Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Semesterarbeit SA-2002.26

Bluetooth Anbindung für Lego Mindstorms

Sommersemester 2002
5.7.2002

Andres Erni
Stefan Reichmuth

Betreuer : Philipp Blum
Stellvertreter : Jan Beutel

Supervisor: Prof. Dr. Lothar Thiele

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Inhaltsverzeichnis

Kapitel 1

Einleitung

Lego Mindstorms wird am Institut für Technische Informatik und Kommunikationsnetze als einfache Experimentierplattform eingesetzt. Leider sind die damit konstruierten Roboter in der Kommunikation eingeschränkt. Da dafür Infrarot verwendet wird, muss immer eine Sichtverbindung zwischen dem RCX-Baustein und dem Sende-Tower des PC's bestehen. Noch umständlicher ist die Kommunikation zwischen zwei Robotern. Wenn Daten von einem zum anderen übermittelt werden sollen, müssen sie sich zuerst in eine Position bewegen, in der die beiden Infrarot-Schnittstellen direkt aufeinander ausgerichtet sind.

In dieser Semesterarbeit wurde nun ein System implementiert, welches eine Bluetooth-Kommunikation für Lego Mindstorms Roboter ermöglicht. Dies beseitigt die Nachteile der ursprünglichen Lösung und gibt mehr Spielraum für deren Anwendung.

Mit einem beliebigen Benutzerprogramm unter Linux können nun damit mehrere RCX gleichzeitig über eine relativ grosse Distanz kontrolliert, gesteuert und Programme darauf geladen werden. Zusammen mit einem Bluetoothmodul verfügt jeder RCX jetzt über einen Microcontroller, der zusätzliche Intelligenz einbringt, die für erweiterte Mindstorms-Anwendungen genutzt werden kann.

In den Kapiteln 2 bis 8 dieses Berichts wird das System vorgestellt. In Kapitel 9 soll eine einfache Anwendung zeigen, wie das Systems benutzt werden kann. Ideen für die Weiterentwicklung werden im Kapitel 10 skizziert.

Kapitel 2

Systemübersicht

2.1 Aufbau des ursprünglichen Systems

Das Downloaden von Programmen auf den RCX-Baustein geschieht mittels eines sogenannten IR-Towers, der über USB oder eine serielle Schnittstelle an den PC angeschlossen wird. Der RCX besitzt eine Infrarot Sende-/Empfangseinheit, die sehr direkt auf den IR-Tower ausgerichtet sein muss und auch nicht zu weit davon entfernt sein darf. Der Aufbau ist in Abbildung 2.1 ersichtlich.

Zum Download kann z.B. NQC eingesetzt werden, das es sowohl für Linux, als auch für Windows gibt. Mit diesem Tool können auch direkte Kommandos, z.B. zur Steuerung der Motoren oder zur Abfrage des Batteriezustandes, an den RCX gesendet werden.

Mit der Programmiersprache NQC ist es zudem möglich, dass der RCX eine 1-Byte-Nachricht an einen anderen RCX senden kann. Das Senden an den PC funktioniert allerdings nicht. Mehr zu dieser Einschränkung folgt im Kapitel 7.

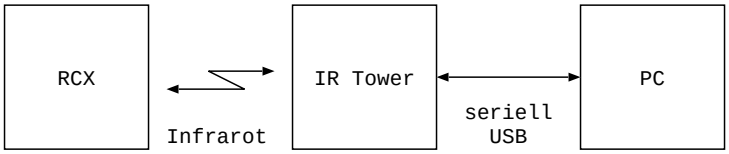

Abbildung 2.1: Aufbau des ursprünglichen Systems

2.2 Aufbau des mit Bluetooth erweiterten Systems

Der Aufbau unseres Systems mit Bluetooth-Anbindung ist in Abbildung 2.2

festgehalten. Anstatt des IR-Towers wird nun ein Bluetoothmodul an den PC angeschlossen. Dies kann ebenfalls über USB oder die serielle Schnittstelle geschehen. Wir haben allerdings nur die Anbindung mit der seriellen Schnittstelle verwendet. Vom Bluetoothmodul des PCs gelangen die Daten zum Bluetoothmodul des BTnodes. Der AVR-Microcontroller verarbeitet die empfangenen Daten so, dass sie über das am BTnode angeschlossene Infrarotmodul zum RCX geschickt werden können. Diese letzte Strecke entspricht derjenigen zwischen IR-Tower und RCX in der Originalkonfiguration. Die Antworten des RCX nehmen natürlich den umgekehrten Weg zurück.

Wenn der RCX mit der SendMessage()-Funktion eine 1-Byte-Nachricht abschickt, wird diese vom Microcontroller des BTnodes abgefangen und er entscheidet aufgrund des Werts des empfangenen Bytes, was nun geschehen soll. Die entsprechenden Aktionen können frei auf dem AVR programmiert werden, z.B. wäre es möglich, dass Daten gespeichert, ein Timer gestartet bzw. gestoppt wird oder natürlich, dass eine Nachricht an einen anderen RCX gesendet wird.

Als Gegenstück zur Software auf dem Microcontroller, die dort die Umsetzung von Bluetooth- auf Infrarotdaten bzw. Infrarot- auf Bluetoothdaten übernimmt, wird für diese Aufgabe auf dem PC ebenfalls eine Software benötigt. Um sich die freie Auswahl des Compiler-/Downloadtools zu erhalten, ist das entsprechende Programm eine eigenständige Anwendung. Wenn man also eine neuere Version von NQC oder etwa ein graphisches Steuerungsgstool benützen möchte, so muss die Software dafür nicht extra angepasst werden.

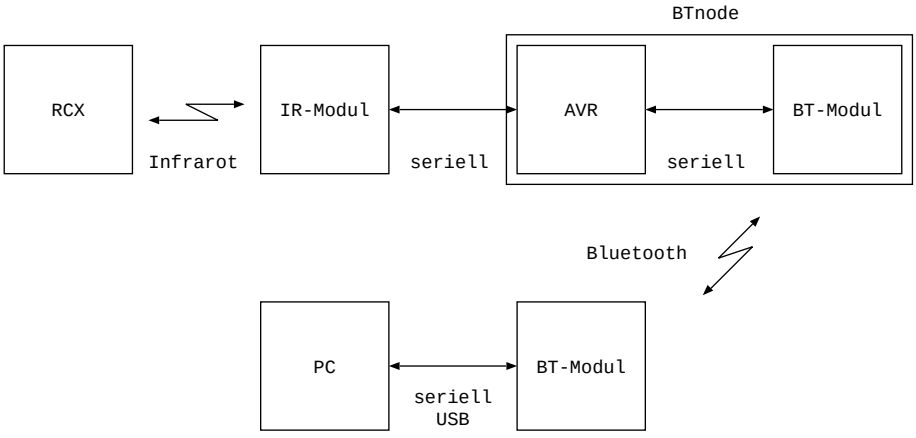

Abbildung 2.2: Aufbau des mit Bluetooth erweiterten Systems

2.3 Schichtenmodell der Protokolle

In Abbildung 2.3 wird der Aufbau der Protokollschichten gezeigt. Die L2CAP-Schicht ist Bestandteil des Bluetooth-Stacks. Das xhop-Protokoll dient zum

einfachen Transport der Lego-Daten über Bluetooth. Das Protokoll, das zur Infrarot-Kommunikation zwischen PC und RCX verwendet wird, wird hier als Lego-Protokoll bezeichnet. Genauere Beschreibungen der beiden letztgenannten Protokolle folgen in den Kapiteln 3 und 4.

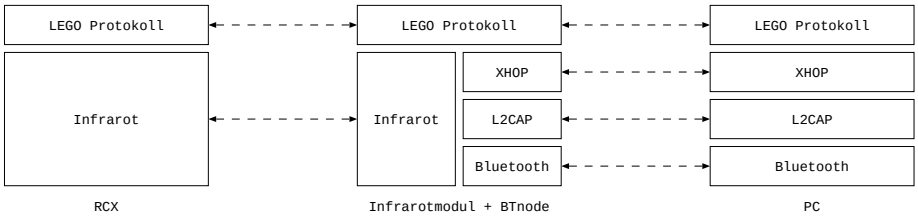

Abbildung 2.3: Protokollschichten

2.4 Verwendete Hard-/Software

2.4.1 PC

Als Betriebssystem diente Redhat Linux 7.2 mit ximian gnome. Die verwendete Kernelversion war 2.4.18. Beim Bluetooth-Stack Bluez [4] benutzten wir die Kernel-Version 2.0 und bei den Libs bzw. Utils je die Version 2.0-pre8.

2.4.2 BTnode

Der BTnode ist im wesentlichen ein Microcontroller mit angeschlossenem Bluetoothmodul. Wir benutzten die Revision 2 des BTnode mit dem ATmega 128L Microcontroller. Mehr Informationen dazu findet sich bei [5].

2.4.3 Bluetoothmodul

Das beim BTnode und beim PC verwendete Bluetoothmodul ist ein Ericsson ROK 101007.

2.4.4 Microcontroller

Wie schon erwähnt ist es ein Atmel AVR 8-Bit RISC Microcontroller ATmega 128L der u.a. über zwei Hardware-UARTs verfügt.

2.4.5 AVR Entwicklungstools

Die Software-Tools stammen von [6] :

- Compiler : gcc Version 3.0.4

- Avr-libc vom 3.3.2002
- Downloadtool : uisp vom 3.3.2002

Für die Entwicklung der AVR Software wurde ein STK500/STK501-Entwicklungsboard eingesetzt. Um die Fuses und die Spannung auf Board richtig zu setzen wurde AVR Studio 4.0 [7] verwendet. Mehr zu den Einstellungen findet man in den Abschnitten 6.5 und 6.6.

2.4.6 Infrarotmodul

Das Infrarotmodul ist ein Eigenbau. Genauere Informationen dazu sind im Kapitel 7 zu finden.

