Mobile Roboter

Dokumentation für Mobile Roboter im Hauptstudium

Christian Seitzer (26298), Markus Bellgardt (25641), Heiko Maier (25655) und Michael Barth (26206)

30.07.2008

Inhalt

1Einführung....3
1.1Motivation....3
1.2Aufgabenstellung....3
2Vorbereitungen....4
2.1Einrichten und konfigurieren der Programmierumgebung....4
2.2Subversion, Subclipse und Trac....4
2.3AVR-Studio....4
2.4Ein zusätzlicher c't-Bot zum Vorbereiten....4
3Planung....5
3.1Team A: Grundfunktionen....5
3.2Team B: WLAN....5
3.2.1Vorüberlegungen zur Übertragung eines Befehls über WLAN....5
3.2.2Anfrage über ein Forum....5
4Vorgehensweise....7
4.1Team A....7
4.1.1Dateistruktur und Konventionen....7
4.1.2Einbindung der Fernbedienung....7
4.1.3Anlegen eines neuen Verhaltens....8
4.1.4Achter fahren, der erste Algorithmus....8
4.1.5Licht folgen-Algorithmus....9
4.1.6Linie folgen....9
4.1.7Linie folgen und Ball einsammeln....11
4.1.8Linie folgen und Ball suchen....12
4.1.9Weg aufzeichnen und abfahren....14
4.1.10Vorgehensweise beim Testen....14
4.2Grundlagen der Bot-2-Bot Kommunikation....14
4.2.1Zugriff auf die Weboberfläche per WLAN bzw. LAN....15
4.2.2UDP-Broadcast....15
4.2.3Der erste gesendete Befehl....15
4.2.4Ein zweiter c't-Bot zum Testen....15
4.2.5Der erste Test mit zwei c't-Bots....15
4.3WLAN Framework....15
4.3.1Datenübertragung (bis Revision 55)....16
4.3.2Datenübertragung – neuer Ansatz....16
4.4Das WLAN-Programm....17
5Aufgetretene Probleme und Herausforderungen....18
5.1Fehlerhafte Abgrunderkennung....18
5.2Fehlerhafte Distanzsensoren....18
5.3Ungleiche Lichtsensoren....18
5.4Roboter in Reparatur....18
5.5Roboter in Reparatur II....18
5.6Roboter in Reparatur III....18
5.7Roboter in Reparatur IV....18
5.8Speicherkartenprobleme....18
5.9Follow Line: Gefährliche 90° Rechtskurven....19
5.10Die Stromversorgung....19

5.11Übersicht herausgefundener Defekte....19
5.12Initialisierungszeit eines Ad-Hoc-WLANs....19
6Projektabschluss....20
6.1Resümee....20
7Literaturverzeichnis....21

1 Einführung

1.1 Motivation

Roboter haben in den letzten Jahren stark an Verbreitung gewonnen. Nicht mehr nur auf Fabrikhallen oder Forschungslabore beschränkt scheint es so als würden Sie in Zukunft ein integraler Bestandteil unseres Alltags sein. Sei es als selbst fahrendes Auto oder als automatischer Staubsauger.

Abbildung : Teilnehmer in der DARPA Urban Challenge von www.darpa.mil/grandchallenge

Abbildung : Staubsauger Roomba, von www.livingtools.de

Bei vielen Robotern ist es vorteilhaft, wenn Sie auf ihre Umgebung reagieren können und in der Lage sind Daten mit anderen Robotern und Systemen auszutauschen. Wir erhoffen uns durch dieses Projekt ein größeres Verständnis für diese wichtigen Fähigkeiten von Robotern.

Desweiteren überschneidet sich das Gebiet der mobilen Roboter mit dem Gebiet der Künstlichen Intelligenz (KI). Hierbei handelt es sich ebenfalls um ein sehr interessantes Gebiet auf dem große Fortschritte erwartet werden dürfen, das aber nach wie vor mit großen Schwierigkeiten zu kämpfen hat. Mobile Roboter sollten sich möglichst Intelligent verhalten um ihre Aufgaben effizient und selbstständig lösen zu können. Daher benötigen Sie eine gewisse künstliche Intelligenz. Interesse für diesen Aspekt von Robotern war auch ein Grund, dieses Projekt als Wahlfach zu wählen.

1.2 Aufgabenstellung

Die Aufgabenstellung war wie folgt vorgegeben:

> ***„Drahtlose Kommunikation zwischen mobilen Robotern***
>
> ***Das WLAN-Interface der c't-Bots ist so zu programmieren, dass mindestens zwei c't-Bots untereinander Daten austauschen können, z.B. Positionen, Fahrtrichtungen oder Sensordaten. Die Software sollte auf eine Gruppe mehrerer c't-Bots skalierbar sein.“***
> ***– Quelle: 0803MR_Projektthemen.rtf***

Es gab keine weiteren Vorgaben.

2 Vorbereitungen

Zuerst galt es, unsere Laptops einzurichten um für den c't-Bot entwickeln zu können. Dieses Kapitel beschreibt grob, welche Schritte dazu nötig waren und auf welche Anleitungen wir zurückgegriffen haben. Dieses Kapitel selbst stellt keine Anleitung dar, da es hierfür bereits genügend Quellen im Internet gibt.

2.1 Einrichten und konfigurieren der Programmierumgebung

Um für den c't-Bot entwickeln zu können, muss man einige Programme und Tools installieren und konfigurieren (1). Als Entwicklungsumgebung nutzten wir eclipse zusammen mit den *C Development Tools* (CDT) (2), wie es von *heise* empfohlen wird. Eine erste Befürchtung, die aktuelle Version 4.0.3 eigne sich nicht für die c't-Bot Entwicklung bestätigte sich nicht, es lässt sich damit ohne Probleme auch für den c't-Bot programmieren.

2.2 Subversion, Subclipse und Trac

Um simultan am Quellcode arbeiten zu können, benötigten wir einen Subversion Server. Mit *OpenSVN* war auch schnell ein freier Anbieter für unser Projekt gefunden (3). Zusätzlich zu Subversion hatte OpenSVN noch den Vorteil, ein Trac System (4) zur Projektverwaltung zur Verfügung zu stellen, womit wir grobe *Milestones* planen, sowie *Tickets* zu Bugs erstellen konnten. Desweiteren bietet Trac eine integrierte *MediaWiki*, welche wir jedoch, von einer Startseite mit Link-Sammlung abgesehen, nicht benötigten.

Als Schnittstelle zwischen eclipse und dem Subversion Server nutzten wir das OpenSource Projekt *Subclipse* (5).

2.3 AVR-Studio

Um unseren Code auf die c't-Bots zu transferieren nutzten wir das Programm *AVR Studio*. Zu Beginn installierten wir Version 4.12, die auf den mitgelieferten AVR-CDs enthalten war. Jedoch hatten wir wegen einer Vista-Inkompatibilität mit einigen Abstürzen zu kämpfen. Linderung des Problems verschaffte die neuere Version 4.14 des Programmes

2.4 Ein zusätzlicher c't-Bot zum Vorbereiten

Unser Betreuer Professor Hellmann brachte uns während des Projekts einen neuen, fast fertig zusammengebauten Roboter mit *WiPort* Modul. Diesem haben wir noch den WiPort aufgeschraubt und danach einige Tests durchgeführt. Anschließend wurde der Roboter für WLAN vorbereitet (siehe 4.2 Grundlagen der Bot-2-Bot Kommunikation).

