zelle wird der Bot zum Schutz des Akkus komplett abgeschaltet.

Desweiteren wird die Stromentnahme des Bots gemessen. Typische Ströme sind rund 250 mA im Ruhezustand, 650 mA mit WLAN-Modul und maximal 900 mA im Fahrzustand mit eingeschaltetem WLAN-Modul. Die Auflösung beträgt dabei rund 6 mA.

Sämtliche Spannungswerte und die Stromaufnahme lassen sich über den I²C–Bus *(JP3 SDA und SCL),* vom Programm des c't-Bots aus, abfragen (siehe Kapitel 6.4). Im Anhang D.3 befindet sich dafür die Belegung des Ausgangspuffers.

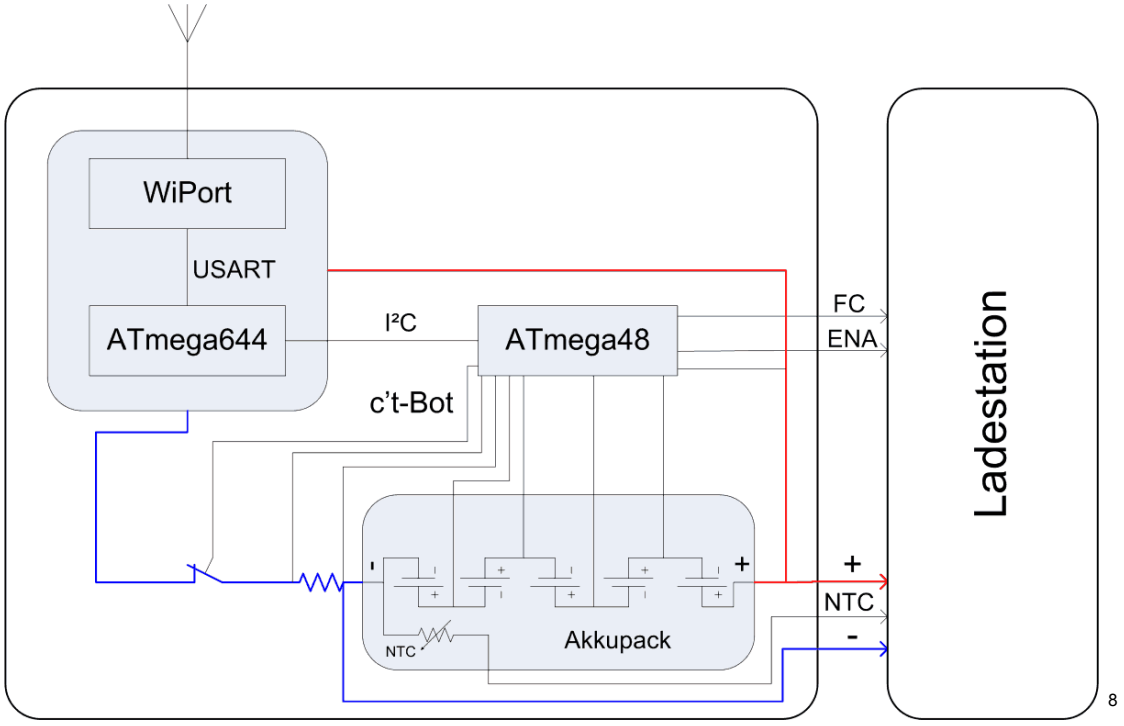


Abbildung 22: Grundstruktur der Akku-Überwachung

[8] USART -- Universal Synchronous/Asynchronous Receiver Transmitter

Dieses Modul versorgt den Roboter mit den für die Sicherstellung der Energieversorgung benötigten Daten. Durch die Abschaltung des Bots bei Unterspannung wird die Tiefentladung des Akkupacks vermieden.

Das Modul selbst benötigt rund 7 mA. Um nach erfolgter Abschaltung eine Tiefentladung durch die Akku-Überwachung zu vermeiden, sollte der Roboter innerhalb von zwei bis drei Tagen manuell (am Hauptschalter) abgeschaltet oder geladen werden.

Überlegenswert wäre auch eine Programmüberwachung des c't-Bots. Dazu könnte der ATmega48 als externer Watchdog[9] verwendet werden. Wenn das Programm des c't-Bots nicht regelmäßig den Zähler über den I²C-Bus zurücksetzt, wird der Roboter neu gestartet. Die Anweisung zum Zurücksetzen des Zählers könnte in der Main-Schleife des Bots erfolgen.

5.2.3 Ladestation

Die Konzeption und Umsetzung der Ladestation stellt einen wesentlichen Punkt dieser Arbeit dar, da keine vergleichbaren Ansätze existieren.

Als Anhaltspunkte für die entwickelte Ladeschaltung dienten ein Vorschlag der Fachzeitschrift „Elektor“ sowie die im Datenblatt empfohlene Beschaltung des verwendeten Ladecontrollers MAX713 [Tav07][Max02].

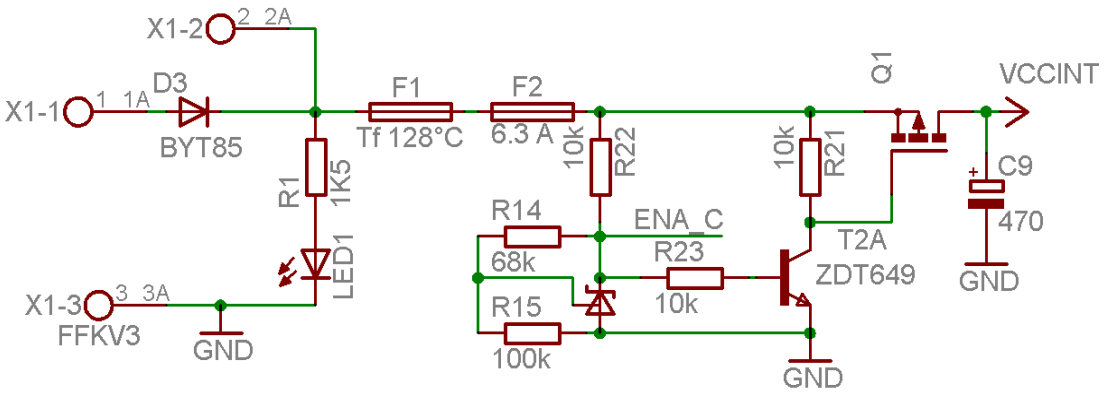


Abbildung 23: Schaltplan Ladestation Teil 1

Abbildung 23 zeigt die Versorgungsspannungseingänge sowie Absicherungen der Schaltung. Als Spannungsquelle dient ein 12 V Gleichspannungsnetzteil, welches an der Leiterbahnklemme *X1* angeschlossen wird. In der Plusleitung des Anschlusses *X1-1* ist die Diode *D3* als Verpolungsschutz eingebaut. Es wird empfohlen, diesen Eingang zu verwenden sofern die Anschlussspannung ausreichend hoch ist (siehe Anhang B.3). Der Kühlkörper an *D3* ist für höhere Ströme zu klein dimensioniert. Für La-

[9] Engl. Wachhund; Komponente der MCU, welche die Programmausführung überwacht. Wenn innerhalb einer bestimmten Zeit das Programm den Watchdog nicht neu startet wird ein Reset der MCU veranlasst.

deströme über 3 A sollte deshalb *X1-2* für den Anschluss des positiven Gleichspannungspols verwendet werden. *LED1* signalisiert den korrekten Anschluss und die Funktion des Netzteils.

Neben einer Schmelzsicherung, die den maximalen Strom begrenzt, verfügt die Ladestation noch über eine Temperatursicherung. Beide sind nur für den äußersten Notfall gedacht, um größeren Schaden an Akku und Ladegerät abzuwenden.

Mittels *ENA_C*, welches über *JP7* zum c't-Bot geleitet wird, besteht die Möglichkeit, den Kontakt mit der betriebsbereiten Ladestation vom Bot bzw. der Akku-Überwachung erkennen zu können. Mittels einer Shunt-Diode wird dazu die Spannung auf 4,2 V reduziert. Dieses Signal wird in der Akku-Überwachung direkt auf einen I/O-Pin des ATmega48 geleitet. Durch setzen dieses Pins auf Masse (Minus der Batterie), kann die Ladestation ausgeschaltet werden. Damit fließt kein Basisstrom, *T2A* ist somit offen. Das Gate des P-Kanal MOSFETs *Q1* wird über *R21* mit voller Versorgungsspannung geladen und *Q1* sperrt. Ansonsten fliest über *R23* ein kleiner Basisstrom und *T2A* zieht die Gate-Spannung des *Q1* auf Masse. *Q1* wird dadurch geschlossen und über *VCCINT* steht der Schaltung dann die Eingangsspannung zur Verfügung.

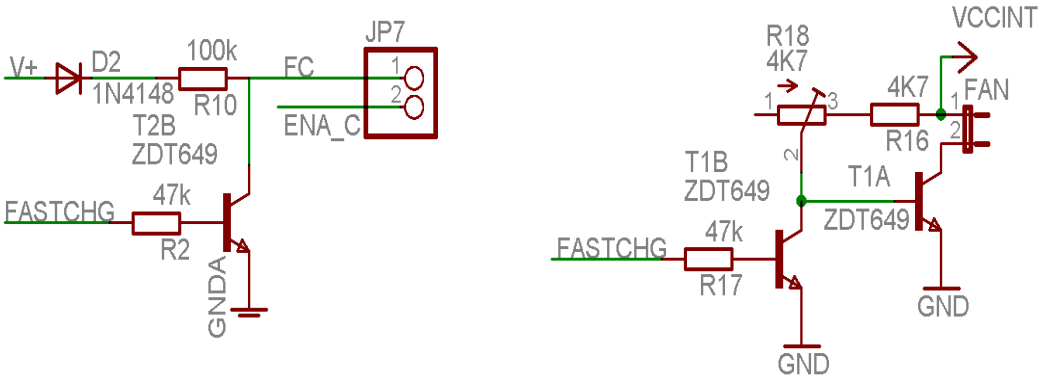


Abbildung 24: Schaltplan Ladestation Teil 2

Abbildung 24 stellt die Ansteuerung des Lüfters sowie mit *JP7* die Zustandsausgänge dar. Diese werden in der Akku-Überwachung bzw. im Programm des c't-Bots ausgewertet. Der Pegel von *FC* liegt bei aktiver Schnellladung auf ungefähr 4,3 V. Mit dem Transistor *T2B* wird dazu ein einfacher Negator realisiert, denn der *FASTCHG*-Ausgang des Ladecontrollers legt bei aktiver Schnellladung den Pin auf Masse. Mit *ENA_C* kann die Funktion der Station überprüft werden und diese, wie oben schon beschrieben, auch deaktiviert werden.

Der rechte Teil der Schaltung dient der Ansteuerung des Gehäuselüfters. Mit *T1B* wird wieder ein Negator aufgebaut, so dass der Lüfter nur bei aktiver Schnellladung arbeitet. Mit Poti *R18* kann der Strom des Lüfters und somit die Geschwindigkeit den Bedürfnissen angepasst werden.

Abbildung 25 stellt den Kern der Schaltung dar.

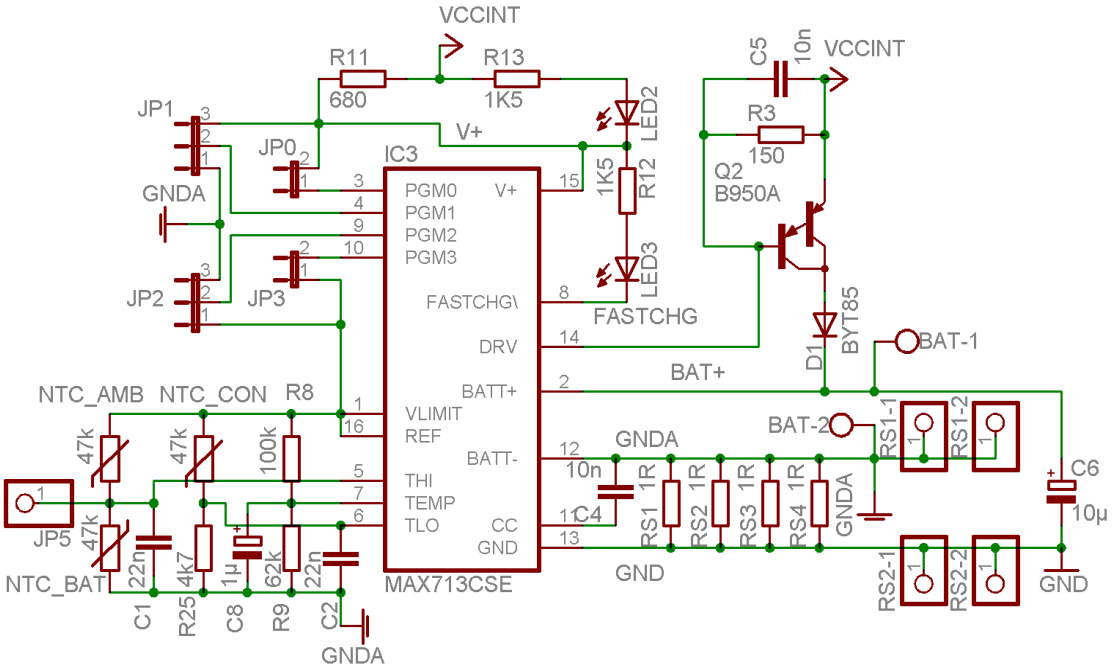


Abbildung 25: Schaltplan Ladestation Teil 3

Mit den Strommesswiderständen *RS1* bis *RS4* wird der Ladestrom eingestellt. Der Ladecontroller regelt den Strom so, dass über den Messwiderständen eine Spannung von 250 mV abfällt. Der Ladestrom lässt sich mit dem Ohmschen Gesetz berechnen.

$$I = \frac{U}{R} = \frac{250\ mV}{R} \tag{5.1}$$

Direkt auf der Platine wurde ein Widerstand mit 0,12 Ω verwendet. Somit ergibt sich ein Ladestrom von rund 2,1 A. Zusätzlich wird mit *RS1-x* und *RS2-x* ein Stecksockel und eine dazu passende kleine Steckplatine bereitgestellt, auf der sich noch weitere Widerstände parallel schalten lassen, um den Ladestrom ohne Ausbau der Hauptplatine anheben zu können.

Im unteren linken Teil der Abbildung 25 werden die Komponenten zur Temperaturüberwachung dargestellt. Für die Schwellenwerte bzw. Spannungsteiler steht dazu an *REF* eine Spannung von 2 V zur Verfügung. Bis auf den Spannungsteiler *NTC_CON - R25* entspricht dieser Teil der Schaltung den Empfehlungen des Datenblattes. Der untere Temperaturgrenzwert wird hierbei nicht wie vorgesehen für die Akku-Überwachung eingesetzt, sondern für eine Temperaturüberwachung des Konnektors. Erwärmt sich der Konnektor zu stark (aufgrund zu hoher Kontaktwiderstände und Ladeströmen), unterbricht die Schaltung die Schnellladung und geht in Erhaltungsladung über. Der Spannungsteiler für die untere Temperatur musste dazu modifiziert werden. Steigt die Temperatur im Konnektor an, reduziert sich der Widerstand des

NTC_CON[10]. Über *R25* fällt dadurch eine höhere Spannung ab. Ist diese größer als die Schwellenspannung des Spannungsteilers *R8 - R9*, wird die Schnellladung unterbrochen.

Für den Überladeschutz werden vom Ladecontroller zusätzlich verschiedene Schutzmechanismen eingesetzt. Zum einem ist das die Delta-V Abschaltung. Zur Erkennung dient dabei die Eigenschaft, dass bei Ende des Ladevorganges die Zellenspannung wieder etwas abfällt (siehe Abbildung 26).

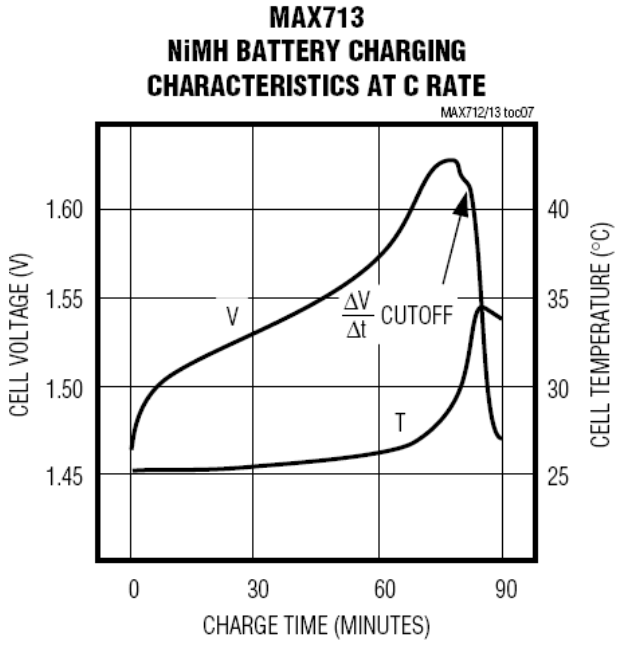


Abbildung 26: Ladecharakteristik von Ni-MH Akkus [Max02]

Desweiteren besitzt der Controller einen Sicherheitstimer, der die maximale Ladezeit begrenzt, sowie die bereits erwähnte Temperaturüberwachung. Dazu wurde ein NTC-Widerstand *(NTC_BAT)* mit thermischem Kontakt zum Akku angebracht. Dieses Signal wird über den Konnektor mit der Schaltung verbunden. Der Abschaltpunkt wurde auf ca. 20 Kelvin über der Umgebungstemperatur (gemessen mit *NTC_AMB*) gelegt. Versuche mit niedrigerem Schwellenwert, wie sich aus der Temperaturkennlinie (Abbildung 26) ableiten lässt, lieferten auch bei geringen Ladeströmen keine brauchbaren Ergebnisse. Die Akkutemperatur erreichte schon deutlich vor Ladeschluss den Schwellenwert, was zur Beendung der Schnellladung führte. An anderer Stelle wird eine Temperaturabschaltung, bei Ladeströmen bis 4 A, von 45 °C empfohlen. 60 °C wird als maximale Temperatur angegeben, welche auf keinen Fall überschritten werden sollsollte.[Hel07]

[10] NTC-Widerstände (engl. Negative Temperature Coefficient) sind Heißleiter dessen Widerstand bei Erhöhung der Temperatur abnimmt.