2.4.7 RCX

Beim RCX gibt es zwei Versionen. Wir verwendeten die Version 1.0, aber es sollte problemlos auch mit der Version 2.0 funktionieren.

Kapitel 3

XHOP-Protokoll

Das xhop-Protokoll ist ein Reduced Dynamic Source Routing (RDSR) Protokoll, das sich speziell für mobile ad hoc Netzwerke eignet, wie dies bei Bluetooth der Fall ist. Mehr zu diesem Thema findet man im Bericht zu [3].

In der Semesterarbeit "Grenzenlose Piconetze mit Bluetooth" [2] wurde xhop bereits teilweise für einen PC unter Linux und den BTnode implementiert. Durch Angabe einer Route von bekannten Bluetoothadressen können damit mehrere von drei verschiedenen Kommandos zu einem Netzwerkknoten gesendet und darauf ausgeführt werden. Unterstützt werden folgende Kommandotypen :

1. CMD_INQ
2. CMD_SEND_DATA
3. CMD_XHOP

Bei CMD_INQ wird auf dem entsprechenden Knoten ein Inquiry ausgeführt, d.h. es werden alle erreichbaren Knoten nach ihrer Adresse gefragt.

Mit CMD_SEND_DATA können Daten zu einem Knoten gesendet werden, dieses ist die für unserern Zweck wichtigste Funktion.

Der Typ CMD_XHOP dient schliesslich dazu nach Abarbeitung von einigen Kommandos das Paket an andere Knoten weiterzuschicken, die dann gegebenenfalls die Antworten auswerten.

Der Grund weshalb wir xhop verwenden ist vor allem der, dass es den Hauptzweck der Datenübermittlung (CMD_SEND_DATA) bereits erfüllt und es kaum Vorteile gebracht hätte, wenn wir diese Funktionalität nochmals neu implementiert hätten. Zudem ist es sehr gut denkbar, dass auch die anderen beiden Kommandotypen für bestimmte Anwendungen nützlich sind. Ausserdem können auch noch weitere Kommandos relativ einfach hinzugefügt werden.

In Abbildung 3.1 ist das Paketformat von xhop ersichtlich.

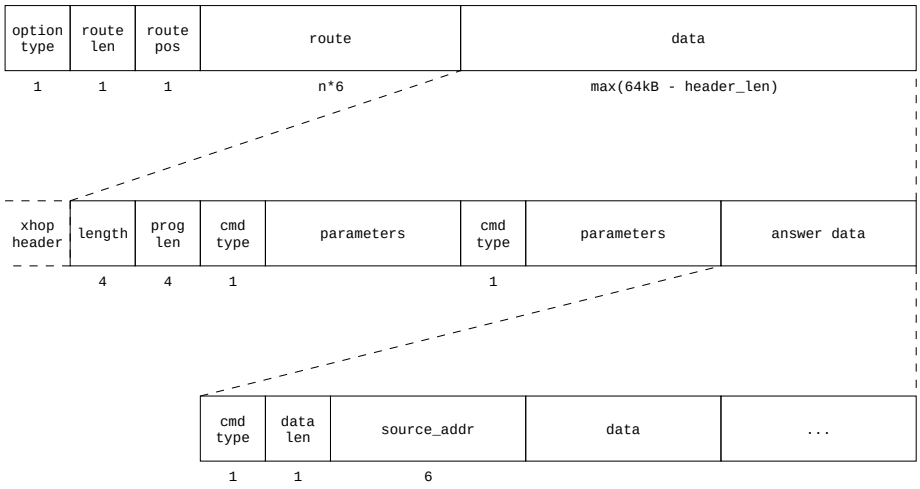

Abbildung 3.1: Paketformat von xhop

Kapitel 4

Lego-Protokoll

Der PC kommuniziert mit dem RCX über eine Infrarotschnittstelle mit Hilfe von Paketen. In diesem Abschnitt wird kurz darauf eingegangen, wie ein solches Paket aussieht und wie die Bestätigung der Daten gehandhabt wird.

4.1 Paket-Format

Der Aufbau der Pakete ist in Abbildung 4.1 dargestellt. Dabei wird das Paket

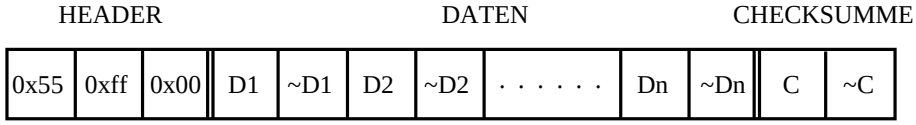

Abbildung 4.1: Lego-Paket

als Byte-Code interpretiert. Der Header besteht bei jedem Paket aus den selben drei Bytes. Auf den Header folgen die Daten (D1 - Dn, n variabel), wobei jedem Datenbyte das jeweilige bitweise komplementäre Byte angehängt wird (~D1 - ~Dn). Ein Beispiel: Wenn das Datenbyte in hexadezimaler schreibweise 0xa4 ist, dann lautet das nächste Byte im Paket 0x5b.

Das erste Byte im Datenblock (D1), der sogenannte Opcode, bestimmt, was für ein Kommando ausgeführt werden soll und was für eine Bedeutung die weiteren Daten haben. Zum Beispiel bedeutet der Opcode 0x51 (bzw. 0x59 siehe Abschnitt 4.2), dass der RCX einen Ton oder eine Tonfolge von sich geben soll. Das nächste Daten-Byte bestimmt dann, um welche Töne es sich handelt. Beispielsweise erzeugt das Byte 0x01 zwei Piepstöne. Die Beschreibung der einzelnen Opcodes findet man unter [11].

Abgeschlossen wird das Packet mit einer Checksumme, die nach folgendem Schema berechnet wird: C = D1 + D2 + ... Dn.

Auch der Checksumme folgt das komplementäre Byte.

Ein korrektes Lego-Paket, welches den RCX veranlasst zweimal zu piepsen, sieht demnach folgendermassen aus: [0x55 0xff 0x00 0x51 0xae 0x01 0xfe 0x52 0xad]

4.2 Bestätigung

Der IR-Tower bestätigt den Erhalt des vom PC gesendete Paketes in Form eines Echos. Anschliessend bestätigt der RCX die erfolgreiche Übertragung der Daten meistens mit einem Paket, das als Dateninhalt den komplementären Opcode des empfangen Paketes trägt (Abbildung 4.2). Zum Beispiel beantwortet der RCX

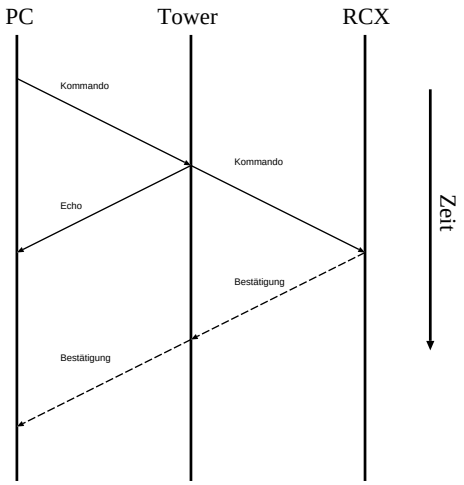

Abbildung 4.2: Timing-Diagramm

das Paket [0x55 0xff 0x00 0x51 0xae 0x03 0xfc 0x52 0xad] mit [0x55 0xff 0x00 0xae 0x51 0xae 0x51].

Antworten auf andere Kommandos (z.B. Get Battery Power) beinhalten noch zusätzliche Informationen. Wieder andere Opcodes (z.B. Message) werden gar nicht bestätigt. Weitere Informationen zu den Bestätigungen findet man ebenfalls unter [11].

Will man das selbe Kommando zweimal unmittelbar hintereinander ausführen, so kann nicht zweimal der selbe Opcode benutzt werden, da sonst der RCX nicht genau weiss, ob es sich um eine neue Anweisung handelt, oder ob der empfangene Bytestrom eine Wiederholung eines bereits ausgeführten Kommandos ist. Dieser Fall trifft zum Beispiel dann ein, wenn die Bestätigung nicht beim Sender angekommen ist. Der neue Opcode berechnet sich aus dem bitweisen XOR des alten Opcodes und des Bytes 0x08. Oder anders ausgedrückt: Soll mehrmals hintereinander die selbe Anweisung gesendet werden, so muss jedesmal im Opcode das vierte Bit, vom LSB aus gezählt, gekippt werden. Zur

Veranschaulichung ein Beispiel: Will man den RCX piepsen lassen, so sendet man das Paket [0x55 0xff 0x00 0x51 0xae 0x01 0xfe 0x52 0xad]. Will man ihn unmittelbar darauf nochmals piepsen lassen, so muss das Paket [0x55 0xff 0x00 0x59 0xa6 0x01 0xfe 0x5a 0xa5] übermittelt werden. Soll nochmals das selbe Kommando verwendet werden, so lautet der Opcode wieder 0x51 usw. Wird dazwischen irgend eine andere reguläre Lego-Anweisung gesendet, so kann danach wieder das Paket mit dem Opcode 0x51 oder mit 0x59 verwendet werden.

Auch hier bilden Kommandos, die nicht bestätigt werden (z.B. das Senden einer Message), eine Ausnahme. In diesen Fällen muss der Opcode nicht geändert werden, da die Bestätigung ja nicht verloren gehen kann.

4.3 Weitere Informationen

Da Lego selbst keine Informationen über das Protokoll herausgibt, muss man auf private Quellen aus dem Internet zurückgreifen. Sehr nützliche Informationen zu diesem Thema findet man z.B. bei [10] und [13].

Kapitel 5

PC- Software

In diesem Kapitel wird beschrieben, welche Aufgaben die Software auf dem PC erfüllen muss, mit welchen Konzepten die Probleme angegangen wurden und welcher bereits vorhandene Code wiederverwendet wurde.

5.1 Aufgabe

Grundsätzlich muss der PC in der Lage sein, Lego-Kommandos mit Bluetooth zu senden und ihre Bestätigung zu empfangen (Direkte Steuerung). Weiter sollten Funktionen, wie das Senden von Messages und das Downloaden von Programmen unterstützt werden.