Unglücklicherweise lies sich der Roboter nicht mit AVR-Studio flashen und da wir als Software-Team den Fehler nicht gefunden haben wurde der Roboter an Professor Hellmann zurückgegeben. Schade, wir hätten einen zweiten Roboter für unsere Gruppe gut gebrauchen können, der nicht von einer anderen Gruppe verwendet wird (siehe 4.2.4 Ein zweiter c't-Bot zum Testen). Hierdurch wären zusätzliche Konfigurationsarbeiten entfallen.

3 Planung

Zu Beginn stellte sich die Frage, wie unsere Gruppe an das Mobile Roboter Projekt herangehen sollte. Da wir zu viert in der Gruppe waren, wurde entschieden die Gruppe in zwei Teams aufzuteilen:

- Team A sollte sich um die Grundfunktionen kümmern
- Team B die WLAN Möglichkeiten erforschen und austesten

Das bot den Vorteil das Team B sich bereits frühzeitig um die WLAN Möglichkeiten des c't-Bots kümmern konnte, während Team A Erfahrung mit der c't-Bot-Programmierung selbst sammelte. Desweiteren bot sich ein 2er Team an für Ansätze des *Extreme Programming* Paradigmas an bei dem der Eine programmiert und der Andere mitdenkt (*Pair Programming*, siehe (6)).

Die nähere Planung der Teams sah wie folgt aus.

3.1 Team A: Grundfunktionen

Um einen Einstieg in die Materie zu erhalten und die Grundlagen aus dem Weg zu haben, begann Team A mit dem Lesen diverser c't Artikel zum c't-Bot (7) (8) (9) und anschließend mit der Implementierung einfacher Grundfunktionen, welche auch von Studenten des Grundstudiums erfüllt werden mussten.

Wir bearbeiteten die Grundfunktionen in folgender Reihenfolge:

1. Achter fahren (Aufgabe 1, c)
2. Licht folgen/fliehen (Aufgabe 1, a)
3. Linie folgen (Aufgabe 1, b)
4. Ball einsammeln (Aufgabe 2, a)
5. Ball suchen und einsammeln (Aufgabe 2, b)
6. Weg aufzeichnen und abfahren (Aufgabe 1, d)

3.2 Team B: WLAN

3.2.1 Vorüberlegungen zur Übertragung eines Befehls über WLAN

Am Anfang galt es, die weitere Vorgehensweise festzulegen. Als erstes haben wir uns für eine Analyse des vorhandenen Quellcodes entschieden. Dort einen Überblick zu bekommen war aufwändig, da die Dokumentation des Quellcodes von der c't nicht immer optimal und ausführlich genug für unsere Zwecke war.

Nachdem wir uns einen Überblick verschafft hatten, stellte sich die Frage wie wir überhaupt auf die WLAN-Schnittstelle zugreifen können. Wir fanden keine Hinweise im Quellcode oder auf *heise.de*. Eine Recherche bei Google erbrachte zwar viele allgemeine Hinweise zum c't-Bot und auch zum WLAN-Modul von *Lantronix*, aber bis auf eine einzige Seite, die leider auch keine Programmbeispiele enthielt, fanden wir nichts Weiterführendes zum Thema c't-Bot WLAN Programmierung.

3.2.2 Anfrage über ein Forum

Wir haben uns dann an ein Forum für c't-Bot Fans gewandt, dass wir bei unserer Google-Recherche entdeckt haben. Unsere Anfrage dort (6) nach Beispielen wurde sehr schnell von einem netten Foren-User beantwortet. Dieser hat uns auf den `devel`-Zweig des Subversion Servers von c't hingewiesen. In diesem sollte eine Art *Bot-2-Bot Kommunikation* über den c't-Sim als Server implementiert sein. Wir haben uns dann diese Version heruntergeladen und wollten das ganze ausprobieren um zu sehen in wie weit wir Teile von diesem Code für unsere Zwecke einsetzen könnten.

Leider hat dies auch nach längerem Testen nicht funktioniert. Zuerst lag es an daran, dass wir zwar den neusten `devel`-Code des c't-Bot hatten, aber den `stable`-Zweig des c't-Sim, danach hat uns die Fehlersuche leider nicht weiter geholfen. Da wir so nicht wirklich weitergekommen sind und die Aufgabenstellung eigentlich auch auf eine direkte Kommunikation zwischen zwei Robotern schließen lässt, haben wir uns entschieden mit dem Test des ct-Sim wieder aufzuhören.

4 Vorgehensweise

Im Folgenden Kapitel erklären wir unsere Vorgehensweise beim Projekt. Allgemein ist zu sagen, dass wir jeden Montag von 9:30 Uhr bis 11:00 Uhr, sowie Dienstag von 11 Uhr bis 17:30 Uhr (außer, es war Grundseminar Vorlesung, inklusive Mittagspause von einer Stunde) und Mittwochs von 9:30 Uhr bis 12 Uhr am Mobile Roboter Projekt im Labor von Professor Hellmann arbeiteten. Es wurde auch unregelmäßig außerhalb dieser Zeiten am Mobile Roboter Projekt gearbeitet.

Um Stress zur Prüfungszeit zu vermeiden, begannen wir bereits frühzeitig (ab dem 31. März) intensiv am Mobile Roboter Projekt zu arbeiten.

4.1 Team A

Nach lesen der angesprochenen c't Lektüren in 3.1 wagten wir uns an ein erstes Programm. Basierend auf den Tipps von c't modifizierten wir `behaviour_simple.c` für eine erste Version des Achterfahren Programms. Dies funktionierte auch ohne Probleme, mal davon abgesehen dass die erste 8 nicht wirklich wie eine 8 aussah, aber das war ja nur Feintuning der Parameter. Das erste Ziel war geschafft: Ein eigenes *behaviour* in das behaviour-Framework des c't-Bots einfügen.

4.1.1 Dateistruktur und Konventionen

Damit klar erkennen bar ist, welche Dateien wir im Zuge des Mobile Roboter Projekt angelegt haben und welche zum Standard c't-Code gehören, haben wir einen eigenen Ordner im Projektverzeichnis angelegt, welchen wir `mobirobo` genannt haben. Dort in einem Unterverzeichnis befindet sich auch eine Kopie dieser Dokumentation. Alle *behaviours* sind gespeichert unter `mobirobo\bot-logic`.

Desweiteren beginnen alle selbst erstellten Funktionen, Dateien und Definitionen mit dem `mr_` bzw. `MR_` präfix. Zum Abschluss erstellten wir noch ein Verzeichnis für die Header-Dateien der behaviours unter `include\mobirobo\bot-logic\`.

Modifikationen an c't-Dateien wurden stehts mit Kommentaren umschlossen wie z.B.

```
// *MobiRobo modifications
source code…
// ~MobiRobo modifications
```

Ein kurzes Suchen nach "MobiRobo" sollte also genügen um alle modifizierten Stellen ausfindig zu machen.

4.1.2 Einbindung der Fernbedienung

Um unsere Programme einfacher testen und präsentieren zu können haben wir uns entschlossen den Code für die Fernbedienung so umzuschreiben, dass man unsere Programme über die Nummerntasten aktivieren kann.