Abbildung 27 stellt die Spannung am Spannungsteiler *NTC_AMB - NTC_BAT* bei verschiedenen Umgebungstemperaturen dar. Wenn die Spannung kleiner als der Schwellenwert ist, wird die Schnellladung beendet. Es ist im Diagramm zu erkennen, dass dieses Ausschaltkriterium von 20 Kelvin bei verschiedenen Umgebungstemperaturen erreicht wird. Bei normalen Zimmertemperaturen liegt der Schaltpunkt, wie empfohlen, bei ungefähr 45 °C. Auch bei hohen Umgebungstemperaturen kann die Temperaturobergrenze von 60 °C eingehalten werden.

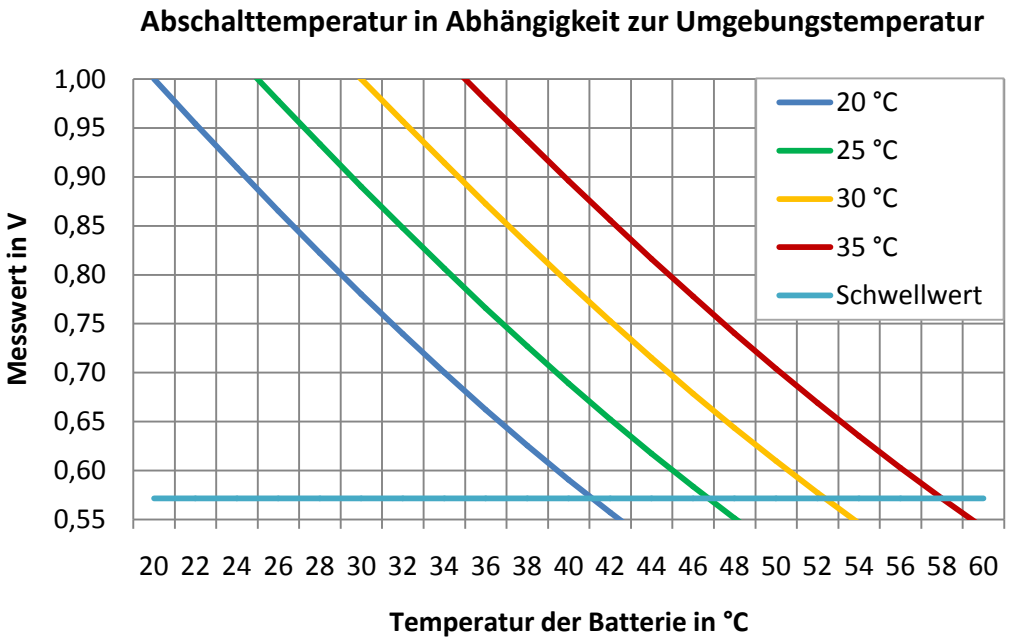


Abbildung 27: Temperaturabschaltungskennlinie der Ladeschaltung

Die Programmierung bzw. Einstellung des Ladecontrollers erfolgt mit den Jumpern *JP0* bis *JP3*. Dazu gehören die Einstellung der Zellenzahl sowie des Timers zur Festlegung der maximalen Ladezeit. Voreingestellt sind fünf Zellen und eine Ladezeit von maximal 90 min. Für eine abweichende Konfiguration befindet sich im Anhang B.3 dazu eine Anleitung zum richtigen Setzen der Jumper.

Der jeweilige Zustand der Schaltung lässt sich von außen durch die drei LEDs erkennen. Bei aktiver Schnellladung leuchten alle drei LEDs. Tabelle 6 im Anhang B.3 stellt die Anzeige sowie die Zustände und möglichen Ursachen dar.

Diese entwickelte Ladeschaltung ermöglicht eine schnelle und schonende Ladung des kompletten Akkupacks. In der verwendeten Ausbaustufe beträgt die Ladezeit des ausgeschalteten c't-Bots etwa eine Stunde. Auch wenn eine ständige Verfügbarkeit gefordert ist, muss gegenwärtig eine Wartezeit von einer Stunde in Kauf genommen werden. Es besteht darüber hinaus die Möglichkeit, mit höheren Ladeströmen (bis zu 6 A) zu arbeiten. Da der Kühlkörper an *D3* dafür nicht ausgelegt ist, muss in diesem Fall auf

den Verpolungsschutz verzichtet werden. Es ist dann *X1-2* für den Anschluss des positiven Gleichspannungspols zu verwenden.

Der äußere Aufbau der Ladestation wurde so gestaltet, dass der Roboter diese anhand des Liniensensors anfahren kann. Ein leicht trichterförmiger Eingang sichert dabei zusätzlich die richtige Position. Im letzten Ende der Anfahrt erfolgt durch Leitflächen die senkrechte Ausrichtung zur Ladestation und ermöglicht so eine sichere Verbindung mit dem Konnektor (siehe Abbildung 28).

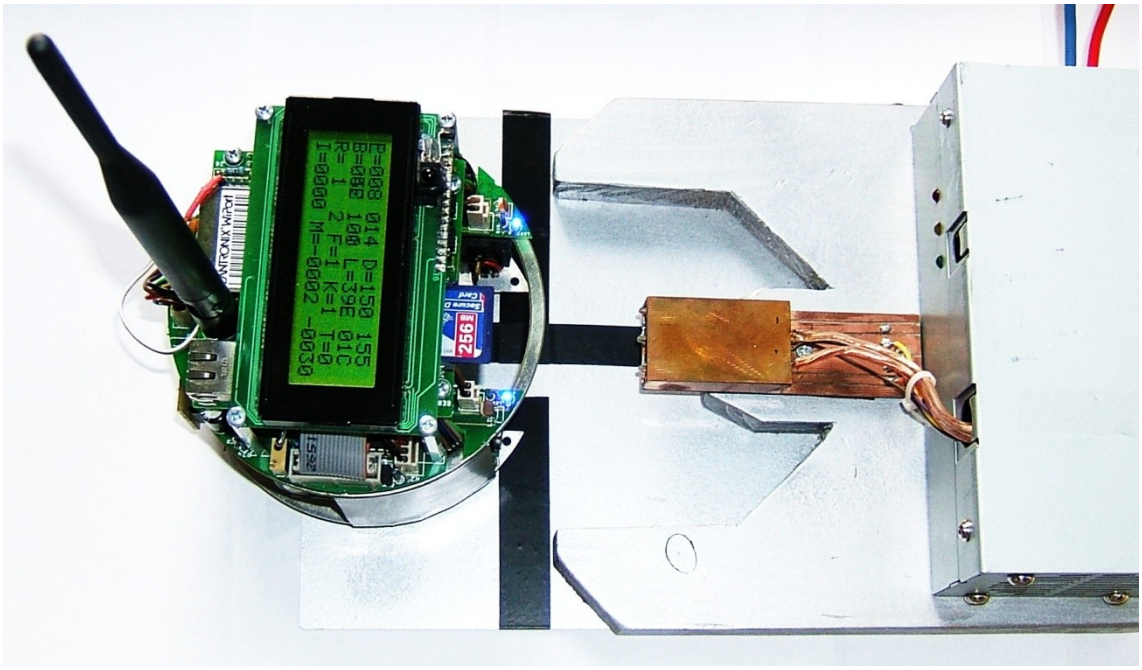


Abbildung 28: c't-Bot bei der Anfahrt der Ladestation

5.2.4 Konnektor Bot — Ladestation

Die Suche nach einer geeigneten elektrischen Steckverbindung zur Ladestation gestaltete sich etwas langwierig. Die Anforderungen waren durch die beabsichtigte Anwendung fest vorgegeben. Zum einen muss (um die Ladezeit kurz zu halten) ein möglichst hoher Strom möglich sein. Zum anderen darf die mechanische Verbindungskraft nicht zu hoch sein, damit der Roboter aus eigener Kraft andocken kann. Außerdem sollte der Steckverbinder möglichst einfach und auch unter ungünstigem Anfahrwinkeln eine sichere Verbindung gewährleisten. Dadurch verringert sich die Anforderung an die Positioniergenauigkeit, die mit dem Roboter erreicht werden muss.

Als Federkontakte stellen dreipolige Batterie-Kontake von Farnell (Bestellnr. 972-8350) die beste Lösung dar. Diese gewährleisten eine Strombelastung von jeweils maximal 3 A und lassen sich nach einer geringfügigen Modifikation hervorragend als leichtgängige Schleifkontakte verwenden.

Für die Übertragung des Ladestromes wurden zwei dreipolige Federkontakte verwendet. Somit lassen sich maximal 9 A übertragen. Für die Übertragung der Steuersignale

wurde eine etwas andere Variante gewählt. Um einen größeren Kontaktabstand zu erhalten, wurde der mittlere Kontakt entfernt. Der Abstand von nun 4 mm sollte ausreichen, um genügend seitlichen Spielraum zu ermöglichen, aber auch gleichzeitig eine Verpolung zu verhindern (siehe Abbildung 29).

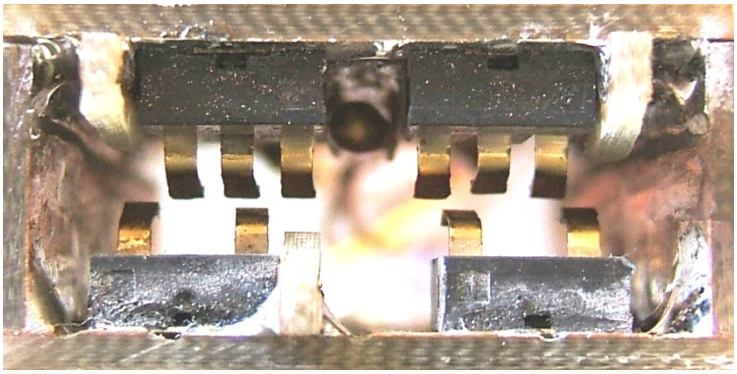


Abbildung 29: Konnektor Bot-Ladestation (weiblich)

Als Gegenstück wird eine Leiterplatte mit vier Kontaktbahnen auf der Unterseite sowie zwei auf der Oberseite verwendet (siehe Abbildung 18 und 20). Mechanische Führungen sichern das korrekte Einführen des Leiterbahnsteckers in die Ladebuchse.

Die Signalleitungen werden zur Temperaturüberwachung des Akkus, zur Erkennung der Betriebsbereitschaft des Ladegerätes sowie zur Erkennung des Ladezyklus (Schnellladung oder Erhaltungsladung) verwendet. Der freigebliebene vierte Kontakt dient als zusätzlicher Masse-Kontakt (Minuspol der Batterie).

Die hohen Anforderungen konnten mit der gewählten Konstruktion erfüllt werden. Die 4 mm Kontaktabstand ermöglichen ein seitliches Spiel von +/- 1 mm, ohne Gefahr einer Verpolung. Durch die mechanische Führung an der Ladestation dreht sich der Bot beim Anfahren automatisch in die richtige Position. Der erforderliche Kraftaufwand ist sehr niedrig und kann ohne Probleme vom Bot aufgebracht werden.

5.3 Spielfeld

Der „Arbeitsbereich“ des Roboters besteht aus einer 80 x 130 cm großen Platte, die mit einer 8 cm hohen Begrenzung versehen wurde. In einer Ecke ist die Ladestation untergebracht. Zur Suche und zum Auffinden der Ladestation wurde am Rand eine umlaufende Markierung angebracht, welche bis zur Ladestation führt. Mithilfe des Liniensensors kann der Roboter entlang dieser Linie in die Ladestation navigieren. Überwacht wird die Anlage von einer mittig über dem Feld angebrachten Netzwerkkamera, deren Bild mit im Test-Applet dargestellt wird (siehe Abbildung 32 auf Seite 62).

Die richtige Wahl des Untergrundes gestaltete sich etwas problematisch. Der optische Maussensor benötigt eine sehr fein strukturierte Oberfläche. Versuche mit Holzoberflächen, Laminat oder mit Holzdekorfolie beschichteten Arbeitsplatten erzielten recht gute

Ergebnisse. Der Anstrich der Platte mit weißem Lack lieferte eine sehr glatte Oberfläche. Die Erkennungsrate des Maussensors war aber so gering, dass eine Navigation damit unmöglich wurde. Abhilfe schaffte eine Beschichtung mit Hammerschlag-Lackspray. Dies lieferte eine leicht grau schimmernde Oberfläche mit feinen Strukturen, welche zu einer hohen Erkennungsrate des Maussensors führten.

Für die Verwendung des Simulators c't-Sim wurde eine, dem Spielfeld entsprechende, Welt entwickelt. Die Größe wurde etwas angehoben, da der Simulator mit recht großen Blockgrößen von 24 x 24 cm arbeitet. Somit können auch speziell für dieses Feld geschriebene Verhaltensroutinen, z. B. die Suche der Ladestation, mit dem Simulator getestet werden.

Mit der Akku-Überwachung und der Ladestation, sowie der dafür erforderlichen Akku-Halterung und dem Konnektor, wurden alle erforderlichen Bauteile für eine autonome bzw. ferngesteuerte Überwachung und Sicherung der Energieversorgung des c't-Bots realisiert. Das umrandete Spielfeld, einschließlich Netzwerkkamera, bietet eine geeignete Testumgebung, mit welcher die Funktionsfähigkeit dieser Komponenten nachgewiesen werden kann. Darüber hinaus kann mit ihm die Grundfunktionalität des c't-Bots demonstriert werden, sowie erste Test von selbstentwickelten Verhaltensroutinen vorgenommen werden.

6 Softwaremodifikationen und Erweiterungen

Die beabsichtigten Verwendungen des c't-Bots und die damit verbundenen Anforderungen an die Steuer- und Bedienbarkeit machten die Überarbeitung und Ergänzung der für den c't-Bot entwickelten Software erforderlich. Neben der Entwicklung der Funktionen zur direkten Programmierung des Roboters über eine Netzwerkverbindung waren vor allem die Funktionen zur Sicherstellung der Energieversorgung und Einbindung der dafür entwickelten Hardware zu erarbeiten.

Als Grundlage dient der Code aus dem Wiki zum c't-Bot in der Version 1279 vom 03.10.2007 [Heise_Wiki]. Es wurde bewusst versucht, die Anzahl der zu ändernden Passagen zu begrenzen, um eine Integration des mit dieser Arbeit entstandenen Programmcodes in zukünftige Softwareversionen nicht zu erschweren.

6.1 Anpassung Bootloader

Für ein Flashen des Mikrocontrollers mittels Applet ist eine Verbindung zum Bootloader erforderlich. Dieser Programmteil, welcher im obersten Speicherbereich des Mikrocontrollers installiert ist, wird als erstes nach einem Reset gestartet. Es initialisiert die USART–Schnittstelle und wartet fünf Sekunden auf ein eingehendes Programmierkommando. Nach Empfang werden die Update-Funktionen aktiviert. Dann kann der ganze untere Speicherbereich gelöscht und mit den über die USART-Schnittstelle vom WLAN-Modul empfangenen Daten neu beschrieben werden. Wird dieses Programmierkommando nicht empfangen, so wird zum Programmstart des „normalen“ Programms gesprungen.

Neben einer kleinen Korrektur musste der Bootloader für die Kommunikation mit dem Test-Applet nachgerüstet werden. Das Applet erwartet beim Start einen funktionierenden und kommunizierenden c't-Bot. Antwortet dieser nicht auf das Initialisierungskommando des Applets, wird der Start des Applets und somit jegliche Verbindung verhindert. Dies könnte z. B. nach einer fehlerhaften Programmierung auftreten.

Die Änderung befähigt nun den Bootloader, die Initialisierungsroutine des Applets zu verstehen und darauf zu antworten. Ein Verbindungsaufbau ist dann auch mit einem nicht „funktionierenden“ Bot durch einen Neustart des Applets und sofortigen manuellen Reset des c't-Bots möglich. Das Applet sowie der Bootloader begeben sich dann in den Programmiermodus. Danach kann in aller Ruhe ein neues Programm in den Flash-Speicher geladen werden.

Da der Code des Bootloaders sich auch in künftigen Softwareversionen nicht ändern sollte wird empfohlen, die mit dieser Arbeit ausgelieferte Version der Datei *bootloader.c* ohne Änderungen zu verwenden. Deshalb befindet sich hier auch keine Änderungsanweisung.

6.2 Anpassung an die verwendete Fernbedienung

Die Verwaltung der Fernbedienungscodes ist in *rc5-codes.h* untergebracht. Dort sind auch die für dieses Modell verwendeten Codes hinterlegt. Da die Grundstruktur dieser Datei immer die gleiche bleiben sollte, kann sie, bei Nutzung einer neueren Softwareversion für den c't-Bot, durch die in dieser Arbeit hinterlegten Datei ohne Änderungen ersetzt werden. Falls dennoch Bedarf besteht die Fernbedienungscodes zu ändern, sollte beachtet werden, dass im Applet diese RC5-Codes ebenfalls verwendet werden.

Die Bedienungsmöglichkeiten übers Applet wurden mit drei neuen Tasten zur direkten Geschwindigkeitswahl ergänzt. Desweiteren wurde eine Funktion implementiert, die eine Kurvenfahrt beendet, ohne den Bot dabei anzuhalten. Dazu wird die mittlere Geschwindigkeit als neue Sollgeschwindigkeit verwendet.

In *rc5.c* sind dazu die Funktion *rc5_bot_straight* hinzuzufügen sowie der *default_key_handler* wie angegeben zu ergänzen:

```
/*!
 * @brief   Setzt Fahrtrichtung gerade
 */
static void rc5_bot_straight(void) {

   int16 speed;
   speed = (target_speed_l + target_speed_r)/2;
   target_speed_l = speed; // Geschwindigkeit gleich mittlere Geschw.
   target_speed_r = speed;

}

void default_key_handler(void){

   switch (RC5_Code){

…

   #ifdef MCU // erweiterte Codes

      case RC5_CODE_STRAIGHT: rc5_bot_straight(); break;

      case RC5_CODE_SLOW:  target_speed_l = BOT_SPEED_SLOW;
                           target_speed_r = BOT_SPEED_SLOW; break;

      case RC5_CODE_NORMAL:  target_speed_l = BOT_SPEED_NORMAL
                             target_speed_r = BOT_SPEED_NORMAL; break;

      case RC5_CODE_FAST:    target_speed_l = BOT_SPEED_FAST
                             target_speed_r = BOT_SPEED_FAST; break;

      case RC5_CODE_TURN_R: bot_turn(0, -45); break;
      case RC5_CODE_TURN_L: bot_turn(0,  45); break;

   #endif
```

6.3 Reset-Kommando

Ein Softwareupdate, bzw. das Flashen des Mikrocontrollers über das Applet, ist nur über eine Verbindung mit dem Bootloader möglich. Dies kann durch ein Reset erreicht werden, da nach einem Reset bzw. Start der MCU der Bootloader als erstes ausgeführt wird. Deshalb ist die Integration eines, vom Applet aus per Steuerbefehl aufrufbaren, Reset-Kommandos unabdingbar gewesen.