5.2 Konzept

In Abbildung 5.1 sind die Kommunikationskanäle des Lego-Anwendungsprogramms zum IR-Tower bzw. zum BT-Modul dargestellt (Die Schicht Bluetooth in der Abbildung beinhaltet die für die Bluetooth-Kommunikation nötigen Protokolle, sowie das BT-Modul selbst). Der linke Kanal (Lego-Anwendungsprogramm - IR-Tower) stellt das ursprüngliche System dar. Die rechte Seite (Lego-Anwendungsprogramm - Bluetooth) wurde im Rahmen dieser Semesterarbeit für Linux implementiert. Die einzelnen Komponenten dieser Kommunikationskanäle werden in den Abschnitten 5.3, 5.4 und 5.6 beschrieben.

Mit dieser Abbildung soll auch angedeutet werden, dass bestehende, sowie künftige Lego-Programme, die für den konventionellen Infrarot-Kanal geschrieben wurden, unverändert für den Bluetooth-Kanal übernommen werden können. Um dies zu erreichen, muss das Programm BTbridge unabhängig von Benutzerprogrammen sein und die selbe Schnittstelle zum Anwenderprogramm haben, wie der IR-Tower (Serielle Schnittstelle). Wie diesen Anforderungen Rechnung getragen wurde, ist im Unterabschnitt 5.4.1 beschrieben.

Ein weiteres Ziel war es, dass man mehrere Kanäle unabhängig voneinander öffnen bzw. schliessen kann. Das heisst, es soll möglich sein, falls das die

Bluetooth-Hardware unterstützt, mehrere Kommunikationskanäle zu verschiedenen RCXs gleichzeitig offen zu haben und beliebig über diese zu kommunizieren. Diese Anforderung drängte sich auf, da der Aufbau von Bluetooth-Kanälen eine beachtliche Zeit in Anspruch nimmt (Grössenordnung von einigen wenigen bis zu 40 Sekunden). Das bedeutet, falls mehrere RCXs abwechselnd angesprochen werden müssen, müssten die BT-Kanäle sonst immer wieder geschlossen und geöffnet werden, woraus eine beachtliche Latenz vom Senden eines Kommandos, bis zu dessen Eintreffen beim RCX resultieren würde. Diese Funktionalität ist im Programm BTbridgeCtrl implementiert und in Abschnitt 5.5 beschrieben.

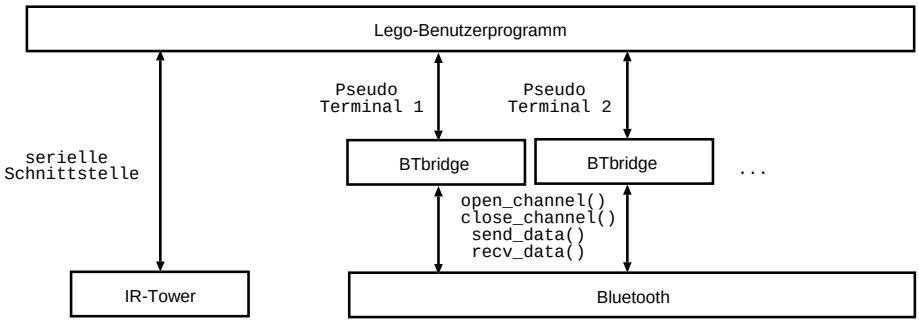

Abbildung 5.1: Aufteilung PC-Software

5.3 Bluetooth

Als Basis für die Bluetooth-Kommunikation auf der PC-Seite wurde die aus der Semesterarbeit [2] hervorgegangene Implementation des xhop-Protokolls für Linux verwendet. Um diese aber für unsere Zwecke benutzen zu können, mussten verschiedene Anpassungen am Code vorgenommen werden.

5.3.1 Anpassungen am Code (xhop.c)

Ein Problem stellte die Tatsache dar, dass keine Funktionen zum Senden bzw. Empfangen von Daten bereitgestellt wurden. Ausserdem war auch das Öffnen und Schliessen von Kommunikationskanälen fest im Code eingebettet.

Deshalb wurden die Funktionen send_data(), recv_data(), open_channel() und close_channel() eingefügt. Sie dienen als Schnittstelle zu den Programmen in der höheren Schicht:

1. void send_data(char* src_addr, char *data, int data_len, int socket_desc): Es werden Pointer auf die Source-BT-Adresse und die Daten übergeben. Ausserdem muss man den socket file descriptor des BT-Moduls angeben.

2. int recv_data(int socket_desc, char *data): Man übergibt einen Pointer, der auf die erhaltenen Daten zeigen soll, sowie wiederum den socket file descriptor. Die Länge der Daten wird zurückgegeben.

3. int open_channel(char* dest_bt_addr): Man übergib den Pointer auf die Ziel-Bluetoothadresse und erhält als Rückgabewert den socket file descriptor.

4. void close_channel(socket_desc): Schliesst den Kanal der zum übergebenen socket gehört.

Zusätzlich wurde die interne Funktion packetize() hinzugefügt, die das "Verpacken" von Daten in Pakete handhabt. Dabei ist zu beachten, dass diese Funktion nur eine Empfängeradresse und keine Route über mehrere Knoten als Argument annimmt und deshalb nur Daten direkt zum Empfänger gelangen können. Will man also für künftige Anwendungen eine Bluetooth-Adresse über mehrere Stationen erreichen, so muss man diese Funktion so anpassen, dass anstelle einer Adresse eine ganze Route übergeben werden kann. Der restliche Code unterstützt diese Möglichkeit bereits.

5.4 BTbridge

Aufgabe dieses Programms ist es, Daten, die von Anwenderprogrammen für den IR-Tower vorgesehen wären, abzufangen und mit Hilfe der in Abschnitt 5.3.1 erwähnten Funktionen über Bluetooth an den Empfänger (RCX) zu schicken. Anschliessend sollen die jeweiligen Bestätigungen den umgekehrten Weg gehen. Den Sourcecode zu diesem Programm findet man im File BTbridge.c.

5.4.1 Schnittstelle zum Benutzerprogramm

Das Benutzerprogramm schreibt seine Kommandos auf eine serielle Schnittstelle (Linux: Terminal), in der Annahme, der IR-Tower befindet sich am anderen Ende. Deshalb muss die Schnittstelle, vom Benutzerprogramm aus betrachtet, wie eine normale serielle Schnittstelle aussehen. Um dies zu erreichen, drängt sich der Interprozesskommunikations-Mechanismus Pseudo Terminal ([8]) auf, der vom Betriebssystem zur Verfügung gestellt wird.

Das Programm öffnet die Master-Seite des Pseudo Terminals und gibt den Namen des Slaves aus (z.B. /dev/pts/3). Die Slave-Seite kann man wie ein "normales" Terminal einer seriellen Schnittstelle (z.B. /dev/ttyS0) beschreiben und lesen. Das Programm BTbridge liest und beschreibt auf der anderen Seite den Master des Pseudo Terminals.

Möchte man also mit einem Lego-Programm eine bestimmte Verbindung zu einem RCX benutzen, kann einfach die Slave-Seite des Pseudo Terminals als serielle Schnittstelle angesehen werden.

Die Funktionsweise des Pseudo Terminals wird in Abbildung 5.2 illustriert.

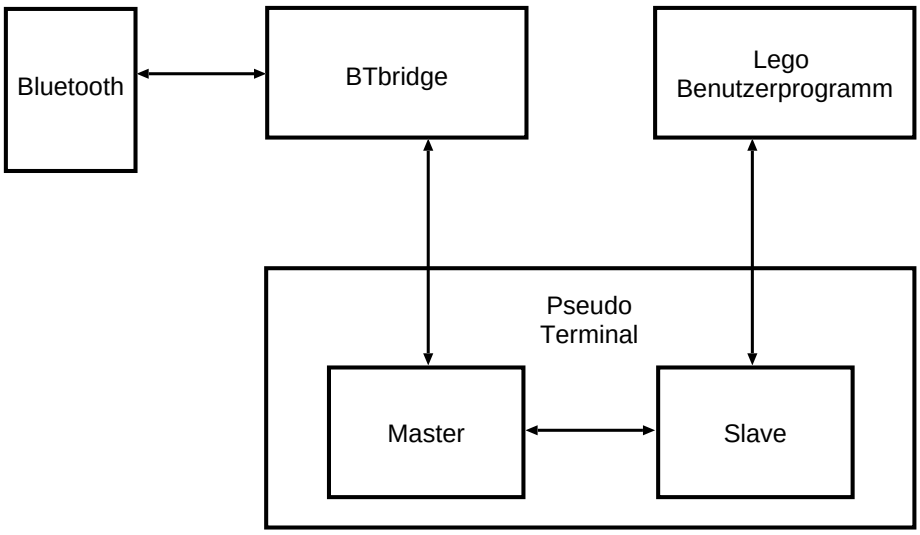

Abbildung 5.2: Funktionsweise Pseudo Terminal

5.4.1.1 Schnittstelle zu Bluetooth

Als Schnittstelle zur Bluetoothschicht dienen die in Abschnitt 5.3.1 erwähnten Funktionen, die das Öffnen und Schliessen von Kanälen, sowie Senden und Empfangen von Daten erlauben.

5.4.1.2 Programmablauf

Nach dem Öffnen der nötigen Schnittstellen sieht der Programmablauf folgendermassen aus:

1. lesen eines Kommandos vom Pseudo Terminal
2. senden des xhop-Packetes über Bluetooth
3. berechnen der Länge der Antwort
4. warten auf das xhop-Paket, welches die Antwort enthält
5. schreiben der Antwort auf das Pseudo Terminal

5.5 BTbridgeCtrl

BTbridgeCtrl kontrolliert das Öffnen und Schliessen der Kommunikationskanäle. Der dazugehörige Sourcecode befindet sich im File BTbridgeCtrl.c.