Will man mit der Fernbedienung alle behaviours deaktivieren(an/aus-Taste), so betrifft das nur behaviours mit einer Priorität größer 3 und kleiner gleich 200. Vermutlich wurde dieser Ansatz gewählt damit essentiell wichtige behaviours wie z.B. die Abgrunderkennung immer aktiv bleiben.

Nachfolgend eine Zuordnungsliste der aufrufbaren Programme.

Ziffer	Programm
1	Achter fahren

2	Licht folgen
3	Vor Licht fliehen
4	Linie folgen
5	Linie folgen und Ball aufsammeln
6	Linie folgen und Ball suchen

4.1.3 Anlegen eines neuen Verhaltens

Um ein neues Verhalten einzufügen sind wir folgendermaßen vorgegangen:

Im Ordner `bot-logic` kopierten wir zuerst `behaviour_simple.c`. Dieser Kopie wurde ein zum Verhalten passender Name gegeben und alle Vorkommnisse des Wortes `simple` in der Datei wurde ersetzt durch den Namen unseres neuen Verhaltens.

Im Ordner `include\bot-logic` musste dasselbe für eine passende Header-Datei getan werden.

In `bot-logic\bot-logic.c` musste danach noch das Verhalten in die `behave_init`-Methode eingetragen werden. Damit es bei jedem Aufruf von `bot_behave` ausgeführt wird. Dieses Beispiel für einen Eintrag ist für das Achter fahren Programm:

```
#ifdef MR_BEHAVIOUR_DRIVE_EIGHT_AVAILABLE
      insert_behaviour_to_list(&behaviour,
      new_behaviour(199, mr_bot_drive_eight_behaviour, INACTIVE));
#endif
```

Der erste Parameter für `new_behaviour` stellt dabei die Priorität dar. Je höher diese ist, desto öfter wird das Unterprogramm aufgerufen. Der zweite Parameter ist der Name der Funktion in der c-Datei, in welcher der eigentliche Code für das Verhalten steht. Als letzten Parameter kann man entweder `INACTIVE` oder `ACTIVE` übergeben. Dies legt fest ob das Verhalten beim Start des c‘t-bots aktiv wird oder erst später aktiviert werden muss.

Als letztes muss man noch in `include/bot-logic/available_behaviours.h` eine `include`-Anweisung für die Header-Datei des Verhaltens schreiben. In dieser Datei befinden sich auch die globalen Schalter für die einzelnen behaviours. Hier muss noch ein `#define` für die Konstante eingetragen werden, auf die auch im behaviour mit `#ifdef` geprüft wird.

Jetzt kann man in der angelegten c-Datei seinen eigenen Code an die Stelle des Codes aus dem `simple_behaviour` setzen.

4.1.4 Achter fahren, der erste Algorithmus

Nachdem die Grundlage erprobt war, machten wir uns einzeln an das Umsetzen je einer der Aufgaben: Den Achter fahren und Licht folgen Algorithmen.

Der Algorithmus zum Achter fahren beruht vom Prinzip her schlicht auf einem Zustandsautomaten. Unser erster Ansatz bestand darin, dass der Bot mithilfe der `bot_drive_distance`-Methode vier Viertelkreise fahren sollte, die zusammengesetzt eine Acht ergeben. Da sich der korrekte Winkel und die Länge der Strecke für jeden Halbkreis jedoch schwerer ermitteln ließen als gedacht gingen wir

dazu über die Acht mit Hilfe von 2 vollen Kreisen zu fahren. Hierbei müssen nur zweimal die richtigen Werte eingetragen werden. Nachfolgend der Code des Algorithmus:

```
switch (state) {
case 0:
      bot_drive_distance(data, 60, BOT_SPEED_MAX, 90);
      state = 1;
      break;
case 1:
      bot_drive_distance(data, -60, BOT_SPEED_MAX, 90);
      state = 2;
      break;
case 2:
      display_printf("Stop");
      display_cursor(1,1);
      break;
default:
      return_from_behaviour(data);
      break;
}
```

Bei Initialisierung befindet sich der Bot im `state = 0`. Dies löst das Abfahren des ersten Kreises der 8 aus, ebenso wechselt der Bot damit auf `state = 1`. Beim nächsten Aufruf des behaviours wird dann ein anderer Zweig der `switch-case` Verschachtelung aufgerufen, welcher einen Kreis in die andere Richtung abfährt. Damit ist die 8 abgefahren und der Bot wechselt in `state = 2`, welcher den Bot schlicht eine Statusmeldung auf dem Display ausgeben lässt.

Das Standardverhalten bei einem Unbekannten Zustand ist das Verlassen des behaviours.

Probleme an diesem Programm sind, dass man für jeden c‘t-Bot die Werte neu ermitteln muss durch ausprobieren, da jeder Bot geringfügig andere Fahreigenschaften besitzt. Außerdem variiert die 8 stark je nachdem auf welchem Untergrund sie gefahren wird.

4.1.5 Licht folgen-Algorithmus

Der Licht folgen-Algorithmus stellte keine all zu große Hürde dar, schließlich gab es schon einen c’t Artikel welcher einen solchen vom Prinzip beschrieb. (9) Das behaviour war schnell geschrieben, jedoch sollten Tests einiges mehr an Zeit benötigen. Die Richtung der Lichtquelle wird ermittelt durch die Differenz der Lichtsensorwerte Links und Rechts.

```
curve = (sensLDRL - sensLDRR)/1.5;
```

Aufgrund des Ergebnisses dieser Differenz wurde dann die Kurvenstärke gesetzt (negativ: fahre nach rechts, positiv: fahre nach links, 0: fahre gerade aus). Die Division von 1.5 reguliert das Ergebnis um die hohen Werte an die niedrigeren Kurvenwerte des Bots anzupassen.

4.1.6 Linie folgen

Bei diesem Programm fährt der c‘t-Bot mit Hilfe seiner Liniensensoren eine Linie entlang. Durch ausprobieren fanden wir einen Wert ab dem man sicher sein kann, dass der Sensor sich auf einer Linie befindet. Dieser Wert ist als `LINE_THRESHHOLD` im Code zu finden. Da mehrere Roboter nicht mit korrekt funktionierenden Liniensensoren ausgestattet sind, haben wir speziell für den rechten Sensor einen weiteren Wert eingebaut, `LINE_THRESHHOLDR`. Wir haben also zwei Sensoren, die sich entweder auf einer Linie befinden oder nicht. Dies macht insgesamt 4 Fälle, die betrachtet werden müssen.

Fall	Linker	Rechter

	Sensor	Sensor	
1	Linie	Linie	c‘t-Bot ist zu weit nach rechts gefahren oder ist bei einer 90°-Linkskurve zu weit gefahren. In beiden Fällen bringt eine Drehung nach links den c‘t-bot wieder zurück zur Linie und damit zu Fall 2.
2	Nichts	Linie	c‘t-Bot ist auf Kurs und fährt gerade aus.
3	Linie	Nichts	c‘t-Bot ist zu weit nach rechts gefahren und muss deshalb nach links fahren um wieder auf korrekten Kurs zu kommen. Dieser Fall sollte im Normalfall nicht auftreten, da der c‘t-Bot damit über die Linie gefahren wäre, was Fall 1 verhindert.
4	Nichts	Nichts	c‘t-bot ist entweder zu weit nach links gefahren oder er ist bei einer 90°-Rechtskurve zu weit gefahren. In beiden Fällen bringt eine Drehung nach rechts den c‘t-Bot wieder zurück zur Linie und damit zu Fall 2.