Dieser Befehl startet den Watchdog des Mikrocontrollers. Der eigentlich zur Programmüberwachung gedachte Timer führt nach Verstreichen des Watchdog-Zeitintervalls einen Reset aus und startet somit den Bootloader.

Mit dem Befehl ist es aber auch möglich, durch den Neustart des Bots, eine Beendung fehlerhafter Routinen sowie die erneute Initialisierung der Hardware zu erzwingen.

Dazu in *command.h* das Kommando definieren:

```
#define CMD_RESET  'b' // Veranlasst Reset des Bots
```

in *command.c* die Watchdog-Bibliothek einbinden:

```
#include <avr/wdt.h>
```

und die Abarbeitung des Kommandos einfügen.

```
case CMD_RESET:

#ifdef MCU

   // Displayausgabe
   display_clear();
   display_cursor(1,1);
   display_printf("Reset c't-bot");
   display_cursor(2,1);
   display_printf("booting in 500 ms");
   display_cursor(3,1);
   display_printf("Please wait!");

   //antwort senden
   command_write(CMD_RESET, SUB_CMD_NORM,0,0,0);
   uart_flush();

   // Watchdog starten
   wdt_enable(WDTO_500MS);

   // Interruptbearbeitung deaktivieren
   cli();

   // warten auf Reset
   for(;;);

#endif

break;
```

6.4 Akku-Überwachung

Für die Integration der entwickelten Hardware, insbesondere zur Kommunikation mit der Akku-Überwachung, waren, neben der Programmierung der Software für dieses Modul sowie der Routinen für den c't-Bot (siehe *bat-control.h* und *bat-control.c*), auch einige Ergänzungen der vorhandenen Quelltexte nötig.

In *ct-Bot.h* ist dazu folgendes hinzuzufügen:

```
#define ACCU_AVAILABLE     // Baterieüberwachung verfügbar
```

Weiter unten dann noch die folgenden Abhängigkeiten implementieren:

```
#ifdef PC
   #undef  ACCU_AVAILABLE
#endif

#ifdef ACCU_AVAILABLE
   #define TWI_AVAILABLE      // TWI-Schnittstelle (I2C) nutzen
   #define TIME_AVAILABLE
#endif
```

In *sensor.h* die beiden globalen Variablen definieren. Diese lassen sich dann ebenso wie die vorhandenen Sensorwerte nutzen.

```
#ifdef ACCU_AVAILABLE
   #include "bat-control.h

   extern uint16 sensVoltage;  // Spannung für Motor in mV
   extern uint16 sensCurrent;  // Stromverbrauch des Bots in mA
#endif
```

In *sensor-low.c* die regelmäßige Abfrage in folgender Routine hinzufügen. Diese updatet unter anderem die oben definierten globalen Variablen.

```
void bot_sens_isr(void){

...

   if (dist_ticks-old_dist > MS_TO_TICKS(32)){

...

      #ifdef ACCU_AVAILABLE
         Read_Bat_Control();
      #endif

...

   }
}
```

In *twidriver.h* ist eine Routine zu ersetzen. Diese Routine hat vorher bei einem Verbindungsfehler unendlich lang gewartet, was zu regelmäßigem Hängenbleiben des Programms führte. Die Routine bricht nun nach einer Zeitüberschreitung ab. Dies führt

dazu, dass die Schreib- bzw. Leseoperationen über die I²C-Schnittstelle abgebrochen werden. Eine Wiederholung der Leseoperation erfolgt im nächsten Programm-Intervall.

```
#include "timer.h"


/*!
 * Warte auf TWI Interrupt
 * wenn kein TWI Interrupt - Abbruch nach 1 ms
 */
void Wait_TWI_int(void){

   register uint8 ticks = TIMER_GET_TICKCOUNT_8;
   uint8 stop = ticks + MS_TO_TICKS((uint16)1);

   while (!(TWCR & (1<<TWINT)) && ticks != stop)
   ticks = TIMER_GET_TICKCOUNT_8;

}
```

Eine Abhandlung über die Programmierung des ATmega48 der Akku-Überwachung ist hier nicht vorgesehen. Die Funktionalität kann der dazugehörigen Hardwarebeschreibung (siehe Kapitel 0) sowie der Schnittstellenbeschreibung im Anhang D.3 entnommen werden. Der Code dafür, bzw. das komplette Projekt, befinden sich auf der CD im Eclipse-Workspace.

6.5 Setup-Menü

Das Setup-Menü ist für die Anzeige und Einstellung verschiedener Parameter entwickelt worden. Es erweitert damit die vorhandenen Anzeigemöglichkeiten um ein komfortables Untermenü. Dadurch ergibt sich eine Reduzierung der Seitenanzahl im Hauptmenü des Bots, was die Navigation durch die Menüs vereinfacht. Die Umschaltung zwischen den einzelnen Screens erfolgt mit der „*Display*“-Taste (siehe 0).

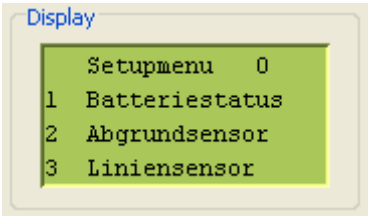


Abbildung 30: Setup-Menü des c't-Bots

Der Quellcode dafür wird mit den Dateien *setup.h* und *setup.c* eingebunden. Ferner sind noch eine einige Ergänzungen des vorhandenen Codes erforderlich.

In *available_screens.h* ist der Aktivierungsschalter hinzuzufügen:

```
#define DISPLAY_SETUP_AVAILABLE
```

In *gui.c* ist die Header Datei einzubinden:

```
#include "setup.h"
```

sowie in der Funktion *gui_init* die Initialisierung vorzunehmen.

```
#ifdef DISPLAY_SETUP_AVAILABLE
   bot_setup_init();
   register_screen(&setup_display);
#endif
```

6.5.1 Setup Abgrundsensor

Ein neues Untermenü gestattet es den Schwellenwert für die Abgrunderkennung zur Laufzeit zu verändern und im EEPROM zu speichern. Vorher wurde der Schwellenwert durch die Konstante *BORDER_DANGEROUS* fest definiert. Um nicht in allen Dateien diese Konstante ersetzten zu müssen, wurde diese unter gleichem Namen als Variable definiert. Für die Programmierung der Verhaltensregeln ergibt sich somit kein Unterschied. In *bot-local.h* sind dafür folgende Änderungen vorzunehmen:

```
/* Einstellunge fuer die Verhaltensregeln */

#ifdef DISPLAY_SETUP_AVAILABLE

   extern uint16 BORDER_DANGEROUS;  // wird im Setup-Menue eingestellt
   #define BORDER_DANGEROUS_DEFAULT   0x2B0 // Voreinstellung

#else

   #define BORDER_DANGEROUS   0x2B0
   // Wert, ab dem wir sicher sind, dass es eine Kante ist

#endif
```

6.5.2 Setup Liniensensor

Wie schon für die Abgrundsensoren wird die Konstante für den Schwellenwert der Linienerkennung durch eine Variablendefinition ersetzt. Dazu in *bot-local.h* diese Definition entsprechend anpassen:

```
// Konstante fuer das bot_follow_line_behaviour-Verhalten
#ifdef DISPLAY_SETUP_AVAILABLE

   extern uint16 LINE_SENSE; // wird im Setup-Menue eingestellt
   #define LINE_SENSE_DEFAULT  0x300

#else

   #define LINE_SENSE     0x300  // Ab wann ist es eine Linie?
   //(schwarz ca. 0x300, helle Tischflaeche 0x50)

#endif
```

6.6 Sonstige Modifikationen

Für den reibungslosen Betrieb waren noch weitere kleine Modifikationen nötig.

Die korrekte Position der Klappe muss in *bot-local.h* angepasst werden:

```
#define DOOR_OPEN 15
```

Wenn nicht schon geschehen sind in *ct-Bot.h* mindestens folgende Definitionen zu aktivieren (siehe auch [Heise_Wiki] bzw. Literatur\Optionen_ct-Bot_h.pdf):

```
#define LOG_CTSIM_AVAILABLE
#define LED_AVAILABLE
#define IR_AVAILABLE
#define RC5_AVAILABLE
#define BOT_2_PC_AVAILABLE
#define DISPLAY_AVAILABLE
#define DISPLAY_REMOTE_AVAILABLE
#define ADC_AVAILABLE
#define MAUS_AVAILABLE
#define ENA_AVAILABLE
#define SHIFT_AVAILABLE
#define SPEED_CONTROL_AVAILABLE
#define BEHAVIOUR_AVAILABLE
#define ACCU_AVAILABLE
```

Die Definition des Bootloaders ist nur für die erstmalige Installation dessen erforderlich, bzw. bei Veränderung des Bootloaders. Das Flashen des Controllers kann dann nicht über das Applet erfolgen. Stattdessen ist der ISP-Programmieradapter zu verwenden (siehe dazu Anhang C.1).

6.7 Einbindung des Beispielverhaltens

Die in Kapitel 4.4.2 bereits vorgestellte Verhaltensroutine „*bot_example_behaviour*“ lässt sich wie folgt mit ins Programm einbinden.

In *available_behaviours.h* das Verhalten definieren sowie die Header-Datei einbinden

```
#define BEHAVIOUR_EXAMPLE_AVAILABLE

#include "bot-logic/behaviour_example.h"
```

In *bot-logic.c* das Verhalten in die Verhaltensliste eintragen.

```
#ifdef BEHAVIOUR_EXAMPLE_AVAILABLE

   insert_behaviour_to_list(&behaviour,
      new_behaviour(68,bot_example_behaviour, INACTIVE));

   // Registrierung zur Behandlung des Notfallverhaltens
   register_emergency_proc(&bot_example_behaviour_handler);

#endif
```

Um dieses Verhalten vom Applet aus aufrufen zu können ist folgender Eintrag in *behaviour_remotecall.c* notwendig:

```
#ifdef BEHAVIOUR_EXAMPLE_AVAILABLE
   PREPARE_REMOTE_CALL(bot_example,2,"int16 var1, int16 var2",2,2),
#endif
```

6.8 Anfahrt der Ladestation

Für das Suchen und Anfahren der Ladestation wurde eine eigene Verhaltensroutine entwickelt. Der Code dazu befindet sich in *behaviour_search_charger.h* und *behaviour_search_charger.c.*

Die Suche der Ladestation erfolgt nach einer einfachen Strategie. Dazu ist auf dem Spielfeld eine umlaufende Linie aufgebracht, welche zur Ladestation führt.

- auf die Wand mit dem geringsten Abstand zum Bot zufahren
- die Linie vor der Wand suchen und entlang dieser im Uhrzeigersinn bis in die Ladestation fahren
- nach Erkennung des Kontaktes mit der Ladestation die Bewegung stoppen.

Für die Einbindung sind, wie bei jedem Verhalten, folgende Schritte notwendig:

In *available_behaviours.h* den Aktivierungsschalter definieren sowie die Header-Datei einbinden.

```
#define BEHAVIOUR_SEARCH_CHARGER_AVAILABLE

#include "bot-logic/behaviour_search_charger.h"
```

In *bot-logic.c* das Verhalten in die Verhaltensliste eintragen.

```
#ifdef BEHAVIOUR_SEARCH_CHARGER_AVAILABLE

   insert_behaviour_to_list(&behaviour,
      new_behaviour(88,bot_search_charger_behaviour, INACTIVE));

   // Registrierung zur Behandlung des Notfallverhaltens
   register_emergency_proc(&border_search_charger_handler);

#endif
```

In *behaviour_remotecall.c* den Aufruf aus dem Applet ermöglichen.

```
#ifdef BEHAVIOUR_SEARCH_CHARGER_AVAILABLE
   PREPARE_REMOTE_CALL(bot_do_search_charger,0,""),
#endif
```

6.9 Motorregelung

Aufgrund der anfangs unzureichenden Fahrgenauigkeit wurden Ansätze für die Entwicklung einer verbesserten Motorregelung gesucht. Im Gegensatz zur Version der c't, welche für jedes Rad einen eigenen Regler verwendet, wurde hier eine Regelung für die Bewegung des Mittelpunktes sowie eine Regelung für den Drehwinkel vorgezogen.

Das Grundprinzip ist eine Vorwärtssteuerung mit Nachregelung sowie adaptiver Parameteranpassung. Dazu werden die für die Bewegung des Roboters benötigten Antriebsleistungen und Geschwindigkeiten berechnet. Mit diesen Werten wird dann die dafür notwendige Motorspannung berechnet. Die Ermittlung des PWM-Wertes kann aufgrund der Kenntnis der Versorgungsspannung (*sensVoltage*) spannungsabhängig realisiert werden. Die Abweichungen der Steuerung sollen mit einem PI-Regler ausgeregelt werden.

6.9.1 Umsetzung im Motor

Die Umsetzung der zugeführten elektrischen Energie im Motor lässt sich durch folgende Gleichungen beschreiben:

$$P_{el} = P_{mech} + P_J \tag{6.1}$$

$$UI = \frac{\pi}{30000} nM + RI^2 \tag{6.2}$$

Mit Gleichung (6.3) kann der Strom ersetzt werden.

$$M = k_M I \tag{6.3}$$

$$U\frac{M}{k_M} = \frac{\pi}{30000} nM + R\left(\frac{M}{k_M}\right)^2 \tag{6.4}$$

Umformen nach U ergibt:

$$U = \frac{\pi}{30000} nk_M + R\frac{M}{k_M} \tag{6.5}$$

Durch die Abhängigkeit der Drehzahlkonstante von der Drehmomentkonstante (6.6) kann k_M ersetzt werden. Man erhält mit (6.7) eine Formel für die benötigte Motorspannung U, welche nur noch von der Drehzahl n und dem Drehmoment M abhängig ist.

$$k_n k_M = \frac{30000}{\pi} \tag{6.6}$$

$$U = \frac{n}{k_n} + \frac{\pi}{30000} Rk_n M \tag{6.7}$$

Mit Hilfe dieses Zusammenhanges lässt sich, aus Kenntnis der gewünschten Drehzahl und des benötigten Antriebsmomentes, die erforderliche Motorspannung berechnen.

Abbildung 31 stellt die Drehzahlkennlinie, Stromkennlinie, Abgabeleistung- sowie Wirkungsgradkennlinie des verwendeten Motors bei einer Spannung von 6 V dar [Rob07]. Allerdings ist zu beachten, dass dies die Kennlinien vom Motor ohne Getriebe sind.

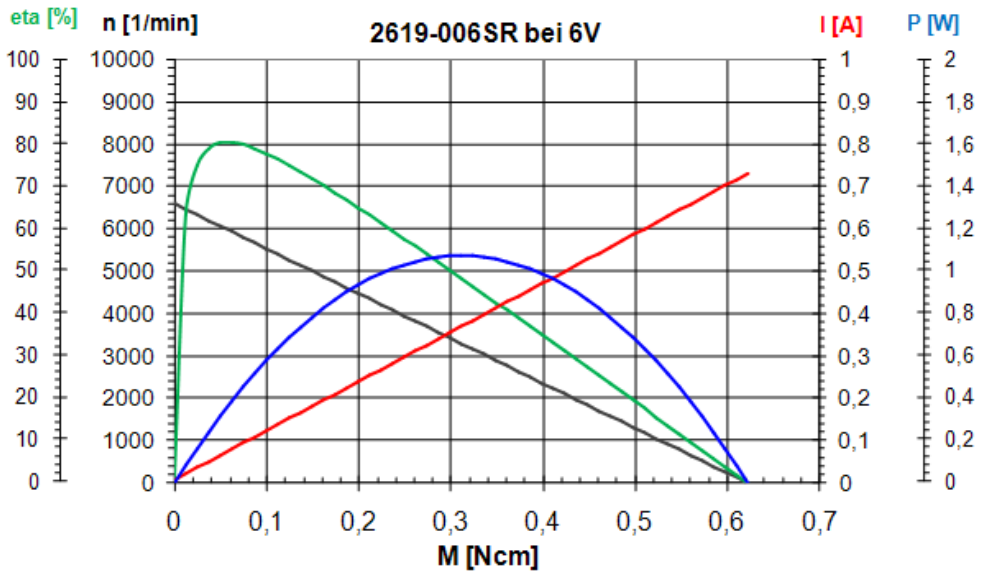


Abbildung 31: Motorkennlinien des FTB 2619-006 SR

Die Formeln und Grundlagen dieses Kapitels sind aus dem Dokument „Das wichtigste über maxon Motoren“ [max07] entnommen worden (siehe auch Anhang E.2).

6.9.2 Berechnung von Drehmoment und Drehzahl

Die für die Berechnung der Motorspannung benötigen Angaben über Antriebsmomente und Drehzahlen werden aus den Sollgeschwindigkeiten des Roboters und den dafür erforderlichen Antriebsleistungen berechnet. Die benötigte Leistung setzt sich zusammen aus der Differenz der kinetischen Energie von Soll- und Istgeschwindigkeit des Intervalls sowie der zur Überwindung der Reibung benötigten Leistung. Die Leistungen werden getrennt für die Translation sowie die Rotation berechnet.

$$P_{trans} = \frac{\Delta E_{kin}}{T} + F_R v = \frac{m}{2T}\left(v_{neu}^2 - v_{ist}^2\right) + F_R v \tag{6.8}$$

$$P_{rot} = \frac{\Delta E_{rot}}{T} + M_R \omega = \frac{J_S}{2T}\left(\omega_{neu}^2 - \omega_{ist}^2\right) + M_R \omega \tag{6.9}$$

Aus den benötigten Leistungen für Rotation und Translation werden die Antriebsleistungen der zwei Räder bestimmt.

$$P_{rechts} = \frac{P_{trans} - P_{rot}}{2} \tag{6.10}$$

$$P_{links} = \frac{P_{trans} + P_{rot}}{2} \tag{6.11}$$

Die einzelnen Radgeschwindigkeiten werden aus den Sollgeschwindigkeiten für Translation und Rotation errechnet.

$$v_{rechts} = v_{trans} - \omega_{Bot} \frac{Radstand_{Bot}}{2} \tag{6.12}$$

$$v_{links} = v_{trans} + \omega_{Bot} \frac{Radstand_{Bot}}{2} \tag{6.13}$$

Mit diesen Werten können die Drehzahl und das Antriebsmoment ermittelt werden. Für das linke Rad erfolgt die Berechnung analog. Zu beachten sind die zum Teil unterschiedlichen Größenangaben und die daraus resultierenden Anpassungen.

$$n_{rechts} = 60 \frac{v_{rechts}}{u_{Rad}} u_{Getriebe} \tag{6.14}$$

$$M_{rechts} = 60 \frac{P_{rechts}}{2\pi n_{rechts}} \tag{6.15}$$

Mit Gleichung 6.7 lässt sich nun die, für die gewünschte Bewegung des Roboters benötigte, Motorspannung bestimmen. Der Effektivwert dieser Spannung wird durch Pulsweitenmodulation aus der Betriebsspannung des Roboters erzeugt und treibt den Motor an.