Über Menus und interaktive Eingabe kann der Benutzer wählen, ob ein neuer Kanal geöffnet oder ein bestehender geschlossen werden soll. Wie eine solcher Kanal und die dazugehörige Schnittstelle aussieht wurde in Abschnitt 5.4 erklärt.

Mögliche Ziele (Bluetoothadressen) werden in einer Liste geführt. Diese wird mit der Funktion get_addr_list() statisch erstellt. Die Adressen sind fest im Programmcode eigebunden. Wünscht man ein dynamisches Erstellen der Liste, so kann diese Funktion angepasst werden (z.B. mit Inquiry).

Offene Verbindungen werden auch in einer Liste verwaltet. Ein Listenelement besteht dabei aus der Bluetoothadresse, zu der eine Verbindung geöffnet wurde und der Prozessnummer des Prozesses (PID des BTbridge-Programms), welcher den Kanal bereitstellt.

Ein neuer Kommunikationskanal wird mit dem Aufrufen der Funktion create_interface() aufgebaut. Dabei wird ein neuer Prozess kreiert, der das Programm BTbridge in einem neuen xterm-Fenster startet.

Zur Illustration ist in Abbildung 5.3 eine mögliche Situation dargestellt. Die Pfeile sollen andeuten, dass die Tasks BTbridge Kind-Prozesse des Prozesses BTbridgeCtrl sind. Das Programm BTbridgeCtrl kennt sechs BT-Adressen welche jeweils zu einem RCX gehören sollten. Es sind vier Kommunikationskanäle geöffnet. Möchte man zum Beispiel den RCX mit dem BTnode mit der Adresse 00:80:37:14:43:86 piepsen lassen, so schreibt man einfach den entsprechenden RCX-Code (in diesem Fall: [0x55 0xff 0x00 0x51 0xae 0x01 0xfe 0x52 0xad] siehe KAPITEL) auf das Terminal /dev/pts/4 (Slave-Seite eines Pseudo Terminals).

Soll aber zum Beispiel die Verbindung zum RCX mit der zugehörigen BT-Adresse 11:11:11:50:11:11 geschlossen werden, so sendet das Programm BTbridgeCtrl ein KILL-Signal an den Kind-Prozess mit der Prozessnummer 383. Anschliessend wird der entsprechende Eintrag in der Liste "Offene Kanäle" gelöscht.

5.6 Lego Benutzerprogramme

Als Benutzerprogramme können beliebige Programme verwendet werden, die das Lego-Protokoll unterstützen und bei denen die serielle Schnittstelle angegeben werden kann.

Wir haben für unsere Tests vor allem NQC [1] verwendet. Leider kann das Timing, welches in NQC integriert ist, nicht eingehalten werden. Das heisst, die Zeit vom Senden des Kommandos, bis zur spätest möglichen Ankunft der Antwort ist zu knapp bemessen. Deshalb musste der Code geringfügig angepasst werden. Konkret heisst das, dass im Sourcecode des NQC-2.4 folgende Zeilen abgeändert wurden:

1. File /rcxlib/RCX_PipeTransport.cpp, Zeile 95-98: fRxTimeout wurde auf 100000 (ms) gesetzt. Die Wahl des Wertes war mehr oder weniger willkürlich. Er muss einfach genügend gross sein.

2. File /rcxlib/RCX_Link.cpp, Zeile 404-406: Die Zeilen, welche die erwartete Antwortlänge (= 0) für die Kommandos 0xf7 und 0xd2 zurückgeben, müssen wieder eingefügt werden. (Sie sind in der Version 2.4 von NQC auskommentiert)

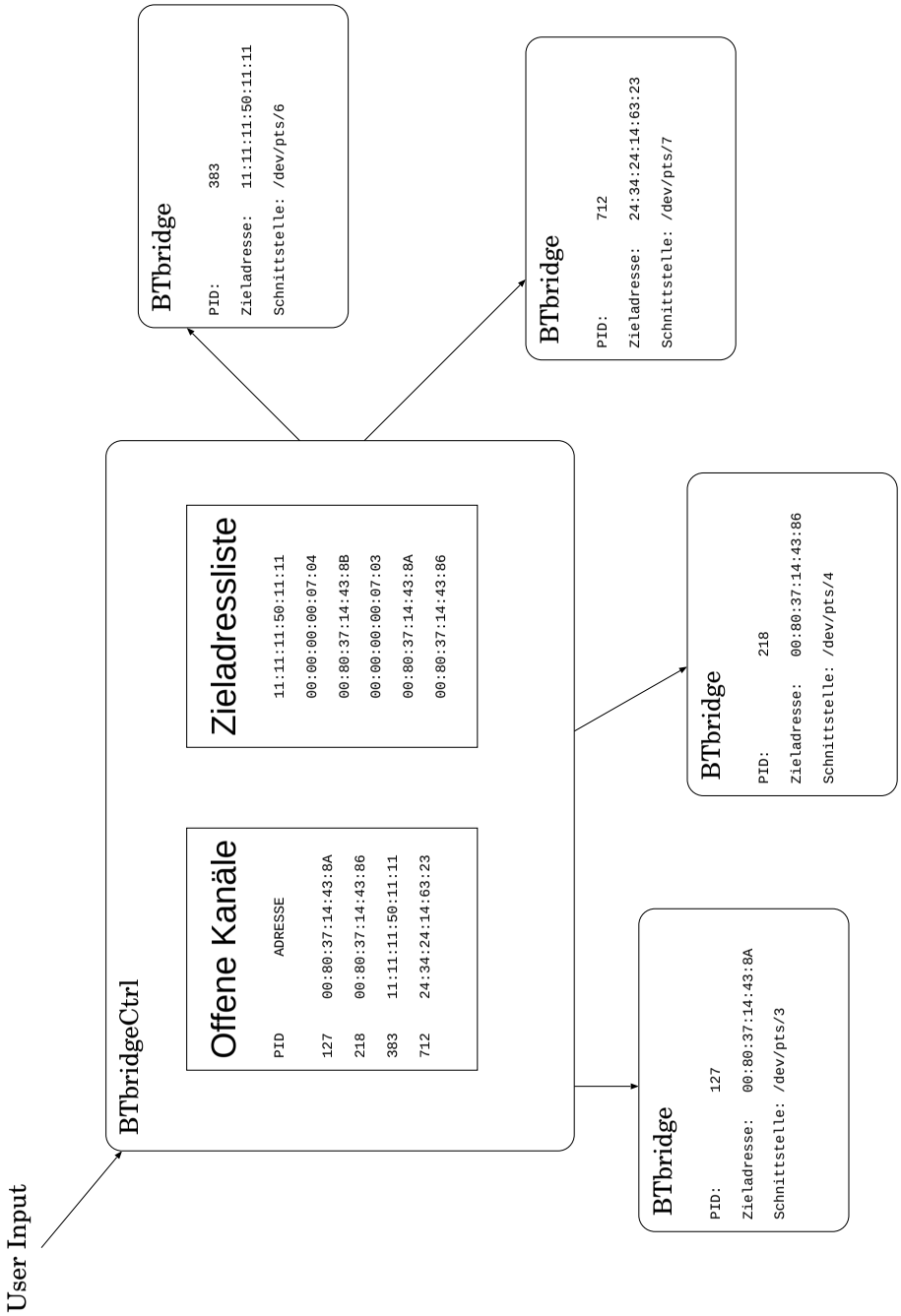

Abbildung 5.3: Funktionsweise BTbridgeCtrl

Bei anderen Programmen kann dieses Time-Out, wenn überhaupt vorhanden, anders aussehen. Deshalb sollte man diesem Punkt besondere Aufmerksamkeit schenken.

Kapitel 6

AVR-Software : RCXctrl

Die Software für den AVR Microcontroller basiert auf dem μxhop-Code der Semesterarbeit "Grenzenlose Piconetze mit Bluetooth" [2], in der die l2cap-Schicht des Bluetooth-Protokoll-Stacks und ein Teil des xhop-Protokolls implementiert wurde.

Weil bei dieser Arbeit eine ältere Version des BTnodes mit einem Atmega103L verwendet wurde, mussten einige Anpassungen vorgenommen werden. Der wichtigste Unterschied dabei ist, dass bei der neuen Version des BTnodes mit dem ATmega128L die Kommunikation mit dem Bluetoothmodul über einen der zwei Hardware-UART läuft. Beim Vorgänger erfolgte sie über einen Software-UART, da dieser AVR nur über eine UART-Schnittstelle verfügt.

6.1 Erweiterung der xhop-Implementation

In der ursprünglichen Version konnten xhop-Pakete vom AVR lediglich weitergeleitet werden. Es mussten also die Paketverarbeitungsfunktionen analog zur xhop-Software auf dem PC hinzugefügt werden. Da es für den Zweck der RCX-Anbindung genügt, wenn man mit xhop Daten übertragen kann, wurde nur die Verarbeitung von Kommandos des Typs CMD_SEND_DATA implementiert. Die entsprechenden Funktionen für andere Kommandos können aber bei Bedarf mit kleinem Aufwand eingebaut werden.

6.2 Weitere Anpassungen

Beim xhop-Protokoll wird davon ausgegangen, dass jeweils eine Bluetooth-Verbindung geöffnet, ein Paket empfangen, die Verbindung geschlossen und dann das Paket analysiert wird. Um einen RCX aber effizient steuern zu können, muss das empfangene Paket analysiert und verarbeitet werden, solange die Verbindung noch offen ist. Sonst müsste vom AVR für die Bestätigung wieder eine neue Verbindung zurück zum PC geöffnet werden, was manchmal bis zu 30 - 40 Sekunden

dauern kann. Deshalb musste der Aufruf der xhop-Paketanalysefunktion verlegt werden. Während er zuvor im Handler process_evt()[1] nach Empfang eines EVT_DISCONN_COMPLETE-Pakets erfolgte, geschieht er nun in der Funktion process_acl()[2].