Nachfolgend der Quellcode welcher diese Fälle abbildet:

```
#define LINE_THRESHHOLD 100
#define LINE_THRESHHOLDR 100

if(sensLineR > LINE_THRESHHOLDR) {
	if(sensLineL >= LINE_THRESHHOLD) {
		speedWishLeft = -BOT_SPEED_NORMAL;
		speedWishRight = BOT_SPEED_NORMAL;
	}
	else {
		speedWishLeft = BOT_SPEED_NORMAL;
		speedWishRight = BOT_SPEED_NORMAL;
	}
}
else {
	if(sensLineL >= LINE_THRESHHOLD) {
		speedWishLeft = -BOT_SPEED_NORMAL;
		speedWishRight = BOT_SPEED_NORMAL;
	}
	else {
		speedWishLeft = BOT_SPEED_NORMAL;
		speedWishRight = -BOT_SPEED_NORMAL;
	}
}
```

4.1.7 Linie folgen und Ball einsammeln

Da man genau weiß dass der Ball sich am Ende der Linie befindet weiß man auch, wo sich das Ende der Linie befindet. Eine Überprüfung nach dem Linienende entfällt also.

Der c‘t-Bot öffnet gleich zu Beginn seine Klappe und fährt dann exakt wie beim Linie-folgen Algorithmus die Linie ab. Der einzige Unterschied ist, dass er ständig die Lichtschranke überprüft. Sobald die Lichtschranke im Inneren des Transportfaches unterbrochen wird ist klar, dass der c‘t-Bot den Ball eingesammelt hat und sich am Ende der Linie befindet. Er hält an und schließt die Klappe.

Natürlich dürfen hierfür keine anderen Gegenstände auf der Linie liegen.

```
switch (state) {
case 0:
      bot_servo(data, SERVO1, DOOR_OPEN); // Klappe auf
      state = 1;
      break;

case 1:
      Linie folgen(siehe 4.1.6 Linie folgen)
      if (sensTrans !=0) {
            bot_servo(data, SERVO1, DOOR_CLOSE); //Klappe zu
            state = 2;
      }
      break;
case 2:
      display_printf("Stop");
      display_cursor(1,1);
      break;

default:
      return_from_behaviour(data);
      break;
}
```

4.1.8 Linie folgen und Ball suchen

Dieses Programm war eines der Schwierigsten. Sowohl das Erkennen des Linienendes, sowie das Finden des Balles stellten Probleme dar welche reichlicher Überlegung bedurften.

Der c‘t-Bot fährt eigentlich genau wie beim Linie folgen Algorithmus der Linie nach. Der Unterschied liegt darin, dass der c‘t-Bot die Zeit misst sobald beide Sensoren keine Linie anzeigen und er sich deswegen nach rechts dreht. Übersteigt diese Zeit eine bestimmte Grenze nimmt der c‘t-Bot an, dass er das Ende der Linie erreicht hat und schaltet in den Ballsuchmodus. In diesem nutzt er die Distanzsensoren um den Ball aufzuspüren. Durch Ausprobieren wurde ein Wert ermittelt bei dem die Distanzsensoren gut anschlagen. Dieser Wert wurde dazu genutzt um zu ermitteln ob sich etwas vor dem jeweiligen Sensor befindet oder nicht. Wie beim Linie-folgen-Algorithmus hat man also zwei Sensoren mit jeweils zwei verschiedenen Zustandsmöglichkeiten. Dies ergibt wieder 4 verschiedene Fälle.

Fall	Linker Sensor	Rechter Sensor	
1	Ball	Ball	Da der Ball nicht so groß ist kann dieser Zustand eigentlich nicht auftreten. Das heißt der c‘t-Bot muss wohl vor einer Wand oder einem anderen größerem Hindernis stehen. Deshalb dreht er sich eine zufällige Anzahl von Grad.
2	Nichts	Ball	Rechts könnte ein Ball sein =>Drehe dich nach rechts und speichere die Zeit
3	Ball	Nichts	Links könnte ein Ball sein =>Drehe dich nach links und ziehe die gespeicherte Zeit aus Fall 2 von der momentanen Zeit ab. Liegt diese innerhalb eines bestimmten Intervalls, so erhöhe den Counter um 1. Hat der Counter 20 erreicht so fahre gerade aus.
4	Nichts	Nichts	Kein Ball in Sicht => drehe dich nach rechts

Der Counter in Verbindung mit der Zeitprüfung sorgt dafür, dass der c’t-Bot nicht durch einen bloßen Sensorfehler in die Irre geführt wird.
Nachfolgend der Source Code für diesen Teil des Programms:

```
if(sensDistR < BALL_SENSE) {
      if(sensDistL >= BALL_SENSE) {
            speedWishLeft = BOT_SPEED_NORMAL;
            speedWishRight = -BOT_SPEED_NORMAL;
            timesave = ticks;
      }
      if(sensDistL < BALL_SENSE) {
            bot_turn(data, rand() % 360);
      }
}
else{
      if(sensDistL >= BALL_SENSE) {
            speedWishLeft = BOT_SPEED_NORMAL;
            speedWishRight = -BOT_SPEED_NORMAL;
      }
      if(sensDistL < BALL_SENSE) {
            speedWishLeft = -BOT_SPEED_NORMAL;
            speedWishRight = BOT_SPEED_NORMAL;
                  if(ticks-timesave > 1000 &&
                        ticks-timesave < 10000) {
                  count++;
            }
            timesave = ticks;
            if(count > 20) {
                  state = 5;
            }
      }
}
```

4.1.9 Weg aufzeichnen und abfahren

Diese Funktion war die Letzte die wir angingen. Leider wurde diese nicht erfolgreich programmiert, was unter anderem an dem Problem mit der Speicherkarte lag (siehe 5.8), dessen Lösung erst eine Woche vor Abgabe gefunden wurde. In der derzeitigen Version sind Ansätze zum Speichern des abgefahrenen Weges vom Licht folgen-Programm in einer Variablen implementiert sowie Ansätze für eine Abspielfunktion. Leider funktioniert es nicht und für weitere Tests und die Einbindung der Speicherkarte blieb keine Zeit.

4.1.10 Vorgehensweise beim Testen

Der erste Test, den jedes behaviour bestehen musste war im c’t-Sim. Das hatte folgende Gründe: Zum einen ermöglichte c’t-Sim ein schnelles Testen der Änderungen. Compilen und starten genügen bereits, das Flashen des Bots entfiel. Darüberhinaus stellt c’t-Sim ein isoliertes System dar, welches keinen willkürlichen Umwelteinflüssen oder mechanischen Fehlfunktionen ausgesetzt ist (ähnlich wie physikalische Modelle). Dies ist eine wichtige Eigenschaft für eine Testumgebung, da hiermit die Gefahr ausgeschlossen wird das unbemerkte Einflüsse das Ergebnis der Auswertung verfälschen.