Einige Fragen blieben bisher noch ungeklärt: Wie kann der Wirkungsgrad des Getriebes mit berücksichtigt werden? Ist der Wirkungsgrad des Getriebes konstant oder müssen die Verluste des Getriebes als ein Reibungsmoment aufgefasst werden? Wie kann ein solches Reibungsmoment bestimmt werden?

6.9.3 Parameterbestimmung

Die Parameter für den Motor konnten aus dem Datenblatt entnommen werden. Nicht ganz trivial gestaltete sich dagegen die Bestimmung der restlichen Parameter des Roboters. Während sich die Masse sehr leicht ermitteln lässt, ist eine genaue Bestimmung des Trägheitsmomentes, wegen der ungleichen Massenverteilung, ohne Hilfsmittel nicht möglich. Aufgrund des recht geringen Einflusses der Winkelbeschleunigungsarbeit auf die benötigte Leistung wird nur eine Abschätzung derer vorgenommen. Als Schätzwert wird der Mittelwert aus den beiden Grenzwerten der Trägheitsmomente für eine gleichmäßig über der Grundfläche verteilte Masse und einer nur am Rand befindlichen Masse verwendet. Demnach lässt sich das Trägheitsmoment mit Gleichung 6.16 abschätzen. Eventuelle Abweichungen müssen von der Nachregelung übernommen werden.

$$J_S = \frac{3}{4} m r^2 \tag{6.16}$$

Die Reibungskraft wurde mit einem Zugkraftmesser bestimmt. Da sie jedoch stark vom Untergrund abhängig ist, soll sie während der Fahrt adaptiv angepasst werden. Dies soll durch einen Vergleich der resultierenden Bewegungsleistung mit der berechneten Leistung erfolgen.

Das Reibungsmoment ist im Wesentlichen gekennzeichnet durch die Reibung der hinteren Auflage (Gleitpin bzw. Stützbürste). Es lässt sich daher von der Reibungskraft ableiten. Der Abstand des Stützpunktes vom Drehpunkt, dem Abstand zur Roboterachse, bildet den Skalierungsfaktor.

$$M_R = r_S F_R \tag{6.17}$$

Im Anhang E.2 befinden sich eine Auflistung der Parameter sowie eine Beschreibung der in den Formeln verwendeten Abkürzungen.

Erste Tests lieferten brauchbare Ergebnisse. Insbesondere der Geradeauslauf des Roboters konnte überzeugen. Die Regelung war aber noch nicht ausgereift und neigte zu Schwingungen. Da vom Entwicklerteam des c't-Bots eine verbesserte Motorregelung veröffentlicht wurde, welche befriedigende Ergebnisse lieferte, wurde hier die weitere Entwicklung eingestellt.

Mit den überarbeiteten Funktionen und speziellen Erweiterungen, insbesondere der Ansteuerung der erweiterten Hardware, stehen nun die für eine Verwendung als mobilen Praktikumsroboter notwendigen Funktionen zur Verfügung. Wichtigste Punkte sind dabei die zur Sicherstellung der Energieversorgung benötigte Funktionalität, die Anfahrt der Ladestation sowie die Unterstützung des Flash-Vorgangs über eine Netzwerkverbindung.

7 Test-Applet

Das Test-Applet ist eine prototypische Implementierung der Grundmodule einer Client-Software und dient zum Test bzw. Nachweis der Funktionsfähigkeit der Soft- und Hardwarekomponenten des c't-Bots. Es bietet dazu Funktionen zur Überwachung, Steuerung und Programmierung (Flashen) des Roboters über eine Netzwerkverbindung. Als Vorlage diente der Applet-Code aus dem Wiki [Heise_Wiki]. Wesentliche Erweiterungen sind die Integration eines Proggers zum Flashen des Mikrocontrollers sowie zusätzliche Überwachungs- und Logging-Funktionalitäten.

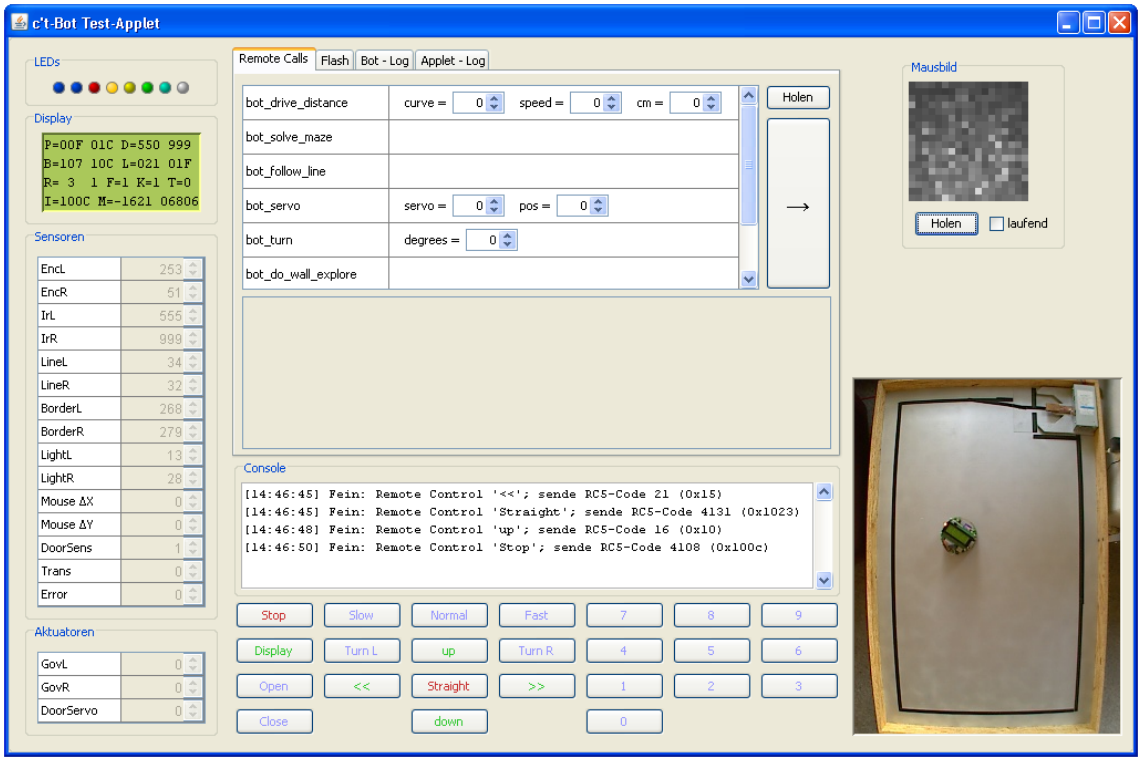


Abbildung 32: Screenshot des Test-Applets

7.1 Überwachung

7.1.1 Anzeige der Statusinformationen

Die linke Spalte zeigt die Statusmeldungen des c't-Bots an. Neben dem aktuellen Zustand der LEDs sowie des Inhaltes des Displays werden alle Sensorwerte sowie die Sollwerte für Motor und Klappe dargestellt. Hinter *IrL* und *IrR* verbergen sich die in mm umgerechneten Messwerte des Abstandssensors. Es ist darauf zu achten, dass die Sensorwerte alle dezimal angezeigt werden. Im Gegensatz dazu wird teilweise eine hexadezimale Darstellung der Werte in den entsprechenden Displays verwendet.

In dem Fenster rechts oben lässt sich das von der optischen Maus des c't-Bots aufgenommene Bild anzeigen. Dies kann zur Überprüfung und Einstellung des Abstandes von der Maussensorplatine zur Bodenfläche verwendet werden. Die Check-Box „*laufend*" sollte im normalen Betrieb deaktiviert sein, weil die Abfrage und Übertragung dieses Bildes den Roboter recht viel Zeit kosten.

7.1.2 Web-Cam

Das Web-Cam Bild bietet eine gute optische Überwachung des Spielfeldes. Nur hiermit lässt sich zuverlässig feststellen wo der Bot sich befindet, ob er fährt oder sich an einem Hindernis festgefahren hat. Die Darstellung ist deshalb nicht nur ein optisches Beiwerk sondern für die Kontrolle unverzichtbar.

7.1.3 Logging

Die dritte Kontrollmöglichkeit wird durch die Ausgabe von Log-Informationen verwirklicht. Dazu dienen ein *Console* sowie die Tabs „*Bot Log"* und „*Applet Log"*. In „*Bot Log"* werden alle Log-Informationen, die vom Bot gesendet werden, angezeigt. Demzufolge stehen in „*Applet Log"* die Meldungen, welche das Applet selber generiert. Zusätzlich werden alle Infos auch in der *Console* dargestellt. Ebenso wurde die Standardausgabe des Applets umgeleitet, so dass dort auch die Java-Fehlermeldungen erscheinen.

7.2 Steuerung

Die Steuerung des Roboters lässt sich auf zwei unterschiedliche Arten bewerkstelligen. Zum einen ist eine direkte Steuerung mittels der Fernbedienungskonsole möglich, zum andren kann durch den Aufruf von Verhaltensroutinen eine indirekte Steuerung erfolgen.

7.2.1 Steuerung mittels Fernbedienungskonsole

Die direkte Steuerung des Bots erfolgt mit der Fernbedienungskonsole. Das Applet übermittelt dem c't-Bot die hinterlegten RC5-Codes. Es macht daher keinen Unterschied ob diese Codes vom Applet kommen oder ob die Signale vom IR-Sensor des Roboters empfangen wurden.

Die Auswertung und Abarbeitung der RC5-Codes erfolgt im Bot durch das ausgewählte User-Interface, welches im Display angezeigt wird. Erfolgt dort keine Abarbeitung wird der *default_key_handler* aufgerufen. Abgesehen vom Ziffernblock sollte die Abarbeitung normalerweise unabhängig vom ausgewählten User-Interface sein.

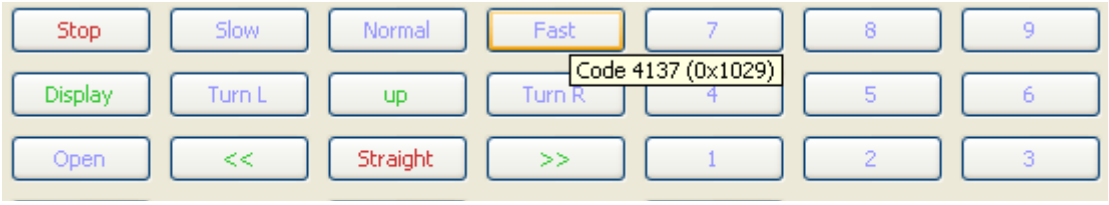


Abbildung 33: Screenshot der Fernbedienungskonsole

Mit *Stop* wird ein sofortiger Stopp der Bewegung des Roboters veranlasst. *Slow*, *Normal* und *Fast* werden zur Geschwindigkeitswahl verwendet. Mit *Display* erfolgt ein Zappen durch die User-Interfaces bzw. der dazugehörigen Displayansichten. Mit *Open* und *Close* kann die Transportklappe des Bots geöffnet und geschlossen werden. *Turn L* und *Turn R* veranlassen jeweils eine Drehung um 45 Grad. Mit *up* und *down* kann die Geschwindigkeit sukzessiv erhöht bzw. erniedrigt werden. Für die Drehbewegung stehen dazu die Tasten „<<“ und „>>“ zur Verfügung. Ein Stopp der Drehbewegung lässt sich mit *Straight* erreichen.

7.2.2 Steuerung mittels Aufruf von Verhaltensroutinen

Desgleichen ist eine gute Steuerung durch den Aufruf der Verhaltensfunktionen möglich (siehe auch Anhang D.2). Dazu den Tab „*Remote Calls*“ öffnen. Mit *Holen* wird eine neue Auflistung der verfügbaren Routinen vom Bot angefordert. Das ist nur nach einer Neu-Programmierung des Bots notwendig.

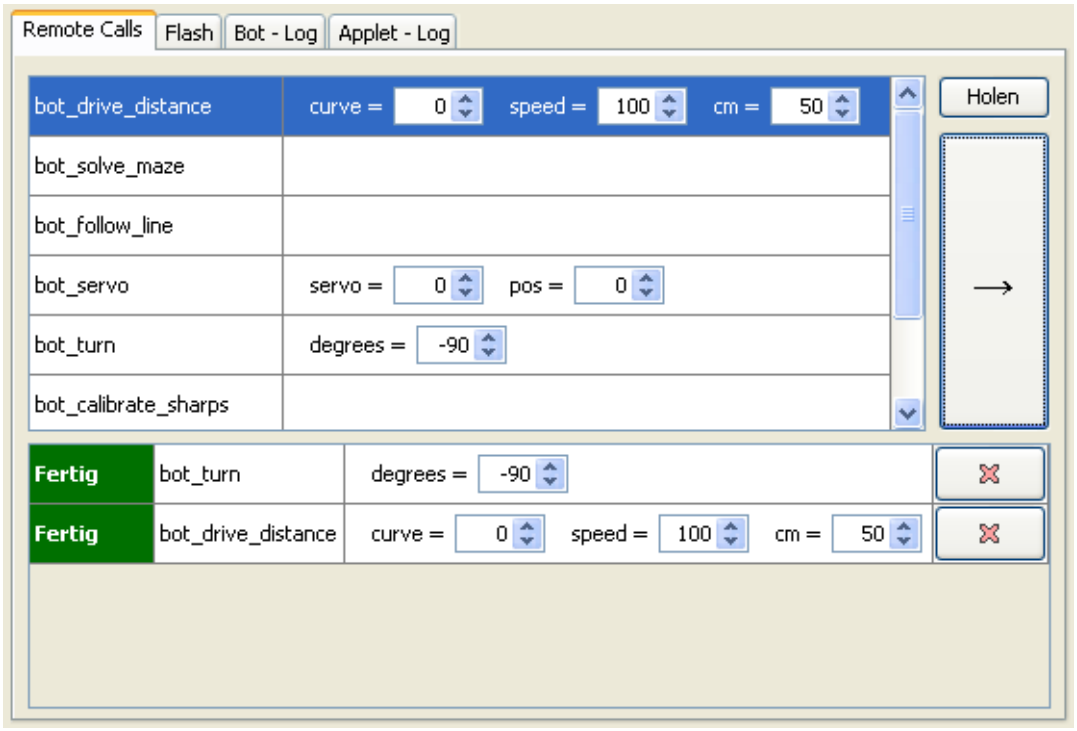


Abbildung 34: Aufruf von Verhaltensroutinen im Applet

Durch Selektion eines Eintrages und Klick auf den nebenstehenden großen Enter-Button wird das Verhalten ausgeführt. Im darunter liegenden Fenster wird der Status der Ausführung angezeigt. Die Abarbeitung erfolgt anhand einer Liste. Es lassen sich daher Befehle aktivieren auch wenn die Abarbeitung des Vorherigen noch nicht abgeschlossen ist.

Durch den Aufruf von Verhaltensroutinen sind präzise Steuerungen durchführbar, da der c't-Bot die Ausführung überwacht und regelt. Dagegen wird eine exakte, direkte Steuerung, aufgrund der Reaktionszeit des Bedieners sowie der Verzögerung des Überwachungsvideos, nahezu unmöglich.

7.3 Programmierung

Im Tab *Flash* sind die Funktionen zum Flashen des ATmeag644 integriert worden. Als Anhaltspunkt konnte JAvrProg [Fri04] verwendet werden. Dieses Java Programm gestattet die Programmierung von AVR-Mikrocontrollern über eine serielle Verbindung. Neben der Unterstützung des verwendeten ATmega644 mussten die Ein- und Ausgaberoutinen, zur Kommunikation über die Netzwerkverbindung, realisiert werden.

Mit „*Reset Bot*" wird ein Reset-Kommando zum Roboter gesendet. Nach dem Neustart des c't-Bots wird der Bootloader aktiviert. In der *Console* sollte dann die Meldung „AVRBOOT gefunden!" erscheinen.

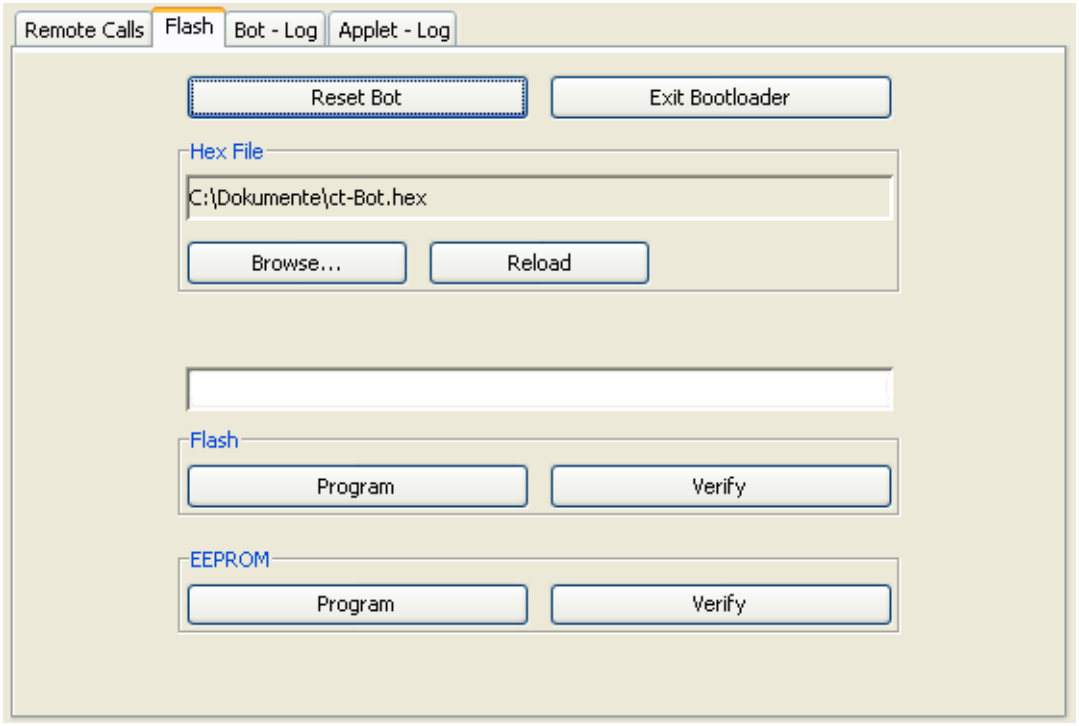


Abbildung 35: Bedienung der Flash-Funktionen des Applets

Danach ist mit „*Browse…“* das zu übertragende Programm bzw. die HEX-Datei[11] auszuwählen. Der kompilierte Programmcode für den Flashspeicher befindet sich in der Datei *ct-Bot.hex*, die Belegung des EEPROMs ist dagegen in der Datei *ct-Bot.eep* zu finden.