Ein Problem von μxhop war die wrapper()-Funktion, die in einer Endlos-Schleife die Ausgaben des Bluetoothmoduls abfragte. Je nach dem, ob der AVR auf Verbindungen von anderen wartete oder ob er selbst eine Verbindung öffnete, musste die Funktion mit anderen Handler-Funktionen aufgerufen werden. Da dies rekursiv geschah, wurde der Stack mit jedem Zustandswechsel etwas mehr gefüllt. Zur Lösung des Problems wurden Flag-Variablen (bInitiatorAVR, bProcessInqResults) eingeführt, mit denen nun jeweils der entsprechende Handler ausgewählt wird. Die Endlosschleife wurde in die main()-Funktion integriert.

6.3 Kommunikation mit dem RCX

Die Kommunikation mit dem RCX verwendet den zweiten UART des ATmega128L.

6.3.1 RCX sendet Daten

In der Endlosschleife im main() werden nicht blockierend, analog zum Bluetoothmodul, vom Infrarotempfänger empfangene Daten gelesen und eine Handler-Funktion aufgerufen. In dieser Funktion (rcx_recv_data()) wird überprüft, ob es sich um eine korrekte LEGO-Message handelt, d.h. von der Form 0x55 0xff 0x00 MSG ~MSG CHKSUM ~CHKSUM (siehe Kapitel 4) ist. Wenn dies der Fall ist, wird in einer switch-Anweisung entschieden, was nun geschehen soll (z.B. eine Message an einen anderen RCX senden). Wenn dies nicht der Fall ist, werden die Daten ignoriert und es wird auf die nächste korrekte Message gewartet.

6.3.2 PC oder anderer RCX sendet Daten

Wird über Bluetooth vom PC oder einem anderen RCX ein xhop-Paket empfangen, wird zuerst entschieden, ob das Paket nur weitergeleitet, oder ob es selbst verarbeitet werden soll. Ist letzteres der Fall wird in der Funktion exec_cmd() der Typ des Paketes bestimmt. Wenn es sich um den Typ CMD_SEND_DATA handelt, werden die Daten in der Funktion exec_send_data() über den Infrarotsender an den RCX weitergeschickt. Falls für dieses RCX-Kommando eine Antwort erwartet wird, wird diese hier abgewartet. (Weil ein RCX nur mit SendMessage() Daten senden kann und eine solche Message keine Bestätigung verlangt, kann angenommen werden, dass wenn man eine Antwort vom RCX erwartet, diese für einen PC bestimmt ist. Eine Bluetoothverbindung von einem

[1] EVT: Event. EVT-Pakete dienen vor allem zum Verbindungsauf- und abbau

[2] ACL : Asynchronous Connection-Less. In ACL-Paketen werden die eigentlichen Daten transportiert

PC aus sollte solange aufrecht erhalten werden, bis er die Antwort bekommen hat.) Ausser der Antwort des RCX wird in das zurückgesendete Paket das sonst vom IR-Tower generierte Echo des Kommandos kopiert (siehe Kapitel 7).

Da es möglich ist, dass der RCX aus einem Grund nicht oder nicht vollständig antwortet (z.B. Störung bei der Infrarotübertragung, der RCX hat sich ausgeschaltet, ...) wird nach etwa zehn Sekunden das Warten abgebrochen und die Funktion beendet (blockierendes Lesen mit Time-Out), die Bluetoothverbindung bleibt aber offen. Andernfalls wird, wenn die erwartete Antwort eintrifft, diese über die Bluetoothverbindung zum PC zurückgeschickt.

6.3.3 Konfliktlösung bei geöffneter Bluetooth-Verbindung

Wenn eine Bluetoothverbindung offen ist und der AVR mittels SendMessage() vom RCX eine Nachricht erhält, die ihn veranlasst an eine vorbestimmte Adresse etwas zu schicken, wird dies solange zurückbehalten, bis die Bluetooth-Verbindung geschlossen wurde. Sollten aber in der Zwischenzeit andere Daten empfangen werden, so wird das zurückbehaltene Paket verworfen. Das gleiche Verhalten gilt, wenn der AVR über die offene Bluetoothverbindung ein xhop-Paket zur Weiterleitung empfängt.

Eine Erweiterung der Software mit einem Scheduler, würde hier sicher eine bessere Verabeitung ermöglichen. Die Jobs könnten dann in einer Liste abgelegt und bei Möglichkeit abgearbeitet werden.

In Abbildung 6.1 wird der hier beschriebene Ablauf der Kommunikation veranschaulicht.

6.3.4 Funktionsaufrufe

Die Abbildungen 6.2 und 6.3 zeigen eine grobe Übersicht der Funktionsaufrufe, wenn eine Bluetooth-Verbindung von jemand anders geöffnet wird (6.2), bzw. wenn der AVR selbst eine Verbindung öffnet (6.3). Daraus wird auch deutlich, dass im letzteren Fall der AVR ankommende Daten vom RCX nicht verarbeitet. Falls noch genügend Platz ist, werden diese Daten im Puffer für UART1 zwischengespeichert und dann analysiert, wenn der Microcontroller in den passiven Modus zurückgekehrt ist.

6.4 Debugging

Das Debugging auf dem AVR erfolgt prinzipiell gleich wie im ursprünglichen μxhop (siehe dazu Abschnitt 5.2.1 in [2]). Der Microcontroller wird mit einem 'geraden' seriellen Kabel mit dem PC verbunden. Mit dem Befehl *minicom -o* können dann die Debug-Meldungen gelesen werden.

Die Debug-Ausgaben erfolgen jedoch im Gegensatz zum Original-μxhop über den Software-UART, da ja beide UART des ATmega128L bereits besetzt sind. Weil der Software-UART nur mit 9600 bps senden kann, muss Minicom nun auf 9600 8N1 gesetzt werden. Ansonsten bleibt alles gleich, wie in [2].

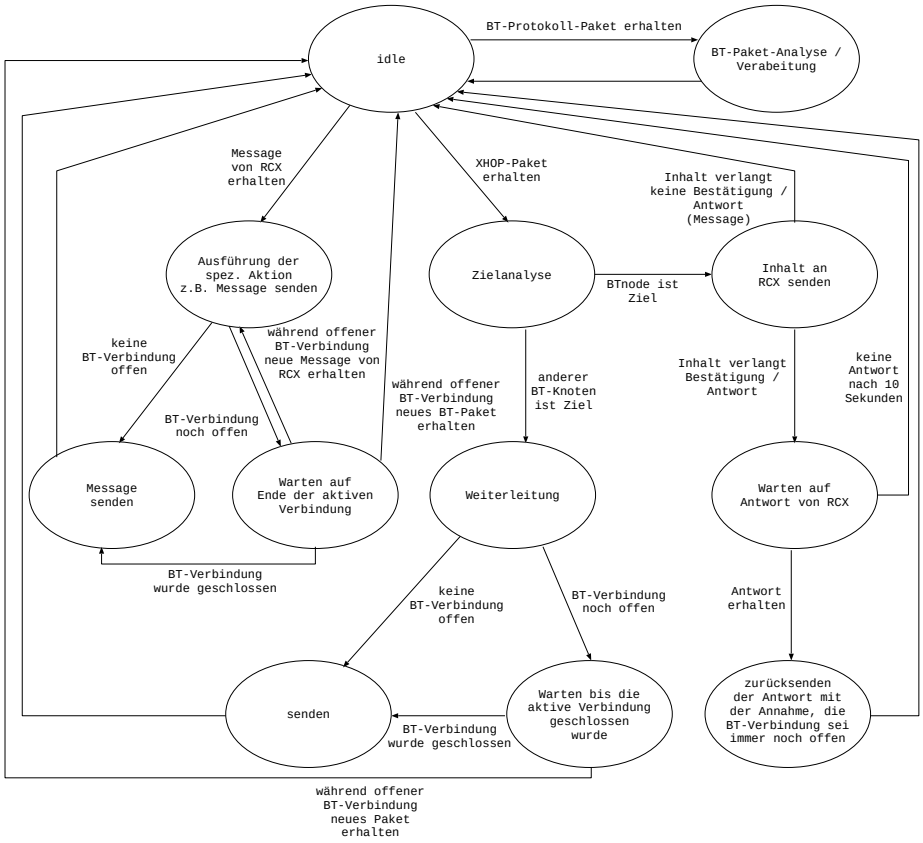

Abbildung 6.1: Zustandsdiagramm von RCXctrl

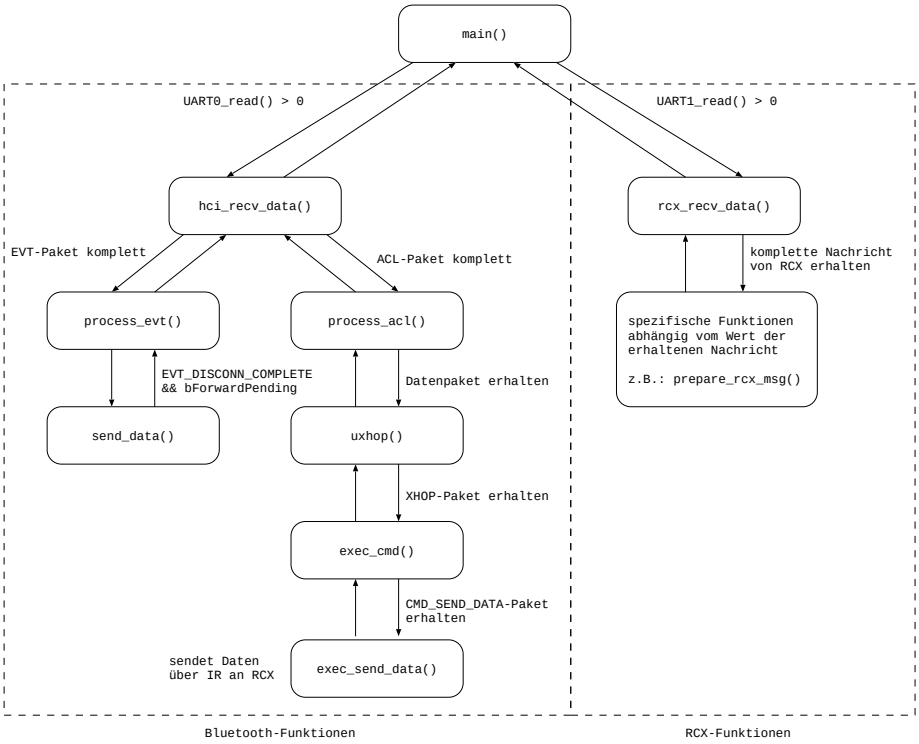

Abbildung 6.2: Funktionsaufrufe im passiven Modus

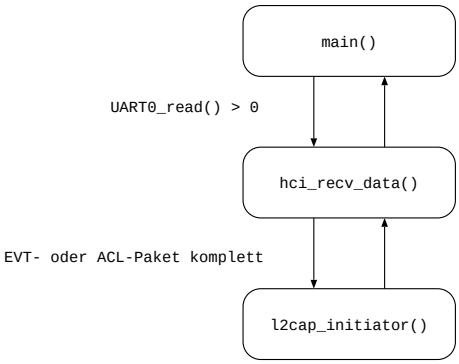

Abbildung 6.3: Funktionsaufrufe im aktiven Modus

Um die Übersichtlichkeit des Codes etwas zu erhöhen, wurden die Ausgabe-Funktionen und Defines zur Steurung der Ausgabe in separate Files ausgelagert (Debug.c, Debug.h).