Bestand das behaviour den Test im c't-Sim wurde auf dem realen Bot getestet. Das stellte sich als schwierig, unzuverlässig und zeitaufwändig heraus, da jeder Bot eine andere Macke mit sich brachte (asynchrone Sensorwerte, unkalibrierte Sensorwerte, defekte Sensoren, mechanische Probleme sowie unerklärliche Probleme) und das Programm sich auch anders verhielt als im Simulator. Oft mussten verschiedene Ansätze und Werte ausprobiert (am Besten mehrmals, um willkürliche Einflüsse auszuschließen) und immer wieder angepasst werden, was jedesmal compilen und flashen des Bots bedeutete.

4.2 Grundlagen der Bot-2-Bot Kommunikation

Dieses Kapitel beschreibt, welche Anstrengungen und Recherchen nötig waren, die Grundlagen für eine Bot-2-Bot Kommunikation zu schaffen.

4.2.1 Zugriff auf die Weboberfläche per WLAN bzw. LAN

Da sich der c't-Bot bzw. der *WiPort* nicht im Fabrik Zustand befand und die Gruppe, die ihn letztes Semester benutzt hatte, die IP-Adresse nicht nennen konnte war unsere erste Aufgabe die IP des Wiports herauszufinden. Dies war schwieriger als zunächst erwartet: Unsere erste Idee durch ausprobieren einiger bekannter Adressen scheiterte leider. Die Gruppe hatte wohl eine „exotische" Adresse gewählt.

Über den *DeviceInstaller* von Lantronix gelang uns auch keine Kontaktaufnahme zum Roboter, da auch hierfür der IP-Bereich bekannt sein muss. Somit kamen wir zur letzten und schließlich erfolgreichen Möglichkeit: Mit dem USB-2-Bot-Adapter und einem Hyperterminal Programm konnten wir eine neue IP setzen(7). Aber auch hier gab es wieder Probleme: In Windows Vista gibt es kein integriertes Hyperterm mehr. Außerdem mussten die Treiber für den USB-2-Bot Adapter gesucht werden. Nachdem wir Treiber gefunden hatten und eine Hyperterm-Alternative für Windows Vista (*HTerm 0.6.5beta*) fanden, versuchten wir eine erste Kontaktaufnahme.

Wie in der Anleitung beschrieben drückten wir die x-Taste während der c't-Bot ein- und ausgeschaltet wurde. Nach einigen Versuchen kam die Aufforderung, dass wir die Enter-Taste drücken sollten, um ins Setup-Menü zu gelangen. Leider haben wir keine Möglichkeit gefunden mit HTerm die Enter-Taste zu drücken.

Als nächste Hyperterminal Alternative haben wir putty getestet. Nach den korrekten Einstellungen hat dies auch funktioniert. Wir konnten die Enter-Taste drücken und waren im lang ersehnten Setup-Menü. Dort haben wir dann die IP-Adresse auf *192.168.0.1* gesetzt (gesetzt war Sie auf 192.168.1.100). Nachdem wir nun die IP-Adresse bestimmt hatten konnten wir über einen Browser auf die Web-Oberfläche von WiPort zugreifen. Diese forderte uns auf ein Passwort einzugeben, doch glücklicherweise war hier kein Passwort gesetzt und als Username wurde „admin" verwendet. Dies haben wir nach einigen falschen Eingaben herausgefunden.

4.2.2 UDP-Broadcast

Wir haben bei unserer weiteren Recherche noch ein weiteres Projekt gefunden: die *Firefighters* (8). Ein Projekt in welchem c't-Bots per WLAN kommunizieren und mit einer Kamera Erkennung Aufgaben lösen können. Leider gibt es auf der Seite keinen Quellcode, aber wir fanden eine weitere Idee zur Bot-2-Bot Kommunikation.

Die Bots werden auf UDP-Broadcast gestellt und somit werden die Daten an alle Bots gesendet. In die Pakete wird dann noch eine Quell- und Ziel-Adresse integriert und fertig ist die Bot-2-Bot Kommunikation – soweit die Theorie.

Die praktische Umsetzung hat natürlich wieder Ihre Probleme. Der Versuch „einfach“ unseren Bot auf UDP-Broadcast einzustellen ist gescheitert, denn die vorhandene Firmware 6.3.0.0 unterstützt diese Funktion leider nicht wie wir vermuteten. Auf der Suche nach einem Firmware-Update wurde dies bestätigt dann bestätigt: Bei Firmware 6.5.0.0 wurde der UDP-Broadcast als neues Feature genannt.

Wir haben uns danach dann über ein Firmware Update für den WiPort unseres c‘t-Bots umgesehen. Leider haben wir auch hierzu keinerlei Informationen auf den Seiten der c’t oder in Foren gefunden. Nach einer weiteren Recherche im Web und dem konsultieren der Anleitung des WiPorts wussten wir das mit dem DeviceInstaller ein Firmware Update möglich ist. Wir haben uns auf den Seiten von Lantronix die aktuellste Firmware (Version 6.6.0.0) heruntergeladen und auf den Bot geflashed.

Nach erneutem Suchen der Broadcast Funktion in der Weboberfläche wurden wir enttäuscht – diese war nicht vorhanden. Unsere nächste Idee war nochmals per USB-2-Bot-Adapter direkt mit putty in die Einstellungen des WiPorts zu schauen und tatsächlich gab es in einem Untermenü den Punkt *Broadcast* den wir dann auf *Yes* gesetzt haben. Wie der Firefighters Webseite zu entnehmen ist, war der c’t-Bot mit dem WiPort nun in der Lage alles was auf die serielle Leitung gesendet wird per WLAN zu versenden. Dank des Broadcasts können wir also nun Pakete an alle Bots im selben Adhoc-WLAN senden.

4.2.3 Der erste gesendete Befehl

Wir haben in die vorhandene `command.c` ein Test-Kommando integriert (`CMD_WLAN_HELLO_WORLD`) das lediglich wenn es empfangen wird „Hallo Welt“ auf dem Display des c’t-Bots ausgibt. Da unserer Gruppe zu diesem Zeitpunkt nur ein Bot zur Verfügung stand war ein direkter Test leider nicht möglich. Wir haben dann mit einem WLAN-Paket-Sniffer (*Wireshark*, der Nachfolger des bekannten *Ethereal*) geschaut ob die Pakete tatsächlich gesendet werden: Es hat funktioniert.

Wir sehen die Pakete am Notebook, die von unserem Roboter (IP: 192.168.0.1) gesendet werden. Da wir uns nicht so gut mit WLAN-Paketen auskennen konnten wir nichts über den Inhalt sagen.

4.2.4 Ein zweiter c’t-Bot zum Testen

In Absprache mit der zweiten WLAN Gruppe durften wir ab sofort zum Testen auch Ihren Roboter verwenden. Vielen Dank an die andere Gruppe an dieser Stelle nochmal. Da dieser Roboter genau dieselben Probleme hatte (unbekannte IP-Adresse, veraltete WiPort Firmware ohne UDP-Broadcast Funktion) Da die Vorgehensweise schon bekannt war ging dies natürlich deutlich schneller als beim ersten Mal.

4.2.5 Der erste Test mit zwei c’t-Bots

Nachdem wir dem zweiten Roboter eine kleine Testroutine für WLAN-Empfang geschrieben und auf den Roboter geflashed haben, hat er nach beheben einiger kleiner Programmierfehler das *Hallo Welt* Test Kommando empfangen. An dieser Stelle wurde unser wichtigstes Ziel erreicht: Die Bot-2-Bot Kommunikation bzw. eine Vorstufe (ein Bot sendet es an alle anderen Bots, was bei zwei Bots gesamt natürlich nur einer ist) war fertig!