Mit „*Program*“ wird die zuvor ausgewählte HEX-Datei in den Flashspeicher bzw. ins EEPROM übertragen. Optional kann noch eine Überprüfung des Speicherinhaltes mit dem im *„HEX File“* angegebenen Daten durch Anwahl von „*Verify*“ erfolgen. Mit „*Exit Bootloader*“ wird der Bootloader wieder verlassen und das übertragene Programm gestartet.

Der Flash-Vorgang erfordert die aktive Kommunikation mit dem Steuercode des Roboters. Ist dies nicht mehr möglich, weil z. B. eine Software überspielt wurde, die diese Funktionen nicht mehr unterstützt, so ist der Reset-Knopf am Bot manuell zu betätigen, sowie der Reset-Button im Applet zu aktivieren. Es wird dann versucht eine Verbindung zum Bootloader herzustellen. In einigen Fällen wurde beobachtet, dass, obwohl eine Verbindung zustande kam, die Bestätigungsmeldung in der *Console* nicht erfolgte. Ein Versuch den Schreibvorgang dennoch zu starten schadet nicht. Die Schreibroutinen fragen den Mikrocontroller-Typ vom Bootloader ab. Antwortet dieser nicht, bzw. gibt einen nicht unterstützten Typ an, so bricht der Schreibvorgang ab. In diesem Fall muss das Flashen über den ISP-Adapter erfolgen (siehe Anhang C.1).

In dem Test-Applet sind alle Funktionen, die für die Online-Nutzung benötigt werden, integriert worden. Mit diesem Applet kann die Funktionsfähigkeit der Hard- und Softwarekomponenten nachgewiesen werden. Es bietet dafür komfortable Möglichkeiten zur Überwachung, Steuerung und Programmierung des c’t-Bots über eine Netzwerkverbindung. Darüber hinaus gestattet es eine erste Verwendung für Lehrzwecke. Auch sind damit schon Experimente und Tests vor Ort möglich.

[11] Das von Intel stammende hex-Dateiformat wird zum Datenaustausch mit Programmiergeräten oder zur Übertragung von Programmen verwendet.

8 Zusammenfassung und Ausblick

Mit dieser Arbeit wurden wesentliche Grundlagen für den Aufbau eines „Remote Engineering“ Praktikums für mobile Roboter gelegt. Auch für eine weiterführende Verwendung des c’t-Bots im Rahmen des „Ad-hoc Labs“ liefert diese Arbeit notwendige Vorarbeiten.

Mit der Akku-Überwachung, der Ladestation sowie den dafür erforderlichen Modifikationen wurden alle erforderlichen Hardwarekomponenten, zur Sicherung der Energieversorgung des c’t-Bots, realisiert. Diese hardwareseitigen Erweiterungen wurden in die Software des Roboters eingebunden. Neben den Arbeiten an der Hardware wurde auch die Software für den c’t-Bot überarbeitet und erweitert, so dass eine Verwendung als ferngesteuerter mobiler Praktikumsroboter möglich wird. Dazu zählen unter anderem die Funktionen zum Flashen des Mikrocontrollers über eine Netzwerkverbindung sowie die Verhaltensroutine zur Suche und zum Auffinden der Ladestation.

Mit dem Test-Applet und dem umrandeten Spielfeld, inklusive der Ladestation, wurde eine komfortable Testplattform geschaffen. Das Spielfeld wird von einer Netzwerkkamera aufgenommen und dessen Bild im Applet angezeigt. Dadurch wurden die Überwachung, die Steuerung und die Programmierung des Roboters über eine Netzwerkverbindung möglich.

Mit der im Rahmen dieser Arbeit geschaffenen Testplattform konnte die Funktionsfähigkeit der Hard- und Softwarekomponenten demonstriert werden. Sie gestattet darüber hinaus eine Vorführung und Demonstration von Remote Engineering Aspekten, mittels des c’t-Bots, in der Lehre. Desweiteren sind damit beaufsichtigte Versuche zur Programmierung und zum Test von selbstentwickelten Verhaltensroutinen möglich.

Für den konkreten Aufbau und den Betrieb eines internetbasierten Praktikumsplatzes zur ferngesteuerten Programmierung bedarf es noch umfangreicher Entwicklungsarbeiten. Dazu gehören unter anderem die Integration in den RemoteLab-Server, die Anfertigung eines Applets für den Endbenutzer sowie die Ausarbeitung von Praktikumsaufgaben. Neben einer Aufzeichnung und Protokollierung des Praktikumsablaufes müssen auch Bewertungskriterien erarbeitet werden. Auch ist eine Planung und Einteilung der Praktikums- sowie der Ladezeiten des c’t-Bots erforderlich, um einen geordneten Ablauf der Praktika zu garantieren.

Wünschenswert wäre es weiterhin, die Simulation sowie die Kompilierung des Programms für den c’t-Bot auf den RemoteLab-Server zu verlagern. Demzufolge müsste der Praktikant nur die Datei, welche die entwickelte Verhaltensroutine beinhaltet, auf den RLS hochladen. Der übrige Code, bzw. die benötigten Bibliotheken, können da-

durch fest vorgegeben werden. Der Vorteil wäre eine Verbesserung der Betriebssicherheit, da so die Grundfunktionen, wie Programmierung über die Netzwerkverbindung oder Anfahrt der Ladestation, nicht abgeschaltet bzw. verändert werden können. Desweiteren bliebe dem Praktikanten die Installation der Entwicklungsumgebung erspart. Die Verhaltensroutinen können dann mit einem einfachen Texteditor geschrieben werden.

Abgesehen von den Entwicklungsarbeiten zum Aufbau und Betrieb des Praktikumsplatzes sind auch weiterführende Modifikationen am Roboter vorstellbar.

Neben einer Reduzierung des Energiebedarfes der Akku-Überwachung wäre eine Kapazitätsabschätzung und Angabe der Restlaufzeit des Akkupacks denkbar. Überlegenswert wäre auch eine Verwendung der Akku-Überwachung zur Programm-Überwachung des c't-Bots. Dazu kann der verwendete ATmega48 als externer Watchdog benutzt werden. Wenn das Programm des c't-Bots nicht regelmäßig den Zähler über den I²C-Bus zurücksetzt, wird der Roboter neu gestartet. Die Anweisung zum Zurücksetzen des Zählers kann z. B. in der Main-Schleife des Bots erfolgen.

Desweiteren können die beschriebenen Ansätze zur Motorregelung weiterentwickelt und umgesetzt werden. Eventuell kann diese Aufgabe im Rahmen eines Spezialpraktikums angeboten werden.

Da viele Nutzer die Entwicklung des Programmcodes für den c't-Bot weiter vorantreiben, sollte in Abständen überprüft werden, ob erweiterte oder verbesserte Versionen angeboten werden. Dieser Code kann nach den Modifikationen, welche in Kapitel 6 beschrieben sind, benutzt werden.

A Einrichtung der Entwicklungsumgebung

A.1 Softwareinstallation

Die Installation der für die Entwicklung benötigten Software wird in der Installationsanleitung auf der WEB-Seite zum Bot ausführlich beschrieben [Heise_Wiki]. Alle benötigte Software ist frei erhältlich.

Benötigt werden folgende Programme:

- Java JDK5
- Eclipse
- CDT (C Development Toolkit für Eclipse)
- WinAVR

Für Nutzung der Simulationsumgebung wird zusätzlich benötigt:

- Java3D
- GCC

Für die einmalige Installation des Bootloaders sowie setzten der Fuse-Bits wird ein Flash-Programm benötigt. Empfohlen wird dazu:

- Ponyprog bzw. Avrdude.

Für die Einrichtung des WiPort wird die Herstellersoftware benötigt:

- Lantronix Device-Installer

A.2 Import des Codes in Eclipse

Der Code für den c't-Bot und das Applet befindet sich als Eclipse-Workspace auf der CD zu dieser Arbeit. Nach dem Kopieren des Workspaces müssen in Eclipse nur noch die Verzeichniseinträge angepasst werden.

Der Quelltext für den Simulator wird am besten aus dem SVN-Repository [Heise_Wiki] importiert. Eine Anleitung dazu befindet sich ebenfalls im Wiki. Zusätzlich befindet sich der Code als Eclipse-Projekt mit auf der CD.

B Hardwarekonfiguration

B.1 Fuse-Bits

Die Fuse-Bits der Atmel Mikrocontroller sind globale Konfigurierungsschalter. Mit diesen lassen sich bestimmte Funktionen einstellen, wie z. B. Reset bei Unterspannung (Brown-out-Detection), Taktauswahl, Schutz des Speichers gegen Lesen/Schreiben und Bootoptionen. Eine richtige Konfiguration ist daher grundlegend. Für den verwendeten ATmega644 werden folgende Einstellungen empfohlen (siehe Abb. 36)(Lock Bits FF, efuse FD, hfuse DC, lfuse FF).

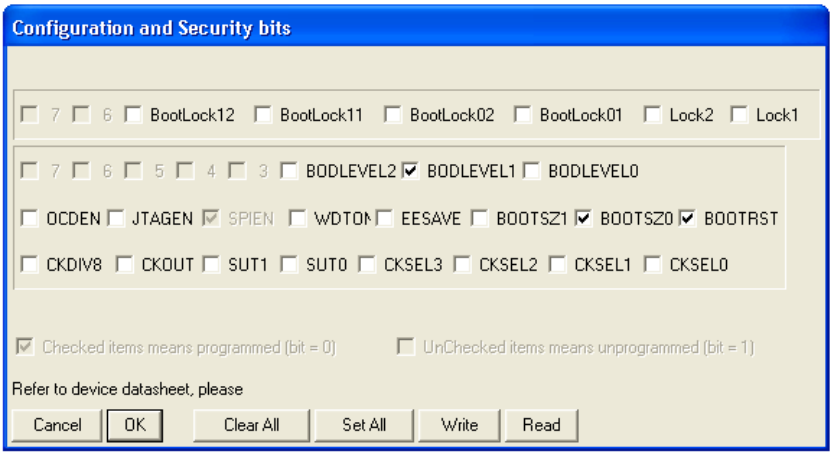


Abbildung 36: Konfiguration der Fuse-Bits -- ATmega644

Für den ATmega48 der Akku-Überwachung wurde folgende Konfiguration verwendet (siehe Abb. 37)(Lock Bits FF, efuse 1, hfuse DE, lfuse E2).

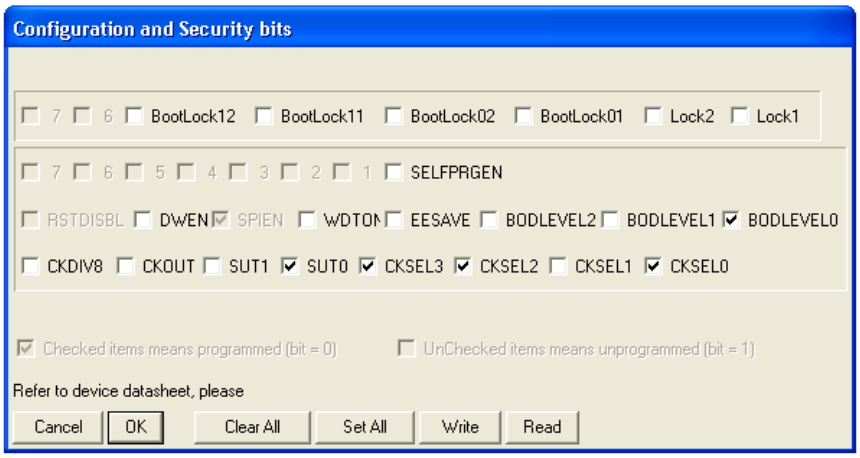


Abbildung 37: Konfiguration der Fuse-Bits -- ATmega48

B.2 WiPort-Einstellungen

Die Einstellung des WiPort-Moduls [Lantronix] lässt sich dank integriertem Webserver komfortabel über HTML-Seiten erledigen. Die Einstellmöglichkeiten sind recht vielfältig. Allerdings kann man ihn auch so verstellen, dass nichts mehr funktioniert. Deshalb ist hier besondere Aufmerksamkeit geboten. Um eine unbefugte Manipulation vorzubeugen, ist das Modul mit einem Passwortschutz versehen.

„Ein fabrikneuer WiPort lauscht entweder auf dem LAN- oder dem WLAN-Interface auf erste Kommandos. Das Programm Device-Installer von Lantronix hilft bei der ersten Konfiguration. Steht der WiPort auf LAN verbindet man ihn wie einen gewöhnlichen PC mit dem Netzwerk oder schließt ihn mit einem gekreuzten Netzwerkkabel direkt an den PC an. Steht er auf WLAN, kann man versuchen, ihn im Adhoc-Modus zu erreichen. Voreingestellt ist der Netzwerk-Name "LTRX_IBSS" und Kanal 11. Jegliche Verschlüsselung ist deaktiviert. Normalerweise findet der Device-Installer den WiPort, sobald man ihm einen Suchauftrag erteilt hat. Sollte dies einmal nicht klappen, so kann man über den Knopf "Assign IP" den WiPort auch direkt ansprechen. Dabei fragt das Programm nach der MAC-Adresse die ein Aufkleber auf dem WiPort verrät. Sobald man dem WiPort eine IP-Adresse zugewiesen hat, kann man ihn auch direkt per HTTP ansprechen und konfigurieren.“ [Hei06]

Die Einstellungen für Netzwerkadresse und WLAN sollten keine Probleme bereiten. „Radio Power Management“ sollte aber unbedingt ausgeschaltet werden, da es sonst zu erheblichen Verbindungsproblemen kommen kann.

Kanal 1 sollte so konfiguriert sein, dass im Notfall eine Konfiguration über diesem Kanal mit dem USB-2-Bot-Adapter möglich ist. Dazu in Serial Settings die Einstellungen überprüfen. Benötigt werden 9600 Baud (8N1) ohne Flusskontrolle.

Für Kanal 2, der die Verbindung zur MCU des Bot’s übernimmt, können die Einstellungen den Abbildungen 38 und 39 entnommen werden. Benötigt werden hier 57600 Baud (8N1) ohne Flusskontrolle. Der lokale Port ist auf 10002 zu setzen.

Falls mal doch was nicht so läuft hier noch ein Tipp aus dem FAQ zum Bot.

„Der Lantronix Device-Installer findet den WiPort nicht (mehr). Wie komme ich an den Baustein heran? Meist stimmen in diesem Fall die Netzwerkeinstellungen des WiPort nicht. Da der Baustein entweder per WLAN oder per LAN erreichbar ist, führt beispielsweise ein falscher WPA-PSK-Schlüssel schnell dazu, dass der WiPort nicht mehr im Netz zu sehen ist. Über den USB-2-Bot-Adapter kommt man aber wieder ins Konfigurationsmenü des WiPort. Dazu verbindet man den Adapter mit J12 -- Pin 1 liegt auf der dem Transportfach abgewandten Seite. Nun nimmt man ein Terminalprogramm wie das Windows-eigene Hyperterm und stellt die Übertragungsrate auf 9600 Baud (8N1). Die Flusskontrolle schaltet man ab. Hält man nun während dem Einschalten des c't-Bot die x-Taste auf der PC-Tastatur gedrückt, sollte im Terminalprogramm das Text-

Menü zur Konfiguration des WiPort auftauchen. Hier kann man Punkt für Punkt alle Einstellungen kontrollieren und verändern. Der WiPort erwartet, dass man jede Eingabe mit "Enter" bestätigt.“ [Hei06]

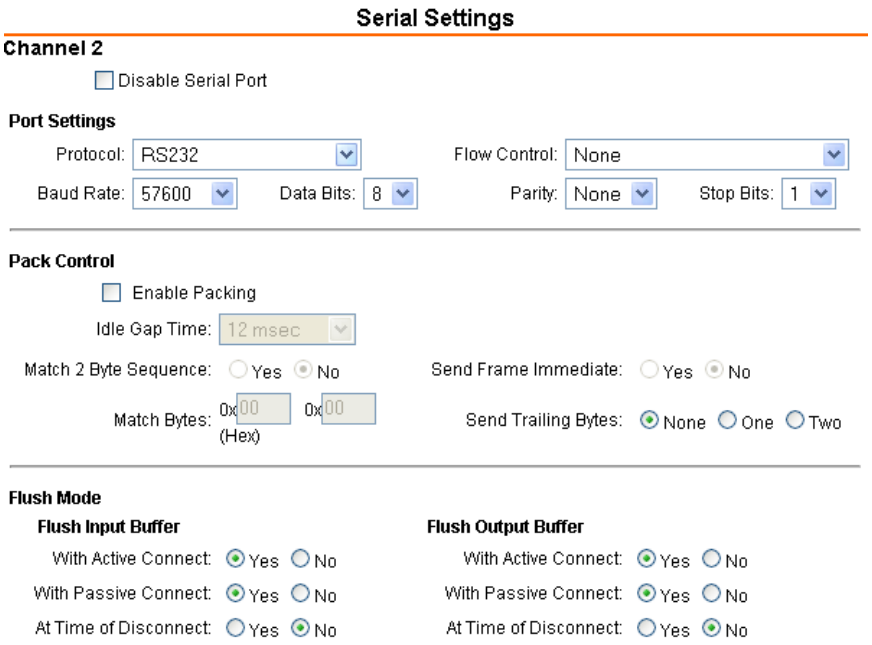


Abbildung 38: Konfiguration WiPort -- Serial Settings

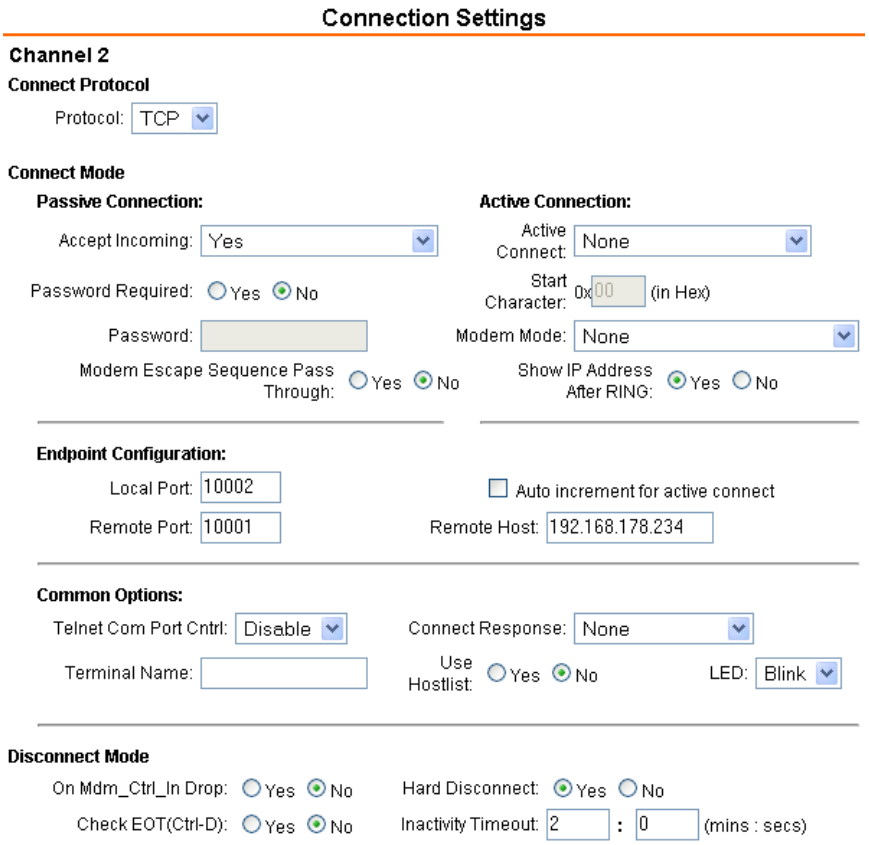


Abbildung 39: Konfiguration WiPort -- Connection Settings

B.3 Konfiguration der Ladestation

Die Programmierung bzw. Einstellung des Ladecontrollers erfolgt mit den Jumpern JP0 bis JP3. Mit den Jumpern JP0 und JP1 wird die Zellenzahl eingestellt. Wenn keine anderen Akkupacks geladen werden sollen darf hier nichts verändert werden. Die Schaltung ist auf 5 Zellen gejumpert. Die Programmierung der maximalen Ladezeit erfolgt durch Setzten der Jumper JP2 und JP3. Nach Abbruch der Ladung geht die Schaltung automatisch in Erhaltungsladung über.