6.5 Einstellungen der Fuses

Die folgenden Fuses des ATmega 128L sollten mit AVR Studio gesetzt werden :

- Preserve EEPROM memory through the Chip Erase cycle; [EESAVE=0]
- Boot Flash section size=4096 words Boot start address=$F000; [BOOTSZ=00]; default value
- Brown-out detection level at VCC=2.7V; [BODLEVEL=1]
- Brown-out detection enabled; [BODEN=0]
- CKOPT fuse (operation dependent of CKSEL fuses); [CKOPT=0]
- ext. Crystal / Resonator Medium Freq.; Start-up time: 258CK + 4ms; [CKSEL=1100 SUT=00]

Insbesondere sollten die beiden letzten Einstellungen zum externen Clocksignal genau beachtet werden, da der Microcontroller mit einer Einstellung geliefert wird, die ein internes Clocksignal benützt. Dieser Clock läuft mit einer Frequenz von 1 Mhz, anstatt der auf dem BTnode verwendeten 3.6864 MHz. Durch eine solche falsche Taktung werden von den UART's falsche Baudraten verwendet, was dazu führt, dass weder Bluetoothmodul, Infrarotmodul noch die Debugausgabe funktionieren.

Zudem kann es bei einer falschen Konfiguration der Clock-Fuses passieren, dass vom AVR gar kein Clock mehr verwendet wird und er so dann auch nicht mehr programmierbar ist.

Ausserdem sollte noch speziell darauf geachtet werde, dass die (defaultmässig aktivierte) ATmega 103 compatibility Fuse nicht gesetzt ist, da in dieser Konfiguration nicht alle Funktionen des ATmega 128 genutzt werden können (u.a. der zweite UART, über den die Kommunikation mit dem Infrarotmodul läuft).

6.6 Einstellung der Board-Spannung

Wenn das Entwicklungsboard benutzt wird, sollte die Board-Spannung mit dem AVR Studio auf 3.3V gesetzt werden, um mit dem BTnode kompatibel zu sein. Obwohl es eigentlich für die Software keinen Unterschied ausmachen sollte, traten bei der Verwendung von 3.3V anstatt 5V einige Probleme, mehr dazu in Kapitel 8.

Kapitel 7

Infrarot - Hardware

7.1 Motivation für eigene IR-Hardware

Im ursprünglichen System wird der Infrarot Tower für die Kommunikation zwischen Computer und RCX verwendet. Dieser Tower hat folgende experimentell festgestellte Eigenschaften:

1. Der Tower schickt ein vom PC erhaltenes Paket unverändert an den Computer zurück (Bestätigung des Empfangs durch ein Echo).
2. Er kann nur vom RCX etwas empfangen, wenn er kurz zuvor selbst etwas an den RCX gesendet hat. Das heisst, der Tower ist nur aktiv, wenn etwas an den RCX gesendet wird und noch ein paar Sekunden danach, um die Antwort zu empfangen. Sonst ist der IR-Tower ausgeschaltet (Stromsparschaltung) und kann nichts vom RCX empfangen.
3. Fehleranfälligkeit, vor allem beim Header eines Lego Paketes (ändert 0x55 0xff 0x00 z.B. in 0x55 0xfe 0x00)
4. Weiter fällt negativ auf, dass der Tower, wenn er ein Lego-Paket vom RCX an den PC weiterleitet, unmotiviert eine unbestimmte Anzahl '1'er am Schluss der Echo-Übertragung anhängt. Der vom PC empfangene Datenstrom sieht beispielsweise folgendermassen aus (Eine Bestätigung auf ein Piepston-Kommando): [0x55 0xff 0x00 0xae 0x51 0xae 0x51 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff]

Diese Eigenschaften sind für unser System wenig akzeptabel. Unter anderem wird durch dieses Verhalten das Senden von Messages von einem RCX zu einem anderen zu einem beliebigen Zeitpunkt verunmöglicht. Um diese Funktionalität zu gewährleisten muss der Infrarotempfänger immer aktiv sein (vgl. mit Punkt 2 der obigen Auflistung). Deshalb wird ein eigener, auf dem in [9] beschriebenen Schema basierender, Transceiver eingesetzt.

Diese IR-Hardwarelösung ist zudem weniger fehleranfällig und hängt auch nicht, wie in Punkt 4 beschrieben, unbrauchbare Daten an ein Paket an. Die

Echo-Funktion (Punkt 1) übernimmt in unserem System die Software auf dem AVR.

Ausserdem sei noch erwähnt, dass die neue Schaltung, eine geringere Komplexität aufweist, als der IR-Tower.

7.2 Schema und Materialliste

Unser Prototyp des IR-Moduls ist für 5V Spannung ausgelegt. Auf dem BTnode wird jedoch ine Spannung von 3.3V verwendet. Da das benötigte Material für eine 3.3V-Version bis zum Ende der Semesterarbeit nicht eingetroffen ist, konnte kein entsprechendes Modul gebaut werden. Prinzipiell müsste aber nur der Vorwiderstand der IR-LED angepasst werden. Das Schema des 5V-IR-Transceivers wird aus 7.1 ersichtlich. Das benötigte Material ist in Tabelle 7.1 aufgelistet.

Stückzahl	Bauteilbeschreibung
1	Inverter DM 74L504
1	LMC 555
1	Widerstand 47 kΩ
1	Widerstand 4.7 kΩ
1	Widerstand 150 kΩ
1	Widerstand 100Ω
1	Kondensator 100 pF
1	Kondensator 4.7 μF
1	TSOP 1176
1	IR LED

Tabelle 7.1: Materialliste (5V-Version)

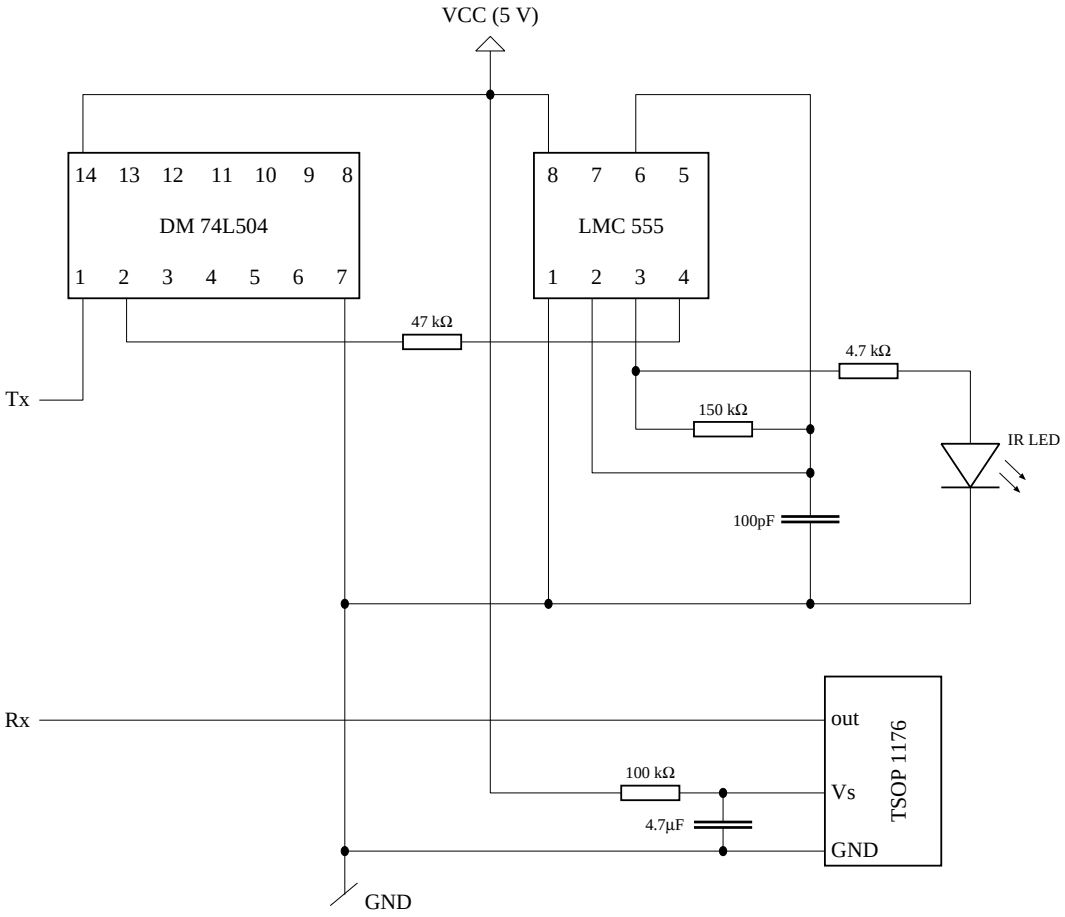

Abbildung 7.1: Schema des IR-Transceivers (5V-Version)

Kapitel 8

Probleme

8.1 AVR Probleme

- Wenn auf dem Entwicklungsboard 3.3V anstatt 5V verwendet wird, kann es sein, dass der Reset mit dem Resetknopf nicht mehr funktioniert. Man muss dann die Stromversorgung des Boards ausschalten.