4.3 WLAN Framework

Unsere nächste Aufgabe bestand darin eine neue Datei namens `wlan.c` zu erstellen in die wir ab sofort unsere selbst programmierten WLAN-Funktionen einfügen.

Die Kommunikation basiert auf einer klassischen Client/Server Architektur, wobei der erste Bot der Master (Server) wird und alle weiteren Bots Slaves (Clients). Jeder Bot bekommt eine eigene Adresse und wird in eine Liste (beim Master-Bot) eingetragen. Die „Absprache“ der Bots basiert auf

Kommandos, die zwischen Ihnen hin- und her geschickt werden. Der Ablauf wird im Folgenden kurz beschrieben.

1. Bot wird eingeschaltet und WLAN wird initialisiert (`wlan_init()`)
2. Bot sendet `SUB_WLAN_REQUEST_MASTER` Command (`master_available()`)
3. Hier muss zwischen zwei Fällen unterschieden werden:
    a. Falls ein Master vorhanden ist, sucht dieser nach der nächsten freien Adresse (`get_unused_address()`), trägt ihn in seine Liste ein (`insert_bot()`)und sendet ein `SUB_WLAN_IS_MASTER` Command zurück.
    b. Falls kein Master vorhanden ist bleibt der Status undefiniert und das Flag für eine bestehende Verbindung wird nicht gesetzt.

Ein Bot kann erst Master werden, nachdem er selbst keine Antwort auf seine Frage nach einem Master erhält und ein anderer Bot auch nach einem Master sucht. Dies bemerkt der Bot und ernennt sich selbst dann zum Master, welcher die Adressverwaltung übernimmt. In der nachfolgenden Tabelle sind die Status beschrieben, die der Bot annehmen kann.

Status	Beschreibung
BOT_MODUS_MASTER	Der Bot ist Master
BOT_MODUS_SLAVE	Der Bot ist Sklave
BOT_MODUS_UNDEFINED	Der Modus ist noch nicht definiert (z.B. nach dem Anschalten)

Dazu kann jeder Bot das *connection flag* (`bot_connected`) auf `BOT_IS_CONNECTED` bei einer bestehenden Verbindung, oder auf `BOT_IS_NOT_CONNECTED` bei keiner bestehenden Verbindung setzen.

Damit funktioniert der Verbindungsaufbau und die Bots sind bereit zum Übertragen von Daten.

4.3.1 Datenübertragung (bis Revision 55)

Hierfür wurde eine Funktion mit dem Namen `wlan_send_data()` geschrieben. Dieser wird die Sende- und Empfangsadresse der Bots sowie die Daten als `char`-Pointer übergeben. In dieser Funktion werden die Daten mittels der Funktion `command_write_data()` (bereits von der c't integriert) versendet. Dieser werden natürlich die Sende- und Empfangsadresse, der `char`-Pointer auf die Daten sowie zusätzlich als Kommando `CMD_WLAN` und als Sub-Kommando `SUB_WLAN_STRING` für die Übertragung der Textdaten übergeben.

Der Datenempfang funktioniert auf folgende Weise: Alle Bots rufen periodisch die Funktion `wlan_update()` auf, die mittels der Funktion `uart_data_available()` (bereits von der c't integriert) prüft, ob neue Daten angekommen sind. Falls gültige Kommandos angekommen sind, wird die Funktion `command_evaluate()` aufgerufen, die auch von der c't geschrieben wurde. Wir haben diese so modifiziert, dass alle WLAN-Kommandos an unsere `wlan_evaluate()`

weitergeleitet werden. Dort schaut der Bot dann ob die Daten für ihn (seine Bot-Adresse) oder für alle (Bot-Adresse: 255) bestimmt sind. Falls ja gibt er die Daten auf seinem Display aus.

Leider ist diese Übertragung sehr fehlerbehaftet, so dass z.B der String „Hallo Welt“ versendet wird und bei den empfangenden Bots dann z.B „Hallo Weltlt“ oder „H%allo Welt“ ankommt. Kurze Texte (wie das eben genannte Beispiel) kommen auch hin und wieder korrekt an, sobald aber viele Daten auf einmal übertragen werden, kommen diese nie korrekt an und haben sehr viele Fehler. Bei dem Versuch eine Checksumme zu generieren stellten wir leider fest, dass größere Strings so gut wie nie korrekt ankommen. Wir haben deshalb nach einer weiteren Möglichkeit gesucht und auch eine gefunden, mehr dazu im nächsten Abschnitt.

4.3.2 Datenübertragung – neuer Ansatz

Da wir gemerkt haben, dass die Master/Slave-Aushandlung so gut wie immer fehlerfrei ist, haben wir versucht, die Funktion welche für das Senden von Kommandos gedacht ist, einfach für die Übertragung von Sensordaten zu missbrauchen. Um das ganze ohne große Umstände nutzen zu können, haben wir die zwei Funktionen `wlan_send_cmd()` und `wlan_send_data()` erstellt. So können wir wahlweiße Kommandos oder Daten senden.

Um Sender und Empfänger weiterhin benennen zu können, nutzen wir nun 4 Bit (haben also 15 Adressen, wobei wir 15 als Broadcast nutzen). Gespeichert werden beide zusammen in einer 16 Bit Variablen. Da wir somit zum Speichern der Nutzdaten auch nur 16 Bit haben, senden wir die Werte der Sensoren nacheinander. Die Funktion „`wlan_toggle_send`“ speichert hierbei, für welchen Sensor zuletzt Daten übertragen wurden, sodass beim nächsten Senden die Daten für den anderen Sensor übertragen werden.

Beim Empfänger hat man das Problem, dass für das Darstellen der Daten auf dem Display die Wiederholrate zu hoch ist. Daher wird in der Variablen „`last_cmd`“ gespeichert, welche Art von Daten übertragen wurden und ggf. nur der alte Wert von dem neuen überschrieben.

4.4 Das WLAN-Programm

Mit dem nun vorhandenen Framework haben wir eine Anwendung geschrieben, die es ermöglicht Sensordaten zu übertragen. Der jeweilige Master Bot sendet an alle verbundenen Slave Bots je nach Belegung der Variablen `send_mode`, die per Fernbedienung auf einen Wert zwischen 1 und 4 gesetzt wird, unterschiedliche Sensordaten:

Ziffer	Daten die versendet werden
1	Linker und Rechter Distanz Sensor
2	Linker und Rechter Licht Sensor
3	Linker und Rechter Linien Sensor
4	Linker und Rechter Border Sensor

Über den Wert `WLAN_DATA_RECV` kann auch ein spezieller Empfänger für die Daten festgelegt werden, so dass die Sensordaten nicht von allen Bots empfangen werden sondern nur von einem speziellen. Die aktuell festgelegte Sendefrequenz ist 100 ms (`WLAN_DATA_FREQ`), d.h. alle 100 ms werden die aktuellen Sensordaten versendet und kurze Zeit später von den empfangenden Bots angezeigt.

5 Aufgetretene Probleme und Herausforderungen

Nachfolgend eine Auflistung und Erläuterung aller Probleme welche im Laufe dieser Projektarbeit auftraten, sowie eine Beschreibung wie diese gelöst wurden.

