Mobile Roboter SS12
Gruppe x

Kevin Walter, Gerhard Klostermeier, Andreas Jansche

Inhaltsverzeichnis

1 Einleitung

1.1 Inhalte und Ziele

Im Rahmen der Vorlesung "Mobile Roboter" der Hochschule Aalen wurden die Aufgaben aus dem Vorlesungsskript bearbeitet. Genutzt wurde Eclipse zum Entwickeln der Programme sowie avrdude zum Flashen des Microcontrollers des ct-Bot.

1.2 Der ct-Bot

...

2 Installation und Inbetriebnahme

2.1 Installation

Für die Entwicklung wurde ein aktuelles Ubuntu Linux verwendet. In diesem Abschnitt ist beschrieben was für vorbereitende Schritte auf dem System durcheführt werden müssen um entwickeln und den ct-Bot flashen zu können.

2.1.1 System vorbereiten

Zunächst müssen einige Softwarepakete nachinstalliert werden:

- eclipse-cdt
  Die Eclipse-Variante zur C/C++-Entwicklung. (Der ct-Bot wird in C programmiert.)
- binutils-avr, gcc-avr, avr-libc
  Werden zum Kompilieren für den Microcontrollers des ct-Bot benötigt. (Cross-Compiler)
- avrdude
  Wird zum Flashen des Microcontrollers des ct-Bot benötigt.
- subversion
  Wird zum Holen des aktuellen Quellcodes für den ct-Bot aus dem heise-Repository benötigt.

Der konkrete Befehl um die Pakete unter Ubuntu zu installieren sieht folgendermaßen aus:

```
sudo apt-get install eclipse-cdt binutils-avr gcc-avr \
avr-libc ^avrdude subversion
```

2.1.2 Quellcode holen

Für den Quellcode erstellen wir zunächst ein Verzeichnis *ctbot* und wechseln hinein:

```
mkdir ctbot && cd ctbot
```

Nun holen wir uns den aktuellen stable-Code vom heise-Repository:

```
svn checkout https://www.heise.de:444/svn/ctbot/stable
```

Der aktuelle Quellcode des ct-Bot befindet sich nun also unter *~/ctbot/stable* und muss im nächsten Schritt nur noch in Eclipse eingebunden werden.

2.1.3 Eclipse einrichten

Zunächst müssen wir den Quellcode in Eclipse einbinden:

- Dazu wählen wir zunächst im Menü *File* den Unterpunkt *Import*.
- Dort wählen wir *General -> Existing Projects into Workspace* und bestätigen mit *Next >*.
- Unter der Auswahl *Select root directory* geben wir entweder direkt das Quellcodeverzeichnis an (*~/ctbot/stable/ct-Bot*) oder wählen das Verzeichnis über *Browse...*.

Nun ist der Quellcode als Projekt in Eclipse eingebunden. Um für den ct-Bot zu kompilieren muss jedoch noch die Build-Configuration angepasst werden:

- Auf der linken Seite wählen wir zunächst das Projekt *ct-Bot* aus.
- Im Menü *Project* wählen wir *Properties* und dort *C/C++-Build*.
- Dort gehen wir auf *Manage Configurations...* und wählen die Konfiguration *Debug-MCU-m32*. Mit dieser Konfiguration wird der zuvor installierte Cross-Compiler genutzt um eine hex-Datei zum Flashen des ct-Bot zu erstellen.
- Wir bestätigen die Auswahl der Konfiguration mit *Set active* und verlassen die Einstellungen mit *Ok*, *Apply* und nochmals *Ok*.

Nachdem nun auch die passende Konfiguration gewählt wurde lässt sich das Projekt nun auch kompilieren. Jedoch kommt es zu einigen Warnmeldungen. Um diese Warnungen loszuwerden öffnen wir die Datei *ct-Bot.h* (*include/ct-Bot.h*) und suchen die Zeile

```
//#define SPEED_CONTROL_AVAILABLE
```

und entfernen die Kommentarzeichen //. Nun lässt sich das Projekt ohne Warnungen kompilieren und die eigentliche Entwicklung kann beginnen.

2.2 Inbetriebnahme

2.2.1 AVR ISP mkII und avrdude

Der *AVR ISP mkII* ist der Programmer, mit dem wir den ct-Bot flashen können. Als Tool dazu verwenden wir *avrdude*, der den *AVR ISP mkII* ansprechen kann.

Nach dem Einstecken des Programmers können wir mit dem Befehl *lsusb* prüfen, ob er korrekt vom System erkannt wird. Die Ausgabe sollte folgenden Text enthalten:

```
Atmel Corp. AVR ISP mkII
```

Um unsere hex-Datei nun auf den ct-Bot zu bekommen, müssen wir *avrdude* mit entsprechenden Parametern aufrufen:

- *-c avrispmkII* legt den *AVR ISP mkII* als Programmer fest.