- Wenn das Board zusammen mit dem Bluetoothmodul mit externer Spannungsversorgung verwendet wird, funktioniert der Reset manchmal nicht, ohne dass auch diese Spannungsversorgung unterbrochen wurde. Zur Programmierung des Microcontrollers hingegen, muss die Versorgung immer eingeschaltet sein.

- Die Probleme mit falschgesetzten Clock-Fuses sind in Abschnitt 6.5 beschrieben.

- Ein seltsames Problem tritt auf, wenn z.B. die in der Funktion exec_send_data() markierte for-Schleife durch die eigentlich gleichwertige memcpy-Anweisung ersetzt wird. Jedesmal, wenn versucht wird eine Bluetoothverbindung zum AVR zu erstellen, erfolgt ein Reset des Microcontrollers. Das MCUSR-Register, in dem die Ursache des letzten Resets vermerkt sein sollte, gibt aber keine Auskunft darüber, wieso der Reset erfolgte. Das gleiche Verhalten kann erreicht werden, indem die Variable replylen in der Funktion exec_send_data() auf 0 gesetzt wird, oder wenn das __INITIATOR__- oder __RECV__ Define zur Debug-Ausgabe eingeschaltet wird. Besonders erstaunlich bei diesem Reset ist auch die Tatsache, dass beim reinen Versuch eine Bluetooth-Verbindung zu öffnen, die Funktion exec_send_data() gar nie aufgerufen wird.

8.2 Probleme bei Anwendungen

- Die Antworten bzw. Bestätigungen vom RCX treffen nicht rechtzeitig beim PC ein (siehe Abschnitt 5.6)
- Der Download von Firmware mit NQC ist nicht möglich. Wahrscheinlich ist die Behandlung eines dabei benötigten, speziellen Opcodes in NQC nicht sauber gelöst. Wenn nämlich das NQC-Programm mit dem angepassten Time-Out zusammen mit dem IR-Tower, wie im Original-System, verwendet wird, tritt das selbe Problem auf.
- Wenn vom RCX unmittelbar nacheinander mehrere Messages gesendet werden, gehen zwischen der ersten und letzten Message einige verloren. Dies wahrscheinlich deshalb, weil entweder der Puffer des RCX oder des AVR UART's zu klein ist. Um dieses Problem zu vermeiden, sollte zwischen zwei SendMessage()-Anweisungen ein Wait()-Befehl stehen, z.B. Wait(200), was 2 Sekunden Wartezeit entspricht.
- Nachdem der BTnode selbst eine Bluetooth-Verbindung geöffnet und dann auch wieder geschlossen hat, funktioniert der Rückkanal einer vom PC initierten Verbindung nicht mehr. Das bedeutet, dass wenn der RCX eine Message an RCXctrl sendet, die eine Nachricht an einen anderen Bluetooth-Knoten zur Folge hat, wird diese Nachricht zwar korrekt abgeschickt und danach können auch noch weitere solche Nachrichten verschickt werden, jedoch ist der Download von Programmen vom PC aus mit NQC nicht mehr möglich, da NQC keine Bestätigungen vom RCX erhält. Das gleiche passiert, wenn über den BTnode ein xhop-Paket weitergeleitet wird (forward). Als Notlösung kann der RCX aber noch gesteuert werden, indem man, z.B. mit dem send Programm [12], direkte Kommandos schickt, ohne auf eine Bestätigung zu warten.

Kapitel 9

Anwendungsbeispiele

9.1 Beispiel mit zwei RCX[1]

9.1.1 Aufgabe

Ein Roboter soll Linien auf dem Boden zählen, bis er auf ein Hindernis stösst. Die ermittelte Zahl soll an einen anderen RCX gesendet werden, worauf dieser die enstprechende Anzahl Piepstöne von sich gibt. Den Ablauf zeigen die Abbildungen 9.1, 9.2 und 9.3. Die dazugehörigen NQC-Programme sind im Abschnitt 9.1.3 aufgeführt.

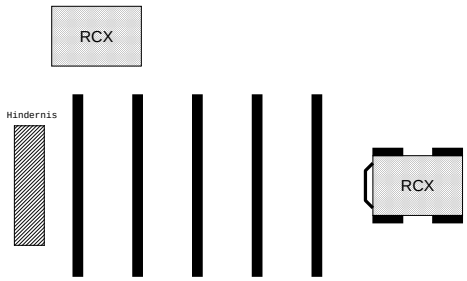

Abbildung 9.1: Situation 1

9.1.2 Vorgehen

1. Einschalten des RCX und BTnodes mit IR-Modul. Etwa eine Minute Warten bis der BTnode empfangsbereit ist.

2. BTmodul über die serielle Schnittstelle an den PC anschliessen.

[1] Aufgrund der fehlenden 3.3V-Infrarot-Hardware konnte dieses Beispiel leider nicht getestet werden.

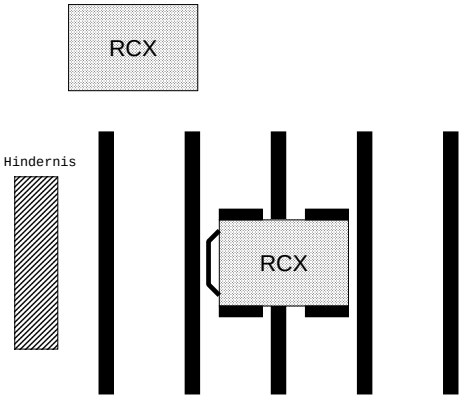

Abbildung 9.2: Situation 2

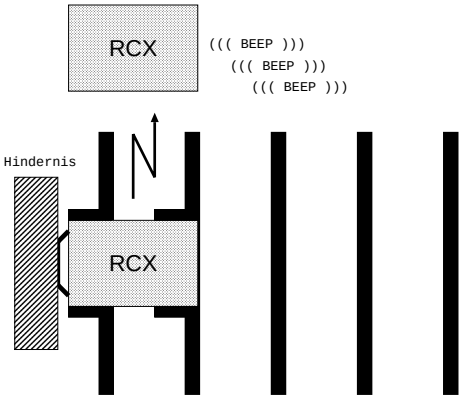

Abbildung 9.3: Situation 3

3. In xterm : *hciattach /dev/ttyS0 ericsson 57600 flow*
4. *hciconfig hci0 up*
5. BTbridgeCtrl starten
6. Menupunkt 1 wählen (open a new interface)
7. Adresse des ersten RCX (Roboter) auswählen z.B. 11:11:11:50:11:11
8. Ein neues Fenster öffnet sich, nach kurzer Zeit erscheint der Name des Slave Pseudo Terminals, z.B. /dev/pts/2
9. Neues xterm öffnen und dort mit *nqc -S/dev/pts/2 -d demoRCX1.nqc* das Program auf den RCX laden.
10. Im Menu von BTbridgeCtrl die Adresse des zweiten RCX wählen und nach dem gleichen Schema das Programm *demoRCX2.nqc* laden. Dieser RCX erhält ein anderes Terminal zugeordnet, z.B. /dev/pts/3
11. Zuerst Programm des zweiten RCX starten : *nqc -S/dev/pts/3 -run*
12. Nun das Programm des RCX-Roboters starten : *nqc -S/dev/pts/2 -run*
13. Nach dem Starten der Programme können die Bluetoothverbindungen mit BTbridgeCtrl wieder geschlossen werden.

9.1.3 Beispiel-Programme

9.1.3.1 demoRCX1.nqc

```
int state, old_state, counter;

#define ON_LINE 1
#define OFF_LINE 2


//thresholds for light sensors
#define DARK 35
#define LIGHT 40

//speed (maximal 7)
#define POWER 1



task main()
{
    SetSensorType (SENSOR_1, SENSOR_TYPE_LIGHT);
    SetSensorType (SENSOR_2, SENSOR_TYPE_LIGHT);
```

```
    SetSensorType (SENSOR_3, SENSOR_TOUCH);
    SetPower (OUT_A+OUT_C,POWER);
    OnFwd(OUT_A+OUT_C);
        state = OFF_LINE;
        old_state = OFF_LINE;
        counter = 0;

        while (SENSOR_3 == 0)
    {
                if (SENSOR_1 < DARK && SENSOR_2 < DARK) {
                        state = ON_LINE;
                }
                else
                {
             state = OFF_LINE;
                        if(old_state == ON_LINE) {
                               counter++;
                       }
                }
                old_state = state;

        }

        Off(OUT_A+OUT_C);

        SendMessage(counter);
}
```

9.1.3.2 demoRCX2.nqc

```
int counter;

task main ()
{
  ClearMessage(); // clear out the received message
  until(Message() > 0 ); // wait for next message

  counter = Message();


  while(counter) {
        PlaySound(SOUND_DOUBLE_BEEP);
        Wait(200);
        counter--;
  }
```

```
}
```

9.2 Direkte Steuerung eines Roboters

Die direkte Steuerung der Motorausgänge ist etwas mühsam, da man mit NQC die entsprechenden Opcodes im Hex-Format an den RCX senden muss. In diesem kleinen Beispiel wird angenommen, dass der RCX zur Fortbewegung zwei Motoren besitzt, die an die Ausgänge A und C angeschlossen sind, und dass mit BTbridge eine Verbindung geöffnet wurde mit dem Pseudo Terminal /dev/pts/2. Die genaue Bedeutung der Opcodes kann in [11] gefunden werden.