Tabelle 3: Konfiguration der Ladestation -- Angabe der Zellenanzahl

Zellenanzahl	JP1	JP0
1	2-3	1-2
2	NC[12]	1-2
4	1-2	1-2
5	**2-3**	**NC**
6	NC	NC
8	1-2	NC

Tabelle 4: Konfiguration der Ladestation -- Einstellung der maximalen Ladezeit

Ladezeit	JP3	JP2
45 min	NC	1-2
66 min	NC	2-3
90 min	**1-2**	**1-2**
132 min	1-2	2-3

[12] engl. Not connected; Pins nicht verbunden

Tabelle 5: Konfiguration der Ladestation -- Dimensionierung

Größe	Dimensionierung
Versorgungsspannung (Gleichspannung) Angabe bezieht sich auf V+ (siehe Schaltplan)	mindestens 6V 1,5 V + 1,9 V * Zellenanzahl
Ladestrom Ifast in A	Ifast = 0,25 V/(RS1..4)
Ladestrom Erhaltungsladung	I = Ifast / 16 (JP3 gesetzt) I = Ifast / 32 (JP3 offen)
Ladezeit T in h	T = Kapazität in Ah / Ifast

Tabelle 6: Zustände, Ursachen und Anzeige der Ladestation

LED	Zustand	mögliche Ursachen
alle aus	aus	keine Versorgungsspannung
LED1 gelb an	aus	Schaltung durch Bot deaktiviert (ENA_C auf Masse), Sicherung defekt
LED1 gelb und LED2 grün an	Ladeschaltung aktiv Erhaltungsladung	Ladung beendet, Temperaturüberschreitung, kein Akku
alle an	Schnellladung aktiv	

C Übertragen der Firmware

C.1 Flashen des c't-Bots

Für die erstmalige Installation oder bei Verbindungsproblemen mit dem Bootloader kann die Software auch mit einem In-System Programmer (ISP) direkt in den Mikrocontroller übertragen werden. Die Buchse ST6 für diesen Anschluss befindet sich auf der Hauptplatine des Roboters auf der linken Seite direkt hinter den LEDs.

Als Programmer Software empfiehlt sich PonyProg2000. Die derzeitige Version erkennt aber den ATmega644 nicht richtig. Der Schreib- und Lesevorgang wird aber korrekt ausgeführt wenn man die Fehlermeldungen ignoriert.

Alternativ kann auch avrdude verwendet werden, welches mit WinAVR mitgeliefert wird. Für avrdude sieht die Kommandozeile in etwa so aus:

```
avrdude -p m644 -c stk200 -P lpt1 -U flash:w: ct-Bot.hex :i -e -F -v -E reset
```

C.2 Flashen der Akku-Überwachung

Die Übertragung des Programmes für die Akku-Überwachung erfolgt ebenfalls mit einem ISP. Auf dem Board ist aber nicht ein zehnpoliger Stecker wie beim c't-Bot integriert. Anstelle dessen wurde eine sechspolige Stiftleiste verwendet. Deshalb ist zum Flashen des ATmega48 das beiliegende Adapterkabel zu verwenden.

Ansonsten erfolgt der Flash-Vorgang wie beim c't-Bot. Bei Verwendung von avrdude sieht die Kommandozeile in etwa so aus:

```
avrdude -p m48 -c stk200 -P lpt1 -U flash:w: accu.hex:i -e -F -v -E reset
```

C.3 Installation eines Applets auf dem WiPort

Eine gute Anleitung dazu befindet sich im Wiki unter
„Applet kompilieren und installieren“ [Heise_Wiki]

Danach lässt sich das Applet wie folgt installieren:

- Den DeviceInstaller starten und Search klicken und den Roboter auswählen
- Den erschienenen Listeneintrag wählen und Upgrade
- "Create a custom installation by specifying ...", Next, nochmal Next
- "Install files individually", Next
- "Add Files", alles aus applet-build wählen, Next
- Im folgenden Schritt kann man nochmal alles auf Vollständigkeit kontrollieren: Weil das WiPort-Modul nur maximal 64 KiB große Dateien speichern kann, werden größere Dateien vom DeviceInstaller ignoriert. Das kann man daran erkennen, dass sie in dieser Liste nicht mehr angezeigt werden. Es macht nichts, wenn der DeviceInstaller mehrere Dateien in eine Partition legt.
- Next, Next, und warten bis alles hochgeladen ist.
- Im Browser die IP (oder Hostname) des Bot eingeben. Es sollte eine index.html geladen werden, welche das Applet startet.

Die jar-Archive müssen unbedingt signiert werden, da nur so ein Zugriff auf die Festplatte möglich ist, welcher zur Auswahl der HEX-Dateien benötigt wird. Nach der erstmaligen Installation des Applets muss auch das Webinterface des WiPorts wieder installiert werden, da es durch das Applet überschrieben wird.

D Softwareschnittstellen

D.1 Globale Variablen c't-Bot

Für die Verhaltensprogrammierung stehen unter anderem folgende globale Variablen zur Verfügung.

Tabelle 7: Auswahl globaler Variablen des c't-Bots

int16	target_speed_l	Sollgeschwindigkeit linker Motor
int16	target_speed_r	Sollgeschwindigkeit rechter Motor
float	v_left	Geschwindigkeit rechtes Rad
float	v_rigt	Geschwindigkeit linkes Rad
float	v_center	Geschwindigkeit im Zentrum
int16	sensDistL, sensDistR	Distanz in mm [100 bis 800]
int16	sensLDRL, sensLDRR	Lichtsensor [0-1023] 1023 = dunkel
int16	sensBorderL, sensBorderR	Abgrundsensor [0-1023] 1023 = dunkel
int16	sensLineL, sensLineL	Liniensensor [0-1023] 1023 = dunkel
uint8	sensTrans	Transportfach [0-1] 1 = Objekt im Tr.
uint8	sensDoor	Klappe [0-1] 0 = Klappe geschlossen
uint16	sensVoltage	verfügbare Motorspannung in mV
uint16	sensCurrent	momentaner Strom des Bot in mA

Die Geschwindigkeiten sind wie folgt definiert:

```
#define BOT_SPEED_MIN     50   /*!< langsame Fahrt in mm/s */
#define BOT_SPEED_SLOW    50   /*!< langsame Fahrt in mm/s */
#define BOT_SPEED_FOLLOW  70   /*!< vorsichtige Fahrt, fuer
                                    Folgeverhalten in mm/s */
#define BOT_SPEED_MEDIUM  100  /*!< mittlere Fahrt in mm/s */
#define BOT_SPEED_NORMAL  150  /*!< normale Fahrt in mm/s  */
#define BOT_SPEED_FAST    300  /*!< schnelle Fahrt in mm/s */
#define BOT_SPEED_MAX     450  /*!< maximale Fahrt in mm/s */
```

D.2 Übersicht aller verfügbaren Verhalten für den c't-Bot ¶

Stand: Version [1238]

Die Verhalten werden jeweils in available_behaviours.h ein- oder ausgeschaltet. Eingeschaltete Verhalten müssen dann zur Laufzeit noch aktiviert werden. Nähere Informationen zu den jeweiligen Verhalten findet man in den zugehörigen Source-Code-Dateien im Unterverzeichnis bot-logic.

Demo-Verhalten

Diese kleinen Verhalten sind ideal zum Einstieg oder um den Bot ganz einfache Dinge vorführen zu lassen.

BEHAVIOUR_SIMPLE: Zwei sehr einfache Beispielverhalten, die für den Einstieg gedacht sind. In behaviour_simple.c findet man genauere Infos.

BEHAVIOUR_DRIVE_SQUARE: Ein weiteres sehr einfaches Verhalten, das den Bot endlos im Quadrat fahren lässt. Eine nette kleine Demo, aber auch geeignet um die Genauigkeit von Drehungen und Geradeausfahrten zu überprüfen.

Notfall-Verhalten

Diese Verhalten greifen nur ein, wenn ein *Notfall* erkannt wird, z. B. ein Abgrund oder Hindernis voraus.

BEHAVIOUR_AVOID_BORDER: Sobald ein Abgrund erkannt wird, fährt der Bot rückwärts, um nicht in den Abgrund zu stürzen. Es müssen auf jeden Fall die Abgrundsensoren korrekt kalibriert sein, damit das auch zuverlässig funktioniert!

BEHAVIOUR_AVOID_COL: Wird mit den Distanzsensoren ein Hindernis in Fahrtrichtung erkannt, dreht dieses Verhalten den Bot und versucht am Hindernis vorbeizufahren. Damit das auch klappt, müssen die Distanzsensoren korrekt kalibriert sein!

BEHAVIOUR_HANG_ON: Mit Hilfe des Maussensors kann dieses Verhalten ein Hängenbleiben des Bots erkennen und Gegenmaßnahmen einleiten (z. B. rückwärts fahren). Der Maussensor muss dafür natürlich korrekt kalibriert sein!

Positionierungs-Verhalten

Die folgenden Verhalten haben die Aufgabe den Bot an einen anderen Ort zu befördern oder ihn zu drehen. Sie kapseln somit die low-level-Funktionen zur Bewegung und ermöglichen den Anwendungs-Verhalten einen abstrakten und einfachen Zugriff auf die Positionierung des Bots.

BEHAVIOUR_GOTO: Eines der ersten Positionierungsverhalten, das den Bot eine anzugebene Anzahl an Schritten fahren lässt. Inzwischen ist es aber nicht mehr up to date, man sollte lieber *drive_distance()* oder *gotoxy()* benutzen.

BEHAVIOUR_DRIVE_DISTANCE: Dieses Verhalten lässt den Bot eine in cm vorgegebene Distanz fahren. Dabei lässt sich optional auch eine Krümmung vorgeben, falls der Bot nicht geradeaus fahren soll. Ebenso kann man die Geschwindigkeit vorgeben.

BEHAVIOUR_GOTOXY: Um den Bot an die Koordinaten (x|y) *seines* Systems zu bringen, verwendet man dieses Verhalten. Zunächst wird der Bot in die Zielrichtung gedreht und anschließend die Strecke zum Ziel abgefahren.

BEHAVIOUR_TURN: Dieses Verhalten dreht den Bot um die gewünschte Gradzahl (im mathematischen Drehsinn gerechnet => positive Zahlen bewirken eine Drehung gegen den Uhrzeigersinn).

BEHAVIOUR_TURN_TEST: Ein kleines Testverhalten für *bot_turn()*, das einige Drehungen ausführt und dabei Debug-Infos per Log ausgibt, so dass sich die Genauigkeit der Drehungen überprüfen lässt.

Anwendungs-Verhalten

Die Anwendungs-Verhalten befinden sich auf der höchsten Schicht des Verhaltenssystems und lassen den Bot komplexe Aufgaben ausführen. In ihnen steckt die meiste *Intelligenz* des Bots.

BEHAVIOUR_SOLVE_MAZE: Löst ein Labyrinth mit der Wandfolger-Methode. Das Labyrinth muss ein paar Voraussetzungen erfüllen, damit das funktioniert.

BEHAVIOUR_FOLLOW_LINE: Mit diesem Verhalten folgt der Bot einer Linie auf dem Untergrund. Natürlich müssen die Liniensensoren korrekt kalibriert sein!

BEHAVIOUR_MAP_GO_DESTINATION: Wenn die Kartographie aktiv ist, steuer dieses Verhalten den Bot zum angegeben Ziel und beachtet dabei die Hindernisse in der Karte.

BEHAVIOUR_OLYMPIC: Dieses Verhalten sucht eine Lichtquelle, fährt auf sie zu und anschließend im Slalom um weitere Lichtquellen.

BEHAVIOUR_CATCH_PILLAR: Ein Verhalten, mit den Distanzsensoren in der näheren Umgebung nach einer Dose sucht und sie in das Transportfach lädt. Es gibt auch ein weiteres Verhalten, das die Dose wieder auslädt (*bot_unload_pillar()*).

BEHAVIOUR_FOLLOW_OBJECT: Mit diesem Verhalten verfolgt der ein (bewegliches) Objekt, solange es sich im Blickfeld befindet.

BEHAVIOUR_FOLLOW_WALL: Ein Verhalten, das den Bot eine Wand suchen lässt, indem er auf sie zu fährt. Anschließend erfolgt eine kleine Drehung und das Spiel beginnt von vorn. Alternativ lässt sich auch eine Abbruchfunktion angeben, die das Verhalten beendet, sobald sie True liefert.

BEHAVIOUR_SCAN: Um in der Karte die nähere Umgebung einzutragen, dreht dieses Verhalten einen den Bot im Kreis und scannt mit den Distanzsensoren dabei die Umgebung. Außerdem wird durch die Komponente *bot_scan_onthefly()* die Karte bei Bewegungen automatisch aktualisiert.

Kalibrierungs-Verhalten

Die Kalibrierungs-Verhalten vereinfachen die Einstellung einiger Parameter, die für die jeweiligen Bots unterschiedlich sein können. Normalerweise führt man sie nach der Inbetriebnahme einmalig aus und braucht die ermittelten Parameter später nicht mehr zu ändern.

BEHAVIOUR_CALIBRATE_PID: Dieses Verhalten kalibriert die Motorregelung automatisch durch systematisches Probieren von Parametern. Achtung, lange Laufzeit (Restlaufzeit wird im Display angezeigt).

BEHAVIOUR_CALIBRATE_SHARPS: Dieses Verhalten kalibriert die Distanzsensoren halbautomatisch, indem man den Bot fortlaufend auf die im Display angezeigte Entfernung stellt und anschließend eine Taste drückt. Anschließend werden die ermittelten Daten im EEPROM abgelegt.

System-Verhalten

Die System-Verhalten sind auf der untersten Schicht des Verhaltenssystems angesiedelt und arbeiten sehr direkt mit den Grundfunktionen des Bot-Frameworks zusammen. Sie lösen keine komplizierten Aufgaben, sondern bieten einen einfachen Zugriff auf systemnahe Funktionen.

BEHAVIOUR_SERVO: Mit diesem Verhalten lassen sich bis zu zwei Servos ansteuern. Es sorgt dafür, dass die Servos nach Erreichen ihrer Zielposition wieder abgeschaltet werden, um Strom zu sparen.

BEHAVIOUR_REMOTECALL: Umfangreiches Verhalten, über das andere Verhalten aus der Ferne aktiviert und auch wieder abgebrochen werden können. Dazu muss der Bot entweder per USB-2-Bot-Adapter oder (W)LAN mit einem PC verbunden sein. Der c‘t-Sim oder das Applet für den WiPort ermöglichen dann eine einfache Steuerung von Verhalten und sogar Verhaltensfolgen. Sozusagen der große Bruder der IR-Fernbedienung.

BEHAVIOUR_MEASURE_DISTANCE: Ein Hilfsverhalten, dass die Messwerte der Distanzsensoren ausliest und jene erst dann zurückliefert, wenn sie auch stabil (gleichbleibend) sind.

BEHAVIOUR_DELAY: Ein Hilfsverhalten, dass aktive Verhalten (mit Ausnahme der Notfall-Verhalten) unterbricht, bis eine gewünschte Zeitspanne angelaufen ist. [Heise_Wiki]

Erweitert wurde die Liste durch:

BEHAVIOUR_EXAMPLE: Ein weiteres Beispielverhalten, welches die Arten der Geschwindigkeitswahl aufzeigt sowie den Aufruf vorhandener Verhalten demonstriert. Dieses Verhalten soll als Einstieg und Vorlage zur Verhaltensprogrammierung dienen.

BEHAVIOUR_SEARCH_CHARGER: Verhalten zum Suchen und Anfahren der Ladestation. Das Verhalten schaltet die Abgrunderkennung ab! Es sollte deshalb nicht außerhalb einer abgrundfreien Umgebung verwendet werden.

D.3 Schnittstelle Akku–Überwachung

Die Kommunikation zwischen der Akku-Überwachung und dem c't-Bot erfolgt über den I²C-Bus. Der ATmega48 der Überwachung arbeitet als „Slave Device". Er lässt sich dabei vom Master, dem ATmega644 auf dem Bot, ähnlich wie ein I²C-Speicher ansprechen. Dazu verfügt er über getrennte Eingangs- und Ausgangspuffer, welche vom Master beschrieben bzw. gelesen werden können.