- *-P usb* gibt USB als connection port an.
- *-p m32* legt m32 (ATmega32) als AVR device fest.
- *-U flash:w:<pfad>:i* legt die Aktion fest:
  - *flash* gibt an, dass geflasht werden soll.
  - *w* (write) gibt an, dass geschrieben werden soll. (Von der Datei in den Flash-Speicher, *r* (read) würde bedeuten vom Flash in die Datei.)
  - *<pfad>* gibt den Pfad zu der Datei an.
  - *i* gibt (optional) das Format der Datei an. (Hier *i* für *Intel Hex*.)

Zu beachten ist, dass *avrude* als root oder über *sudo* aufgerufen werden muss. Der konkrete Aufruf würde also folgendermaßen aussehen:

```
sudo avrdude -c avrispmkII -P usb -p m32 -U \
flash:w:"~/ctbot/stable/ct-Bot/Debug-MCU-m32/ct-Bot.hex":i
```

3 Aufgabenberichte

3.1 Motte / Kakalake

3.1.1 Ansatz 1: Verhaltensframework

Dieser Ansatz versucht die Aufgabe zu lösen in dem das Verhaltensframework genutzt wird, dass mit dem ct-Bot Reopsitory ausgelierfert wird. Die Nutzung des Frameworks hat einige Vorteile:

- Eigener Code leich einzusortieren und stört kein vorhandenen Code.
- Eigene Verhalten können simpel auf andere Verhalten zugreifen.
- Durch die Priorisierung der können mehrere Verhalten zusammen Arbeiten. (z.B. Bot soll eine Distnaz von 30cm fahren. Wenn er dabei aber an eine Tischkannte kommt wird abgebrochen damit der Bot nicht vom Tisch fällt.)
- Der ausgelieferte Verhlatenssatz bietet eine vielzahl von nützlichen Verhalten.

Nachteil ist leider die etwas komplexe Einarbeitung, da der Einstieg nicht sehr gut dokumentiert ist.

In dieser Lösung wurde die Aufgabe einer Lichtquelle zu folgen (Motte), in drei Unteraufgabenunterteilt:Linksdrehen, Rechtsdrehen, geradeaus Fahren. Welche Funktion ausgeführt wird, entscheidet ein Vergelich der beiden Fotowiderstände (LDR - *Light Dependent Resistor*). In diesen Vergleich fliesen zwei Konstanten ein: *LDR_CORRECT* und *TOLERANCE*. Mit *LDR_CORRECT* lassen sich (Hardwarebedingte) Unterschiede zwischen den beiden Sensoren ausgleichen. Liefert der linke Widerstand Beispielsweise bei gleichmäßiger Bestrahlung immer einen Wert der um 20 höher ist wie der des Rechten, so kann mit *LDR_CORRECT=20* dieser Fehler ausgeglichen werden. Die zweite Konstante ist für das geradeaus Fahren wichtig. Logisch wäre die Lichtquelle direkt vor dem Bot, wenn beide Sensoren den gleichen Wert liefern. In der Praxis wird das aber selten der Fall sein. Deswegen kann mit *TOLERANCE* angegeben werden, wie viel mehr Licht auf den linken bzw. den rechten Widerstand strahlen darf, ohne das es als links bzw. rechts Fahren interpretiert wird. Bei einerm *TOLERANCE* Wert von 15 wird Beispielsweise trozdem geradeaus gefahren, wenn der linke Sensor einen um 10 höheren Wert wie der rechte aufweist. Jetzt kann der Roboter grade auf die Lichtquelle zu fahren, selbst wenn diese nicht zu 100% vor ihm liegt.

Das Verhalten der Kakalake (vor einer Lichtquelle fliehen) ist exakt gleich dem der Motte implementiert, nur das rückwärts gefahren wird. So wird die Lichtquelle immer von der Roboter Vorderseite fixiert und dann davon weggefahren.

Die modifizerte Version der Motte, die sich den gefahrenen Weg merken und diesen wieder zurückfahren können soll, wurde über vorhandene Verhalten realisiert. Dazu

wurde zuerst das Verhalten *BEHAVIOUR_DRIVE_STACK_AVAILABLE* zugeschalten (in *ct-Bot/include/bot-logic/available_behaviours.h* und anschließend auf aktiv gesetzt (*bot_save_waypos_behaviour* in *ct-Bot/bot-logic/bot-logic.c*). Dieses Verhalten zeichnet im Folgenden alle Informationen auf, die für das wiederfinden der relevanten Positonen nötig sind. Nach einer über die Konstante *MAX_WAYPOINTS* festgelegten Anzahl Wegpunkten, wird das drive_stack() Verhlaten aufgerufen das die Punkte wieder anfährt. Wenn der Roboter an der Ausgangsposition angekommen ist, beginnt er wieder von vorne mit der Wegaufzeichnung und der Lichtquellensuche.

3.1.2 Ansatz 2: by Kevin

...

3.1.3 Ansatz 3: by Andy

...

4 Zeitplan