1. Power für beide Motoren auf den Wert 3 setzen : *nqc -S/dev/pts/2 -raw 13050203*
2. Beide Motoren sollen in Vorwärtsrichtung drehen : *nqc -S/dev/pts/2 -raw e185*
3. Beide Motoren starten, der Roboter sollte nun vorwärts fahren : *nqc -S/dev/pts/2 -raw 2185*
4. Mit diesem Befehl werden beide Motoren gestoppt : *nqc -S/dev/pts/2 -raw 2145*

Kapitel 10

Ausblick

10.1 Erweiterte Anwendungen von RCXctrl

Wie schon erwähnt, kann der RCX mit SendMessage() eine Zahl zwischen 0 und 255 senden[1]. Dieser Zahl kann nun eine bestimmte Aktion auf dem AVR zugeordnet werden. Die zugehörige Funktion muss dazu einfach in der switch-Anweisung der Funktion rcx_recv_data() aufgerufen werden. Beispielsweise wäre es möglich, dass der AVR zusätzlichen Speicher für den RCX bereitstellt.

10.2 Steuerung mit PC

Die Fernsteuerung eines RCX-Roboters vom PC aus ist sehr umständlich (siehe Abschnitt 9.2). Deshalb wäre es wünschenswert, ein graphisches Tool dafür zu haben.

Weiter ist denkbar für einfache Bedienung und effiziente Benutzung des Bluetoothsystems, Programme zu schreiben, die die gesamte Funktionalität von BTbridge, BTbridgeCtrl und des Lego-Benutzerprogramms vereint. Dies natürlich zu Lasten der Kompatibilität mit dem ursprünglichen Infrarot-System.

10.3 IR-Modul

Wie im Abschnitt 7.2 erwähnt, müsste davon noch eine 3.3V-Version erstellt und getestet werden.

[1] In der ursprünglichen Anwendung der Mindstorms wird von der Verwendung der Zahl 0 abgeraten, da ein RCX per Konvention auf eine unbekannte Nachricht eines anderen RCX in einer until(Message() > 0)-Anweisung wartet. Man könnte nun aber die 0 als spezielles AVR-Steuerzeichen verwenden.

Literaturverzeichnis

[1] Dave Baum. NQC- Not Quite C. http://www.enteract.com/~dbaum/nqc/

[2] Egon Burgener und Peter Fercher. Grenzenlose Piconetze mit Bluetooth. Semesterarbeit, ETH Zürich, WS 2001/02

[3] Lars Wernli und Riccardo Semadeni. Bluetooth unleashed. Semesterarbeit, ETH Zürich, SS 2001

[4] Official Linux Bluetooth protocol stack. http://bluez.sourceforge.net/

[5] BTnode Project Page. http://www.tik.ee.ethz.ch/~beutel/bt_node.html

[6] http://www.amelek.gda.pl/avr/

[7] AVR 8-Bit RISC - Software. http://www.atmel.com/atmel/products/prod203.htm

[8] The GNU C Library. http://www.gnu.org/manual/glibc-2.2.3/html_chapter/libc_17.html#SEC375

[9] http://baserv.uci.kun.nl/~smientki/Lego_Knex/Lego_electronica/IR_tower/IR_tower.htm

[10] RCX Internals. http://graphics.stanford.edu/~kekoa/rcx/

[11] RCX Opcode Reference. http://graphics.stanford.edu/~kekoa/rcx/opcodes.html

[12] RCX Tools. http://graphics.stanford.edu/~kekoa/rcx/tools.html#Send

[13] Lego Minstorms internals. http://www.crynwr.com/lego-robotics/

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik und Kommunikationsnetze (TIK)

Gruppe Technische Informatik

Sommersemester 2002

SEMESTERARBEIT

für
Andreas Erni und Stefan Reichmuth

Betreuer: Philip Blum, ETZ G64.2
Stellvertreter: Jan Beutel, ETZ G63

Ausgabe: 25. März 2002
Abgabe: 1.Juli 2002

Bluetooth Anbindung an Lego Mindstorms

Einleitung

Die Lego Mindstorms Roboter haben sich an vielen Unis (z.B. PPS Mindstorms an der ETH) als einfache Experimentierplatform etabliert. Diese kleinen Kreaturen sind mit einer Infrarot Schnittstelle ausgerüstet um miteinander oder mit einer Basisstation (PC) zu kommunizieren. Leider ist dies aber mit Einschränkungen verbunden, namentlich können nur kurze Distanzen bei Sichtkontakt der Sender/Empfänger überbrueckt werden. Als Lösung dieser Probleme bietet sich die Wireless-Technologie Bluetooth an.

Im Rahmen dieser Semesterarbeit soll nun eine entsprechende Lösung implementiert werden. Dazu stehen am Institut entwickelte Bluetooth Module zur Verfügung. Unter Linux soll für diese Module eine Treiberapplikation erstellt werden, welche IR-Signale einliest und sie per Bluetooth weiterverschickt, bzw. umgekehrt. Es soll ein simples Protokoll definiert und implementiert werden, mit dem einzelne Mindstormscontroller selektiv angesprochen werden koennen. .

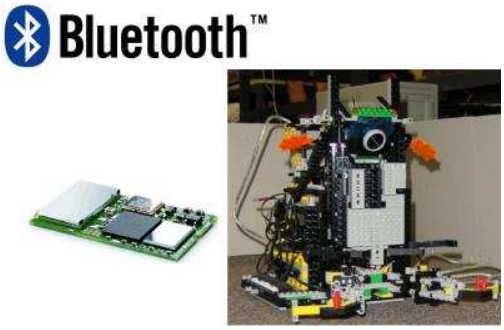

Abbildung 1: Lego Mindstorms sollen über Bluetooth Schnittstellen programmiert und gesteuert werden.

Mit Bluetooth [1] lassen sich die unterschiedlichsten Geräte drahtlos verbinden. Dabei können aber immer nur bis zu acht aktive Geräte in einem sogenannten Piconetz miteinander kommunizieren. In einem Raum können mehrere Piconetze in einem Scatternetz koexistieren. Über einen Backbone (z.B. Ethernet) können Bluetooth Geräte in unterschiedlichen Räumen dann miteinander kommunizieren.

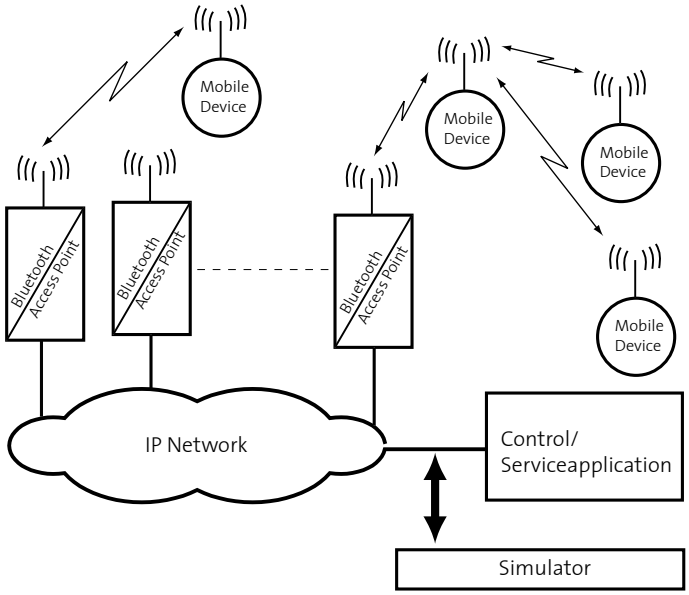

Abbildung 2: Verteilte Bluetooth Access Points und Piconetze.

Aufgabenstellung

1. Erstellen Sie einen Projektplan und legen Sie Meilensteine fest [2]. Erarbeiten Sie in Absprache mit dem Betreuer ein Pflichtenheft.
2. Führen Sie eine Literaturrecherche
3. implementieren Sie
4. Dokumentieren Sie Ihre Arbeit sorgfältig mit einem Vortrag, einer kleinen Demonstration, sowie mit einem Schlussbericht.

Durchführung der Semesterarbeit

Allgemeines

- Der Verlauf des Projektes "Semesterarbeit: Bluetooth Anbindung für Lego Mindstorms" soll laufend anhand des Projektplanes und der Meilensteine evaluiert werden. Unvorhergesehene Probleme beim eingeschlagenen Lösungsweg können Änderungen am Projektplan erforderlich machen. Diese sollen dokumentiert werden.
- Sie verfügen über PC's mit Linux/Windows für Softwareentwicklung und Test. Falls damit Probleme auftauchen wenden Sie sich an Ihren Betreuer.
- Stellen Sie Ihr Projekt zu Beginn der Semesterarbeit in einem Kurzvortrag vor und präsentieren Sie die erarbeiteten Resultate am Schluss im Rahmen des Institutskolloquiums Ende Semester.
- Besprechen Sie Ihr Vorgehen regelmässig mit Ihrem Betreuer.

Abgabe

- Geben Sie vier unterschriebene Exemplare des Berichtes spätestens am *1. Juli 2002* dem betreuenden Assistenten oder seinen Stellvertreter ab. Diese Aufgabenstellung soll vorne im Bericht eingefügt werden.

- Räumen Sie Ihre Rechnerkonten soweit auf, dass nur noch die relevanten Source- und Objectfiles, Konfigurationsfiles, benötigten Directorystrukturen usw. bestehen bleiben. Der Programmcode sowie die Filestruktur soll ausreichen dokumentiert sein. Eine spätere Anschlussarbeit soll auf dem hinterlassenen Stand aufbauen können.

Literatur

[1] Jaap C. Haartsen. The Bluetooth Radio System. *IEEE Personal Communications*, February 2000.

[2] Eckart Zitzler. Studien- und Diplomarbeiten, Merkblatt für Studenten und Betreuer. ETH Zürich, TIK, March 1998.

Date	Section	Changes
Jan. 28, 2002		Initial Version

Tabelle 1: Revision History