5.1 Fehlerhafte Abgrunderkennung

Das erste Programm, welches wir geschrieben und ausprobiert hatten war, den c't-Bot in Form einer 8 fahren zu lassen. Als wir dieses Programm auf dem realen Bot ausprobierten kam es jedoch zu folgendem seltsamen Verhalten: Auf dem Tisch fuhr er die 8, wie im Simulator. Sobald wir ihn jedoch auf den Boden setzten ignorierte der c't-Bot das Verhalten, das wir für ihn programmiert hatten und fuhr die ganze Zeit rückwärts.

Des Rätsels Lösung fanden wir nach einiger Zeit in einem Forum. Da der Boden im Labor recht dunkel ist nahm das Programm an, dass der Bot vor einem Abgrund stehen würde und startete das entsprechende "`Avoid Border`"-Verhalten. Nachdem wir dieses Verhalten auf `INACTIVE` gesetzt hatten fuhr der Bot auch auf dem Boden die 8.

5.2 Fehlerhafte Distanzsensoren

Bei unserem zweiten Programm, bei dem der c't-Bot zum Licht fahren sollte, fuhr der reale c't-Bot ständig gegen Wände und andere Hindernisse. Der virtuelle c't-Bot in c't-Sim jedoch nicht. Nachdem wir eine Weile in unserem Code nach dem Fehler gesucht hatten überprüften wir schließlich die Sensoren und uns fiel auf, dass die Distanzsensoren ständig auf 999 standen. Bei einem anderen c't-Bot war es genau gleich.

Der Fehler war, dass wir immer nur die hex-Datei mit AVR-Studio auf den Bot übertragen haben, aber nicht die eep-Datei in der sich die Kalibrierungsdaten für die Sensoren befinden.

5.3 Ungleiche Lichtsensoren

Ein weiteres Problem bestand in den ungleichen Lichtsensoren. Während der linke Sensor einen Wert von 30-50 anzeige, war der recht Sensor bereits bei Werten über 300, und das in einem ausgeglichenen Lichtverhältnis. Eine erste Suche um diese Ungleichheit anzupassen in der `sensor_correction.h` blieb erfolglos, da dort nur die Distanzsensoren angepasst werden können. Weitere Nachforschungen eine äquivalente Möglichkeit, die Sensoren allgemein anzupassen führte zu keinen Ergebnissen.

Als Lösung wäre es denkbar, diese unterschiedlichen Werte direkt im behaviour-Code zu berücksichtigen.

5.4 Roboter in Reparatur

Als wir an einem Tag ins Labor kamen reagierte unser c't-Bot unerwarteter weise nicht mehr. Der Fehler war schnell gefunden: Ein Kabel des Batterie-Fachs hatte sich gelöst und dadurch bekam der Roboter keinen Strom mehr. Wir, als reines Software-Team ;-) , haben diesen Fehler durch anlöten des Kabels selbst behoben.

5.5 Roboter in Reparatur II

Am 22.07.08 mussten wir feststellen, dass am Roboter Nr. 8 das Behältnis für die Batterien fehlte. Nach Anlöten des Batteriefachs, das neben dem Roboter lag, funktionierte der c't-Bot jedoch immer noch nicht. Durch Messen fanden wir heraus, dass das Fach defekt war. Zwischen einer Feder und

dem äußeren Anschluss floss kein Strom. Erst durch längeres ausprobieren und durch Einsatz von viel Lötzinn lies es sich reparieren.

5.6 Roboter in Reparatur III

Das Display des Roboter Nr. 1 schwankte oft in der Stärke der Darstellung. Oft konnte man nur schwer entziffern was auf dem Display stand. Auch die Leistung der Motoren war stellenweise sehr gering. Erneut lag der Fehler an einem fehlerhaften Batteriefach, der sich durch Löten beheben lies. Es scheint so, als sei das Batteriefach eine sehr fehleranfällige Komponente im c‘t-Bot. Zumindest lässt sich bei der Stromversorgung der Fehler oft leicht lokalisieren.

5.7 Roboter in Reparatur IV

Während des Projektes bekamen wir noch einen weiteren Bot, den Roboter Nr. 20. Dieser musste von uns noch zu Ende gebaut und verschraubt werden. Schon nach kurzer Zeit bemerkten wir, dass einige Sensoren defekt waren und das Transportfach nicht richtig funktionierte. Auch das Flashen der Software war plötzlich nicht mehr möglich. Wir gaben eine Liste der Mängel weiter. Gegen Ende des Projekts waren wir jedoch dazu gezwungen, den Roboter wieder halbwegs lauffähig zu machen, dazu haben wir ihn nochmals teilweise auseinander genommenen und einen Fehler entdeckt, durch dessen Behebung wir den Bot wieder flashen konnten. Dies brachte uns jedoch nicht den gewünschten Bot für das WLAN-Team, da das WLAN-Modul fehlerhaft scheint (die Reaktionszeiten sind extrem hoch) und der linke Distanzsensor nicht funktioniert.

5.8 Speicherkartenprobleme

Als wir uns daran machten, den Algorithmus der einen zurückgelegten Weg aufzeichnen soll, zu implementieren, wurden wir darauf aufmerksam, dass die Speicherkarten Initialisierung bei keinem Bot funktioniert. Getestet haben wir es an Bot 1, 8 und 9. Bei jedem Bot gab es einen `init`-Fehler beim Versuch, `mmc_enable()` aufzurufen.

Durch mühsames und zeitaufwändiges Ausprobieren gelang es schließlich, zumindest bei Bot 1 eine Initialisierung der MMC Speicherkarte erfolgreich durchzuführen. Wie wir herausfanden muss man den `MMC_TIMEOUT` Wert in der `mmc_low.h` header-Datei im include-Verzeichnis von 500 auf 5000 erhöhen.

5.9 Follow Line: Gefährliche 90° Rechtskurven

Beim Programmieren der `Follow Line`-Funktion trat ein ungewöhnlicher Fehler auf: Da der Bot der Linie folgt, indem er mit dem rechten Liniensensor über ihr blieb, kann eine Abweichung von der Linie nur nach rechts zuverlässig erkannt werden. Weicht der Bot nach rechts von der Linie ab wird er prüfen, ob der linke Liniensensor eine Linie erkennt. Dann ist klar, der Bot muss links gegensteuern um zur Linie zurückzukehren.

Solch ein Komfort steht für eine Abweichung nach links leider nicht zur Verfügung. Bleibt ein ansprechen des linken Liniensensors aus, muss vermutet werden, der Bot ist links von der Linie abgekommen und muss rechts gegenlenken. Fährt der Bot nun auf eine scharfe 90° Rechtskurve zu, nimmt er folgerichtig an, er ist zu weit nach links abgekommen. Dummerweise ist der Bot bereits ein gutes Stück über die Kurve hinaus, bis er dies feststellt, was ihn dazu veranlasst sich beständig im Kreis zu drehen auf der Suche nach der verlorenen Linie.