In bat-control.h sind für die Kommunikation mit der Akku-Überwachung folgende Funktionen definiert:

```
/**
 * Schaltet Bot aus
 * param condition Einschaltbedingung
 * param delay Ausschaltzeit in Sekunden
 */
void Switch_off_Bot(uint8 condition, uint16 delay);
```

```
/**
 * gibt die Spannung der Zelle[0..4] in mV zurück
 */
int16 get_Cell_Voltage(uint8 cell);
```

```
/**
 * Auslesen und Auswerten des kompletten Datensatzes
 * setzt die globolen Werte sensVoltage und sensCurrent
.* Warnung und Stopp aller Verhalten bei Unterspannung
 * und hohen Strömen
 */
void Read_Bat_Control(void);
```

```
/*
 * Sendet Bytefolge zur Akku-Ueberwachung
 * uint8 size - Anzahl zu Uebertragener Bytes
 * uint8* tx - Adresse des Sendebuffers
 * return   uint8 state - Status der Uebertragung
 */
uint8 Send_Bat_Control(uint8 size, uint8* tx);
```

Desweiteren wurden folgende globale Variablen definiert:

```
uint8 state_charger;
    // 1- Ladestation erkannt 3 – Schnellladung aktiv (siehe Tab. 11)
uint8 state_accu;
    // soll später den Ladezustand des Akkus wiedergeben
    // Ausgabe des seit letzter Ladung abgegebener Ladung in 10 mAh
uint16 sensVoltage;
    // Eingangsspannung des c't-Bots in mV, maximale Motorspannung
uint16 sensCurrent;
    // Stromverbrauch des Bots in mA
```

Tabelle 8: Akku-Überwachung -- Belegung des Ausgangspuffers

Byte	Funktion
0-1	Strom in mA
2-3	Spannung Celle 1 in mV
4-5	Spannung Celle 2 in mV
6-7	Spannung Celle 3 in mV
8-9	Spannung Celle 4 in mV
10-11	Spannung Celle 5 in mV
12-13	Gesamtspannung in mV
14	Zustand Akku
15	Zustand Ladegerät

Tabelle 9: Akku-Überwachung -- Belegung des Eingangspuffers

Byte	Funktion
0	Ausschaltbedingung
1	Einschaltbedingung Bot
2-3	Ausschaltzeit in s

Tabelle 10: Akku-Überwachung -- Konstanten für die Einschaltbedingung

Konstanten		Funktion
SWITCH_ON_AFTER_FC	4	Einschalten nach Beendigung der Ladung
SWITCH_ON_AFTER_TIME	8	Einschalten nach angegebener Auszeit
SWITCH_ON_AFTER_BOTH	12	Einschalten nach Ladung oder Zeit

Tabelle 11: Akku-Überwachung -- Konstanten der Zustände des Ladegerätes

Konstanten		Funktion
CHARGER_DETECTED	1	Ladestation erkannt und aktiviert
FASTCHARGE	2	Schnellladung aktiv
CHARGER_DISABLED	4	Ladestation wurde durch diese Akku-Überwachung deaktiviert

E Technische Daten und Parameter

E.1 Überblick c't-Bot

Tabelle 12: Technische Daten des c't-Bots

Durchmesser	120 mm
Gesamtmasse	780 g
Geschwindigkeit	450 mm/s
Motor	FTB Faulhaber 2619-006SR (6 V Gleichstrommotor)
Sonstiges	Transportklappe, Antrieb mit Servo-Motor
Akku	5 NI-MH 2500 mAh
Prozessor	Atmel ATmega644 16 Mhz
Speicher	64 KiB Flash, 4 KiB Ram, 2 KiB EEPROM
Speicherkarte	Unterstützung von SD- und MMC-Karten
Anzeige	LCD Text-Display mit 4x20 Zeichen
WLAN	Lantronix WiPort-Modul [Lantronix]

E.2 Parameter und physikalische Größen der Motorregelung

Tabelle 13: Parameter des Motors (FTB 2619-006SR ohne Getriebe)

Parameter	Formelzeichen	Größe	Einheit
Anschlusswiderstand	R	8,2	Ω
Drehzahlkonstante	k_n	1111	$\frac{1}{minV}$
Drehmomentkonstante	k_M	8,59	$\frac{mNm}{A}$
Leerlaufdrehzahl (Motor)	n_0	6600	$\frac{1}{min}$
Rotorträgheitsmoment	J_R	0,68	gcm^2

Tabelle 14: Getriebeparameter des Motors (FTB 2619-006SR)

Parameter	Formelzeichen	Wert	Einheit
Abtriebsdrehzahl	n_{max}	151	$\frac{1}{min}$
Drehmoment Dauerbetrieb	M	30	mNm
Drehmoment Kurzzeitbetrieb	M	100	mNm
Untersetzungsverhältnis	$u_{Getriebe}$	33	
Wirkungsgrad Getriebe	$\eta_{Getriebe}$	0,6	

Tabelle 15: Masse und Parameter des c't-Bots für die Geschwindigkeitsregelung

Parameter	Formelzeichen	Wert	Einheit
Masse	m	780	g
Trägheitsmoment	J_S	2100	$kgmm^2$
Reibungskraft	F_R	240	mN
Reibungsmoment	M_R	13	Nmm
Raddurchmesser	d_{Rad}	57	mm
Radstand	$Radstand_{Bot}$	97	mm
Radius	r_{Bot}	60	mm
Stützpunktabstand	r_S	54	mm

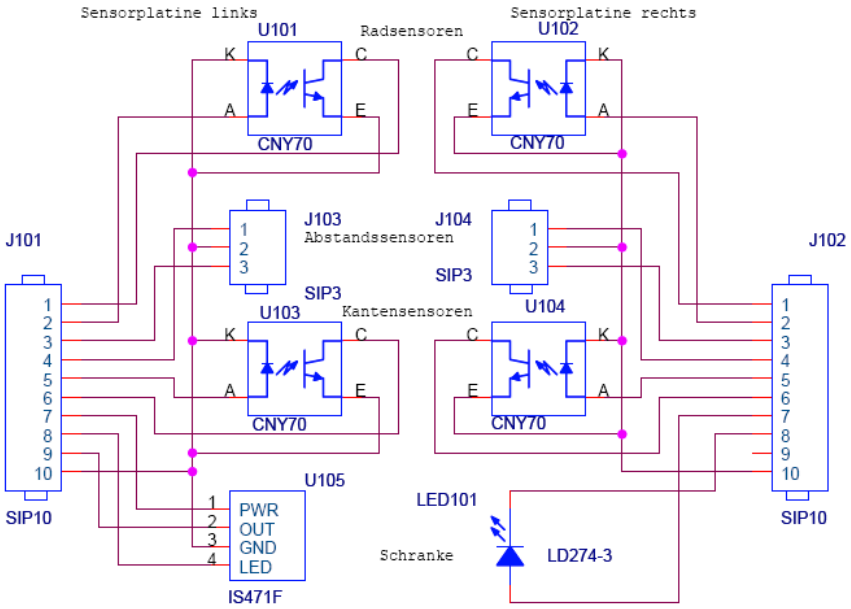
Tabelle 16: Verwendete physikalische Größen und Einheiten

Größe	Formelzeichen	Einheit
elektrische Leistung	P_{el}	W
mechanische Leistung	P_{mech}	W
Joulsche Verlustleistung	P_J	W
Drehzahl	n	min^{-1}
Motorspannung	U	V
Motorstrom	I	A
Motordrehmoment	M	mNm
Anschlusswiderstand	R	Ω
Drehmomentkonstante	k_M	mNm/A
Drehzahlkonstante	k_n	min^{-1}/V
Leistung Translation	P_{trans}	W
Leistung Rotation	P_{rot}	W
Kinetische Energie	E_{kin}	Nm
Rotationsenergie	E_{rot}	Nm
Abtastzeit	T	s
Reibungskraft	F_R	N
Reibungsmoment	M_R	Nm
Masse des Roboters	m	kg
Trägheitsmoment (Roboter)	J_S	kgm^2
Fahrgeschwindigkeit (Roboter)	v	m/s
Winkelgeschwindigkeit (Roboter)	ω	rad/s

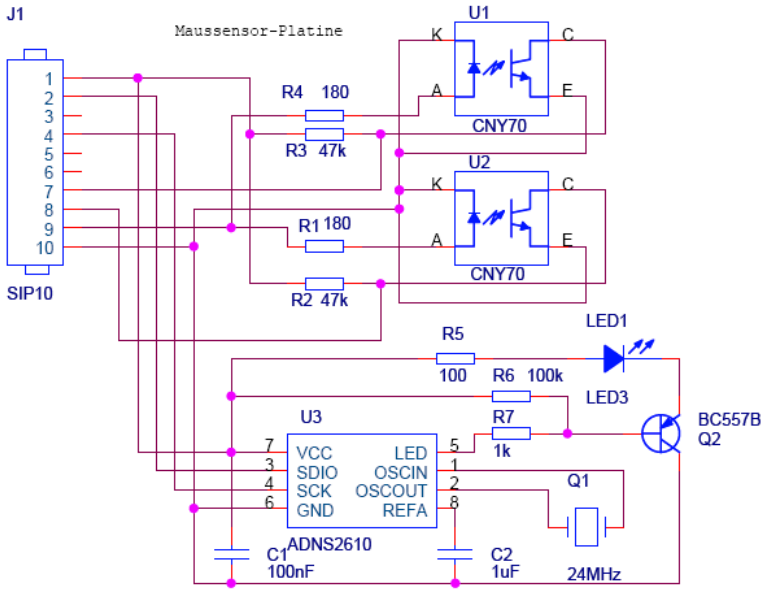
F Schaltpläne

Um eine vollständige Übersicht über die Schaltung des c't-Bots zu erhalten, sind hier alle Schaltpläne des c't-Bots, sowie der Erweiterungen dieser Arbeit, aufgeführt.