5.10 Die Stromversorgung

Auch dies soll im Rahmen dieser Arbeit nicht unerwähnt bleiben, da es doch an fast jedem Tag unsere Aufgabe war, die Akkus zu laden. Anfangs öfters als zum Ende, da zwischenzeitlich einige neue Akkus hinzugekommen sind. Leider braucht der c't Bot relativ viel Energie, so dass die Akkus je nach Belastung mit vollen Akkus nur 20-90 Minuten gehalten haben, bevor zumindest einzelne (schwächere) Akkus ausgetauscht und wieder aufgeladen werden mussten.

5.11 Übersicht herausgefundener Defekte

Bot Nr.	Fehlerbeschreibung
1	Klappe defekt, Rechter Lichtsensor fehlerhaft
8	Speicherkartenslot defekt, rechter Liniensensor fehlerhaft, ungleiche Lichtsensoren
9	Klappe defekt , Speicherkartenslot defekt
20	WiPort reagiert deutlich langsamer als bei allen anderen Bots

Diese Auflistung deckt jedoch sicher nicht jeden Fehler ab, der aufgetreten ist. Dies liegt daran, dass wir recht spät damit begonnen haben die Fehler aufzuschreiben. Würden wir das Projekt nochmal machen würden wir dies früher tun um immer einen genauen Überblick zu haben.

5.12 Initialisierungszeit eines Ad-Hoc-WLANs

Ein Fehler, den wir uns einige Tage nicht erklären konnten, war dass die Initialisierung des WiPorts eine unterschiedliche Zeit dauert. Alle Daten die während dieser Zeit an das Modul gesendet werden gehen verloren. Aus diesem Grund ist diese Zeit sehr wichtig, da diese immer nach dem Einschalten des Roboters (inkl. aktiviertem WiPort) abgewartet werden sollte, bevor Daten per WLAN versendet werde.

Während dieser Zeit waren wir öfters mit unseren Notebooks auf dem Webserver des WiPorts um Einstellungen zu kontrollieren oder anzupassen, d.h. wir haben eine Ad-Hoc Verbindung zu den Bots aufgebaut. Diese Verbindung haben wir natürlich auch während des Ein- und Ausschalten des WiPorts nicht beendet, d.h. es existierte während dieser Zeit ein Ad-Hoc WLAN mit derselben SSID wie die welche wir bei allen WiPorts eingestellt haben.

Wenn man nun keine solche Verbindung hat, dauert die Initialisierung des WiPorts deutlich länger. Wir vermuten das dies daran liegt, dass der WiPort dann der erste Teilnehmer in diesem Netzwerk ist und das Netzwerk selbst aufbauen muss oder sich sonst nur einem bestehenden anschließt.

Warum dies so ist, ist für uns nicht von Bedeutung. Lediglich das herausfinden dieser Tatsache erforderte einige Versuche und kostete daher Zeit. Da uns Herr Hellmann zwischenzeitlich einen WLAN-Router zur Verfügung gestellt hat, nutzten wir diesen um ein Ad-Hoc WLAN mit der SSID der Roboter aufzubauen, so ging die Initialisierung des WiPorts deutlich schneller und vor allem dauert es immer gleich lange, wodurch wir die Zeit mit einer Warteschleife (`wlan_init()`, Ausgabe der Meldung „Initializing Wlan“) überbrücken konnten.

6 Projektabschluss

6.1 Resümee

Im Nachhinein können wir auf ein interessantes Projekt mit vielen Herausforderungen und Problemen zurückblicken. Müssten wir nochmals ein solches Projekt durchführen stünde allerdings eines für uns fest: Wir hätten uns zuerst mit der Konstruktion des c't-Bots an sich befasst, denn leider haben uns viele große und kleine Probleme welche hardwarebedingt waren daran gehindert, uns um unsere eigentliche Aufgabe zu kümmern, welche rein softwareseitig war. Da uns die Erfahrung und das Wissen mit dem Umgang des c't-Bots auf Hardwareebene fehlte nahmen Fehlersuche bei der Hardware weit mehr Zeit ein, als wir vermutet hätten.

Wenn die Hardware allerdings mal nicht streikte (oder wir einfach auf c't-Sim auswichen ;-)) vermochte das Projekt doch zu motivieren. Zu sehen, wie der Bot eine gegebene Aufgabe erfolgreich erfüllte war mehr als befriedigend. Für das Lösen der Aufgaben war Kreativität und Hartnäckigkeit gefordert. Ebenso musste man selbstständig nachforschen oder schlicht ausprobieren, da eine ausführliche Dokumentation gänzlich fehlte. Die Vorlesung *Rechnernetze* erwies sich als nützlich für die Arbeiten am WLAN Protokoll. Letztlich waren Kommunikation und Soft-Skills noch vonnöten um die Arbeit in einer vier Mann starken Gruppe zu koordinieren und die Wogen gestresster Teamkameraden zu glätten.

Beim Bearbeiten der Aufgaben stellte sich heraus, dass auf den ersten Blick einfach erscheinende Aufgaben auch ihre Tücken besitzen und sich als komplexer herausstellen können als zuerst gedacht. Dies war bei mehreren Aufgaben der Fall. Man sollte bei einer Einschätzung der Aufgaben also vorsichtig sein. Zumal es auch oft viele äußere Einflüsse gibt, welche die Arbeitsweise des Roboters beeinträchtigen und die man natürlich möglichst berücksichtigen sollte. Beispielsweise unterschiedliche Böden, welche die Fortbewegungsgeschwindigkeit beeinflussen oder Umgebungslicht, das die Lichtsensoren verwirrt.

Abschließend können wir noch sagen, dass wir aufgrund der gewonnenen Erkenntnisse über die Technik von Robotern und über Künstliche Intelligenz sagen können, dass keiner von uns noch Angst davor hat dass diese unselbstständigen, fehleranfälligen kleinen Gesellen die Weltherrschaft übernehmen könnten, wie dies beispielsweiße in Filmen wie *Terminator* oder *Die Matrix* dargestellt wird.

7 Literaturverzeichnis

1. Installationsanleitung für den c't-Bot und den c't-Sim. [Online] http://www.heise.de/ct/projekte/machmit/ctbot/wiki/Installationsanleitung.

2. Eclipse C/C++ Development Tooling. [Online] http://www.eclipse.org/cdt/.

3. OpenSVN. [Online] http://opensvn.csie.org.

4. Trac - Integrated SCM & Project Management. [Online] http://trac.edgewall.org.

5. Subclipse. [Online] http://subclipse.tigris.org.

6. Extreme Programming: A gentle introduction. [Online] 17. February 2006. http://www.extremeprogramming.org/.

7. **Benjamin Benz, Peter König.** Virtuelle Spielgefährten. [Online] 2006. http://www.heise.de/ct/06/03/186/.

8. **Benjamin Benz, Peter König, Lasse Schwarten.** Drängelnde Spielgefährten. [Online] 2006. http://www.heise.de/ct/06/05/224/.

9. **Grimmer, Christoph.** Hohe Schule. [Online] 2006. http://www.heise.de/ct/06/07/218/.

10. ct-Bot.de Forum -Wlan: Bot 2 Bot Kommunikation. [Online] 4 2008. http://www.ctbot.de/forum/wlan-bot-2-bot-kommunikation-t692.html.

11. heise.de - Funkmodul FAQ. [Online] http://www.heise.de/ct/projekte/ct-bot/faq/w.shtml#w03.

12. Firefighter Wiki. [Online] Uni Paderborn, 2007. http://firefighters.cs.upb.de.