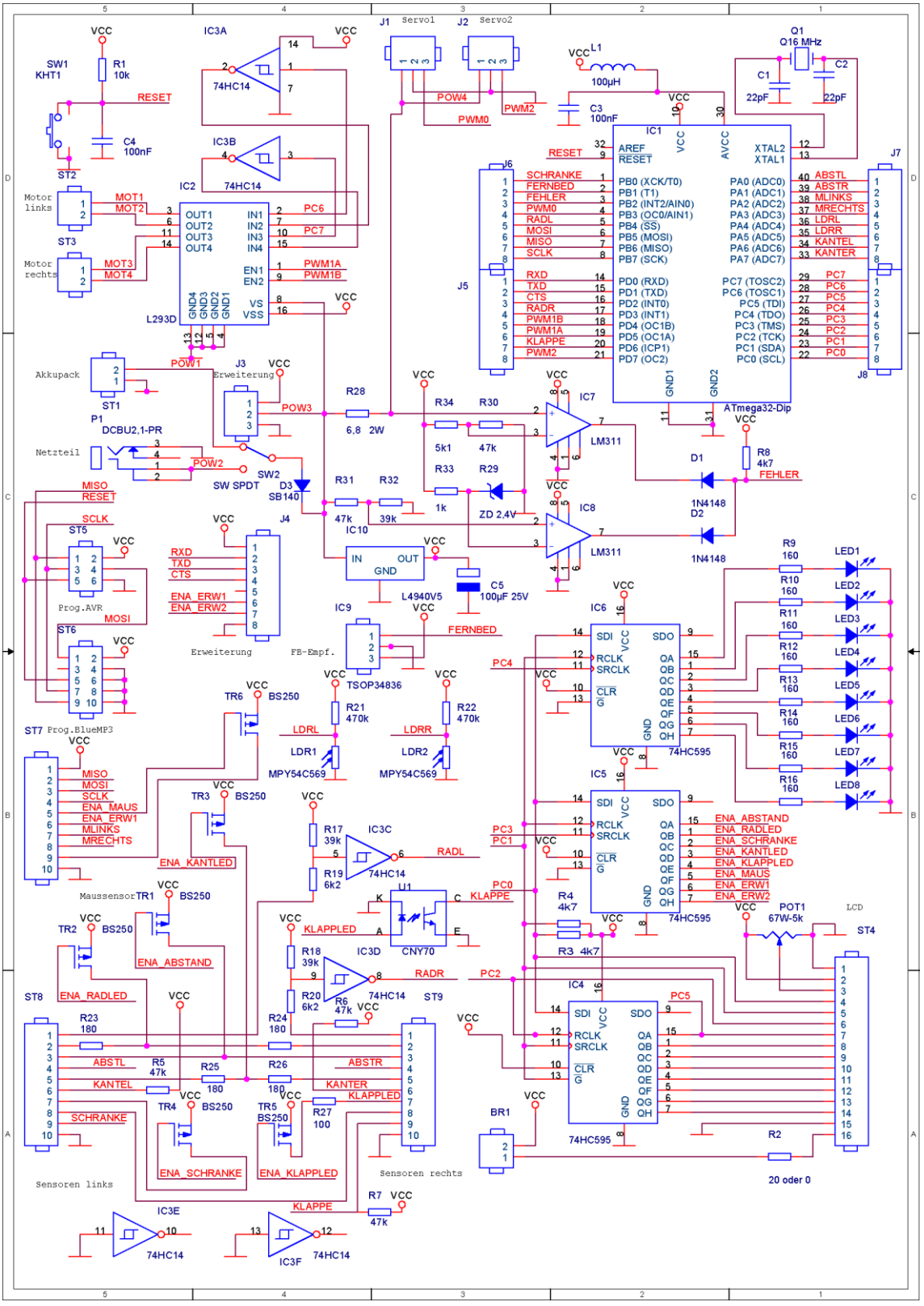
F.1 c't-Bot Sensoren



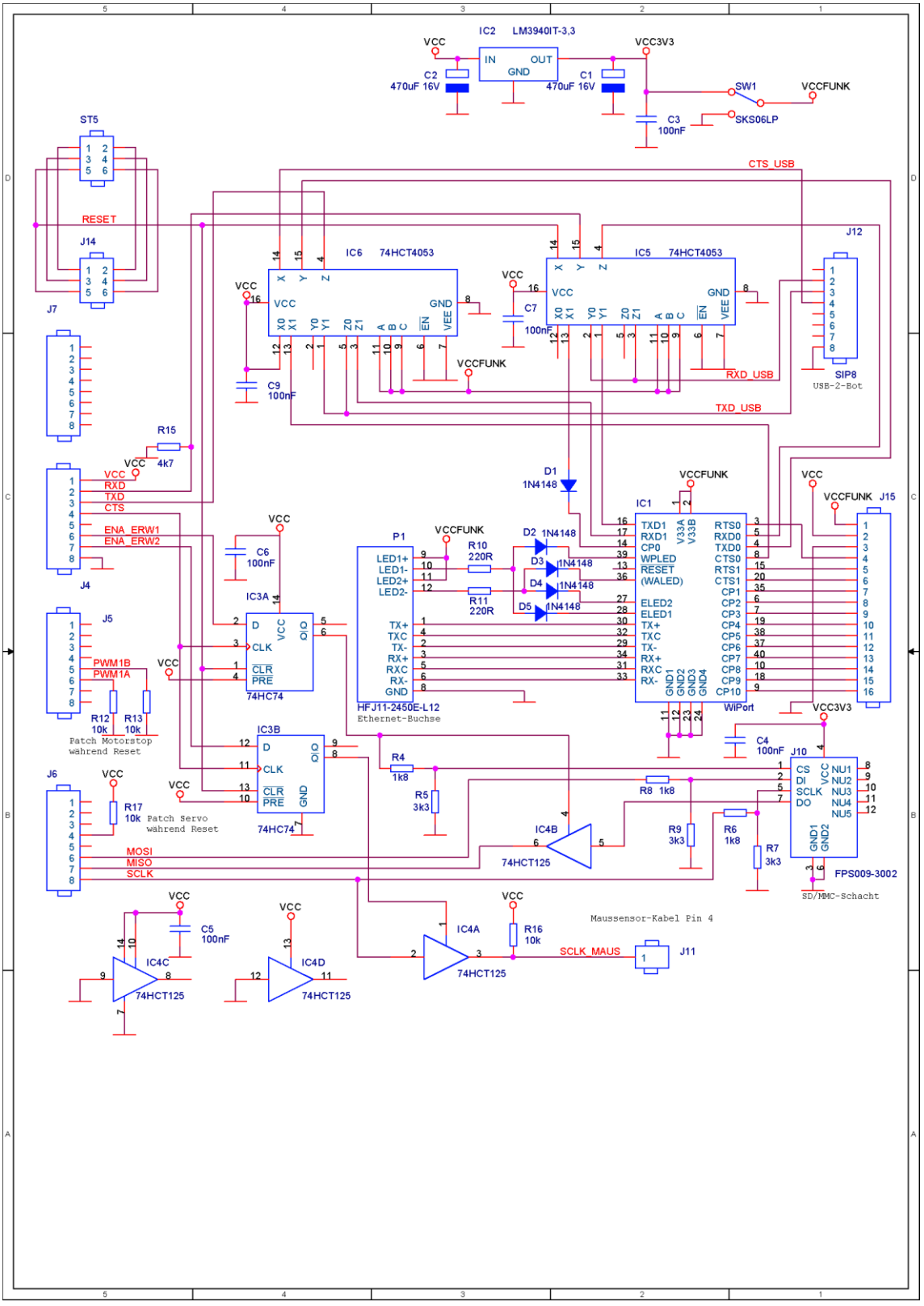
F.2 c't-Bot Maus



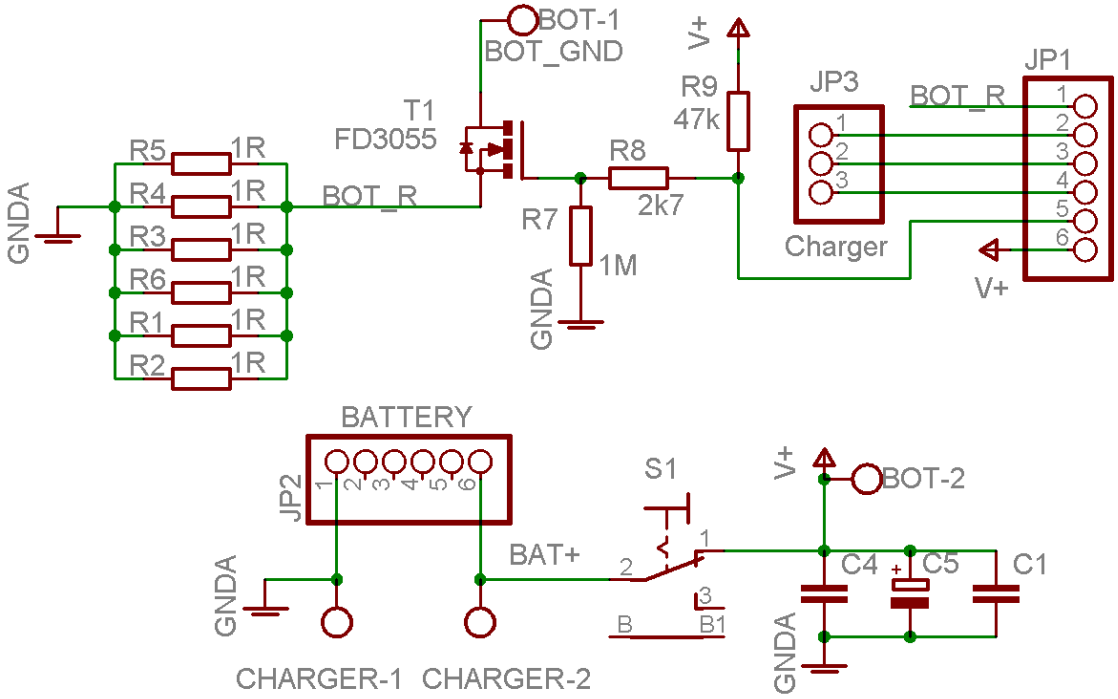
F.3 c't-Bot Main



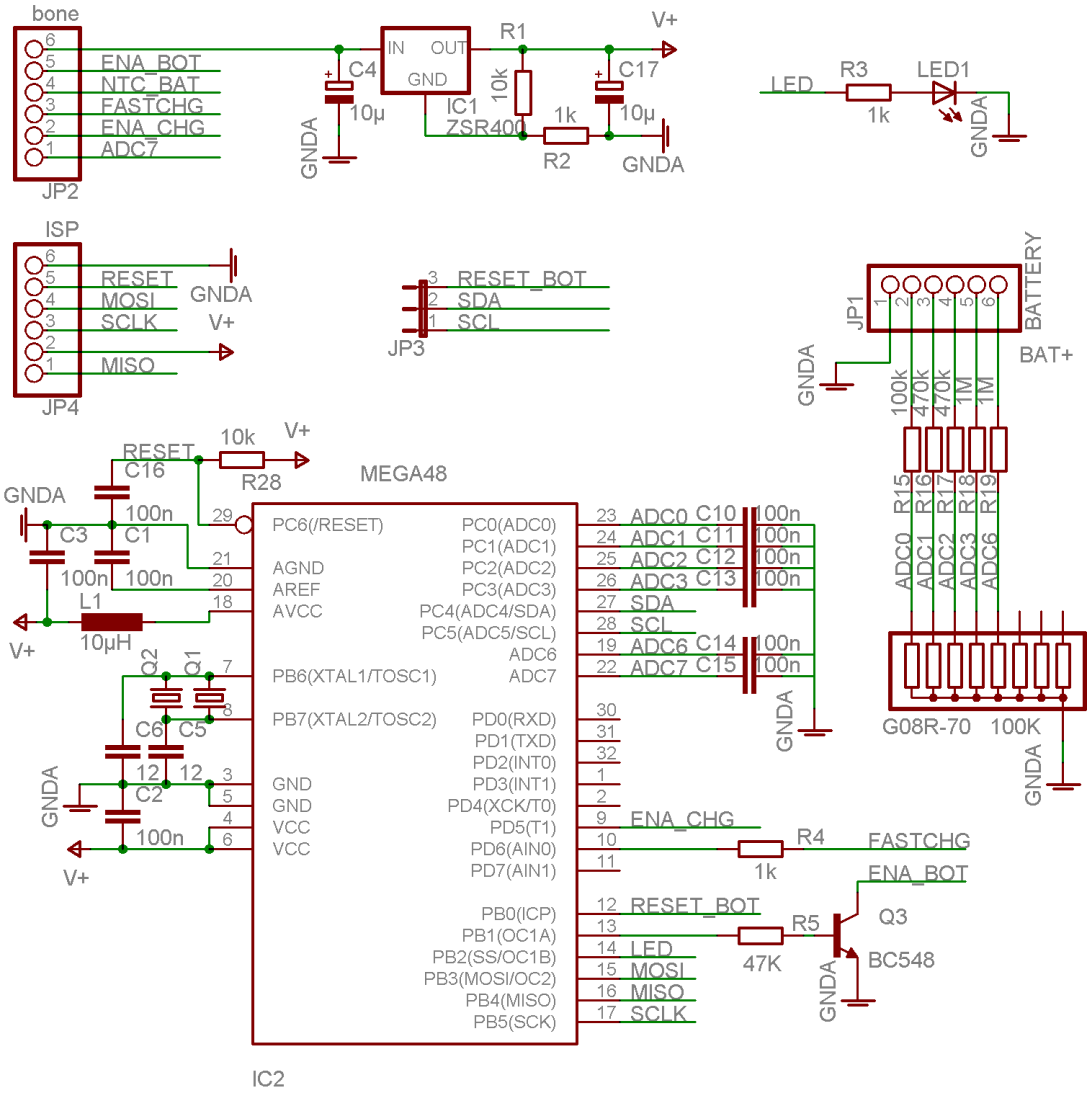
F.4 c't-Bot Erweiterungsboard (WLAN)



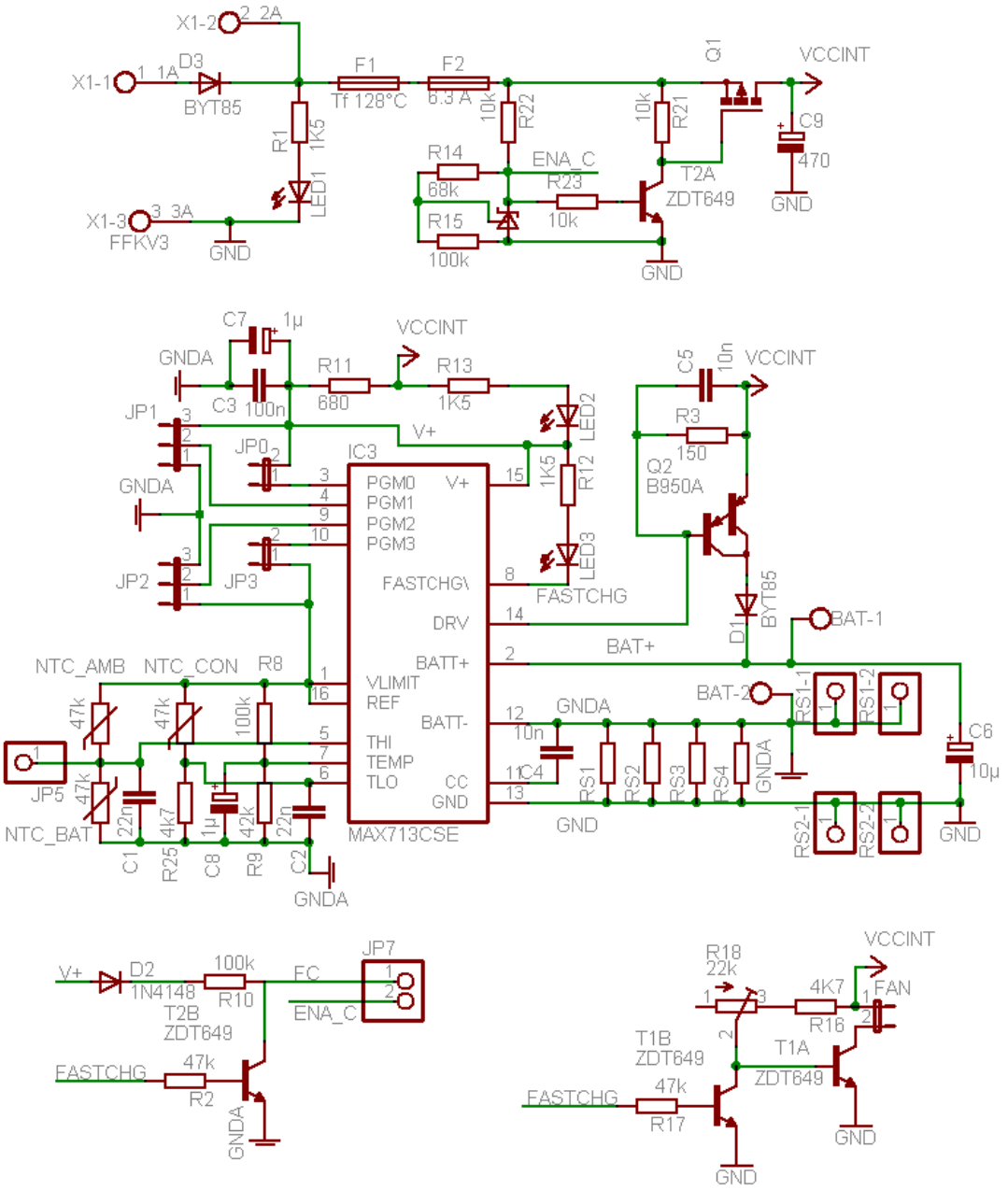
F.5 Akku–Halterung



F.6 Akku–Überwachung



F.7 Akku–Ladestation



G Inhalt der CD

- Verzeichnis **Diplomarbeit**
  - **Schmidt_Diplom_ct-Bot.pdf** Die PDF-Datei dieses Dokumentes. Die Datei ist mit dem Adobe Acrobat-Reader lesbar.
- Verzeichnis **Schaltplan**
  In diesem Verzeichnis befinden sich die mit Eagle erstellten Schaltpläne und Leiterplattenlayouts
- Verzeichnis **Datasheets**
  Datenblätter der verwendeten Mikrocontroller sowie Sensoren
  - **ATmega32.pdf**, **ATmega48.pdf** und **ATmega644.pdf**
  - **ADNS-2610.pdf** –optischer Maussensor
  - **B400-NTC-02_Serie.pdf** – NTC Temperatursensor
  - **CNY70.pdf** – Reflexlichtschranke
  - **GP2D12.pdf** – IR-Entfernungsmesser
  - **L293D.pdf** – Motortreiber
  - **TSOP34836.pdf** – IR-Fernbedienungsempfänger
  - **WiPort_UG.pdf** – Handbuch für WiPort WLAN-Modul
- Verzeichnis **Literatur**
  in diesem Verzeichnis befinden sich Kopien der wichtigsten Online-Quellen
- Verzeichnis **Source** beinhaltet den Eclipse Workspace
  - Verzeichnis **Accu-Control**
    Eclipse Projekt für die Akku-Überwachung
  - Verzeichnis **TEST-Applet**
    Eclipse Projekt für das Test-Applet
  - Verzeichnis **c't-Bot**
    Eclipse Projekt für den c't-Bot
  - Verzeichnis **c't-Sim**
    Eclipse Projekt für den Simulator

- Verzeichnis **Tools**
  Die in diesem Verzeichnis gespeicherten Werkzeuge sind Freeware bzw. unter der GPL[13] zur Veröffentlichung freigegeben.

  - **AdbeRdr810_de_DE.exe** Adobe Acrobat Reader,
    wird zum Lesen dieses Dokumentes benötigt
  - **jdk-1_5_0_14-windows-i586-p.exe** Java Development Kit 5.0
    Die Java Laufzeitumgebung wird zur Nutzung des Client-Applets sowie der Entwicklungsumgebung Eclipse benötigt.
  - **java3d-1_5_1-windows-i586.exe** Java3D Bibliothek
    Wird zur Nutzung des c't-Sim Simulators benötigt
  - **eclipse-java-europa-fall2-win32.zip** Eclipse Entwicklungsumgebung
  - **WinAVR-20070525-install.exe** WinAVR Bibliothek für AVR-Mikroprozessoren
  - **MinGW-5.0.0.exe, MSYS-1.0.10.exe, pthreads-w32-2-7-0-release.exe**
    gcc und Unix-Umgebung für Windows
    Wird zur Nutzung des c't-Sim Simulators benötigt
  - **ponyprogV207a.zip** PonyProg2000 Programmer Software
    Wird zum setzen der Fuse-Bits und zur erstmaligen Installation des Bootloaders auf den AVR-MCUs benötigt
  - **DI_4.1.0.14_Web.exe** Lantronix Deviceinstaller
    Wird zur Konfiguration des WLAN-Moduls benötigt.
  - **dotnetfx.exe** Microsoft .NET Framework v1.1
    Benötigt der Lantronix Deviceinstaller
  - **CDM 2.02.04 WHQL Certified.zip** VCP-Treiber für den USB-2-Bot-Adapter, wird zur Konfiguration des WLAN-Moduls verwendet, wenn Zugang über WLAN nicht möglich ist

[13] GPL: General Public License

H Literaturverzeichnis

[qfi07]. **qfix robotics GmbH.** [Online] www.qfix.de, 18.09.07.

[Rob07]. **Roboternetz.** *Excel-Tabelle zur Berechnung der Motorkennlinien.* [Online] http://www.roboternetz.de/phpBB2/download.php?id=3479&sid=ad377612fd5ca317c396931eeae0fb03, 25.10.2007.

[iRo07]. **iRobot.** *iRobot Create Programmable Robot.* [Online] 2007. http://www.irobot.com/sp.cfm?pageid=305, 04.07.2007.

[AREXX_ASURO]. **AREXX Engineering.** ASURO. [Online] http://www.arexx.com/arexx.php?cmd=goto&cparam=p_asuro, 18.09.2007.

[Heise_Wiki]. **c't-Bot-Wiki.** c't-Bot und c't-Sim - Trac. [Online] Heise. http://www.heise.de/ct/projekte/machmit/ctbot/wiki.

[max07]. **maxon motor gmbh.** Das wichtigste über maxon Motoren. [Online] http://www.maxonmotor.com/de/dc_motor.asp.

[SHARP]. **Sharp.** *Datenblatt GP2D12.*

[Act_4WD2]. **Active Robots.** Lynxmotion 4WD2. [Online] http://www.active-robots.com/products/platforms/lynx-4wd2.shtml,18.09.2007.

[Max02]. **Maxim.** NiCd/NiMH Battery Fast-Charge Controllers. 2002. Bde. MAX712-MAX713.pdf, Rev 5.

[AREXX_PR6]. **AREXX Engineering.** PR6 Robot System. [Online] http://www.arexx.com/rp6/html/de/index.htm, 11.12.2007.

[Act_SRV]. **Active Robots.** Surveyor SRV–1 Mobile Robot. [Online] http://www.active-robots.com/products/robots/surveyor-srv1.shtml, 18.09.2007.

[Lego]. **Lego Mindstorm.** TriBot. [Online] http://mindstorms.lego.com/Overview/MTR_Tribot.aspx, 11.12.2007.

[Lantronix]. **Lantronix .** WiPort. [Online] http://www.lantronix.com/device-networking/embedded-device-servers/wiport.html, 11.12.2007.

[Hei06]. **Benz, B., Thiele, T., Thiede, C. u.a.** *c't Projekte - c't-Bot und c't-Sim.* [Online] Heise Zeitschriften Verlag, 2006. http://www.heise.de/ct/projekte/ct-bot/, 18.09.2007.

[Fri04]. **Fried, Limor.** SourceForge.net, JAvrProg. [Online] 2004. http://sourceforge.net/projects/javrprog/, 26.09.2007.

[Hel07]. **Held, Jörg.** RC-Car Akkuladung. [Online] http://rccar.blessem.de/rccar5.htm, 11.12.2007.

[Mit07]. **Mitschele-Thiel, Prof. Dr.-Ing. habil. Andreas.** *Förderverein Antrag zum Aufbau des "Ad hoc Labs".* 2007.

[Mue07]. **Müller, Sebastian.** Möglichkeiten einer mobilen Roboterplattform. 2007.

[Ras06]. **Raschke, Michael.** iBot - Entwicklung eines selbstständigen Roboters. [Online] Universität Heidelberg, 2006. http://www.alternativesdenken.de/roboland/strukturierung.html, 18.09.2007.

[Sch07]. **Schadeck, Michael.** Dokumentation zur Inbetriebnahme des c't-Bot. s.l. : TU-Ilmenau, Februar 2007.

[Sch06]. **Schoener, Friedemann.** *Entwurf einer Arbeitsumgebung zur verteilten Entwicklung paralleler Automaten.* s.l. : Diplomarbeit, Technische Universität Ilmenau, 2006.

[Tav07]. **Tavernier.** Vielseitiger Akkulader. *Elektor.* 2007, Juli/August.

Weiterhin wurden die auf der CD beigelegten Datenblätter verwendet.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift

Thesen

- Das RemoteLab–Projekt des FG IHS benötigt einen neuen Praktikumsversuch „Mobiler Roboter“ zur Ausweitung des Angebotes und zur Durchführung von Praktika im Bereich Remote Engineering.
- Der c’t-Bot ist mit seiner umfangreichen Sensorausstattung und dem optional verfügbaren WLAN-Modul gegenwärtig die geeignetste Plattform auf dem Markt.
- Die dafür zur Verfügung stehende Software inklusive eines Simulators bietet eine gute Grundlage für darauf aufbauende Entwicklungen.
- Für die Nutzung als Praktikumsroboter sind umfangreiche Modifikationen und Erweiterungen der Hard- als auch Software notwendig.
- Die Akku-Überwachung schützt den Akkupack vor Unterspannung und liefert die benötigen Daten zur Sicherstellung der Energieversorgung.
- Der Roboter kann selbständig die Ladestation anfahren. Diese ist in der Lage den Akkupacks des Roboters in einer Stunde zu laden. Somit ist die Energieversorgung gesichert.
- Der c’t-Bot lässt sich mit Hilfe der Modifikationen und Erweiterungen der Hard- als auch Software über das Internet überwachen, steuern und programmieren.
- Das umrandete Spielfeld mit Netzwerkkamera und Ladestation bietet eine geeignete Testplattform, mit welcher die Soft- und Hardwarekomponenten des Roboters getestet werden können.
- Mit dem Test-Applet konnte der Nachweis der Funktionsfähigkeit aller Erweiterungen erbracht werden. Es zeigt darüber hinaus die Einsatzmöglichkeiten dieser Plattform und kann zur Demonstration in der Lehre verwendet werden.
- Mit dieser Arbeit wurden wesentliche Grundlagen für den Einsatz des c’t-Bots als fernbedienbaren mobilen Praktikumsroboter, als auch einer Verwendung im Rahmen des „Ad-hoc Labs“ gelegt.
- Für die konkrete Realisierung eines Remote Engineering Praktikums für mobile Roboter sind noch umfangreiche weiterführende Arbeiten erforderlich.

Ilmenau, 19.12.2007 ……

Friedemann Schmidt