Benjamin Benz

c't-Bot: Programmieren leicht gemacht

Drehen, drehen, ein Stückchen vorwärts, noch ein letzter Schwenk nach links und dann mit Vollgas ab ins Ziel. So in etwa könnte es sich anhören, wenn Sie dem kleinen c't-Bot beibringen, wie man ein Labyrinth meistert. Das klingt zu schön, um wahr zu sein? Ist es aber! Auf den folgenden Seiten zeigen wir, dass man weder studierter Programmierer oder gar Assembler-Künstler noch Elektroingenieur sein muss, um eigene Roboter zu dirigieren. Der Einstieg ist nicht nur völlig kostenlos, sondern klappt auch ohne Hardware und Lötkolben, weil zum c't-Bot auch ein Simulator – der c't-Sim – gehört. Keine Sorge, dem Spaß tut der Simulator kaum Abbruch. Vielmehr kann man sorglos drauflosexperimentieren, denn wenn der virtuelle Bot mal vom simulierten Tisch fällt, geht rein gar nichts kaputt. Für die ersten eigenen Robotikschritte brauchen Sie nicht mehr als einen PC (mit Windows, Linux oder Mac OS) und im Idealfall einen Internet-Zugang. Am einfachsten klappt es unter Windows, denn auf der Heft-DVD haben wir im Verzeichnis „ct-Bot" ein Archiv mit allen dafür benötigten Tools vorbereitet. Sie müssen es lediglich auf Ihrer Festplatte oder einem anderen beschreibbaren Medium entpacken. Linuxer und Macies finden im Projekt-Wiki eine ausführliche Installationsanleitung.

Auf die Plätze ...

Der c't-Sim präsentiert den virtuellen Robotern eine dreidimensionale Welt. Da er in Java geschrieben ist, brauchen Sie eine Java-Laufzeitumgebung (JRE) oder gleich das ganze Java Development Kit (JDK) sowie die 3D-Erweiterung Java3D. Unter Windows müssen Sie normalerweise gar nichts installieren, es reicht ein Doppelklick auf die Batch-Datei „ct-Sim.bat" aus dem frisch extrahierten c't-Bot-Verzeichnis.

Dann öffnet sich das Hauptfenster des c't-Sim, das auf der rechten Seite ein dreidimensionales La-

Roboter-Programmieren mit dem c't-Bot-Simulator

Keine teure Hardware, keine am Lötkolben verbrannten Finger, kein Assembler-Code – mit dem Simulator gelingt der Einstieg in das c't-Bot-Projekt schnell, leicht und vor allem völlig kostenlos.

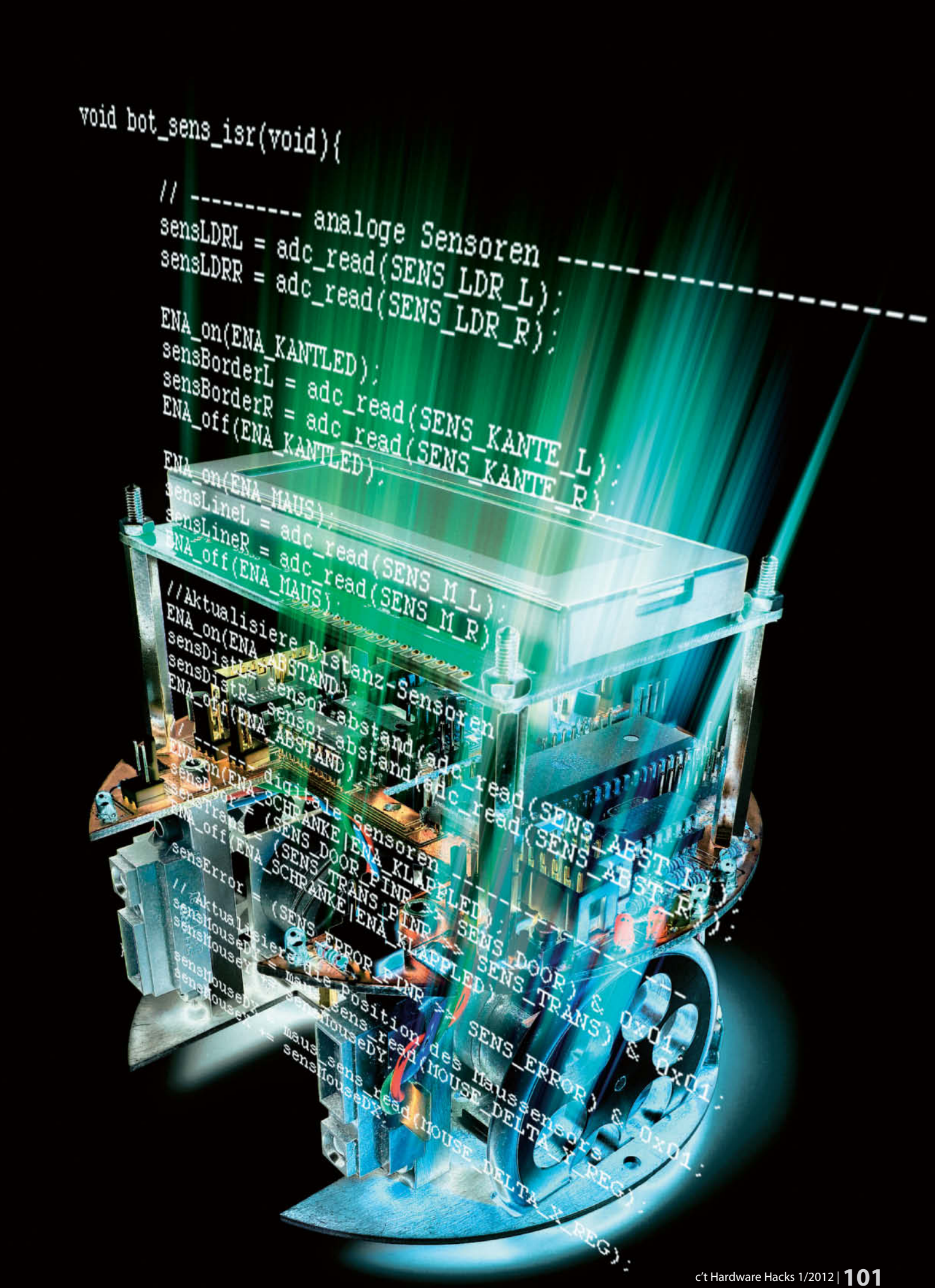
Kurzinfo

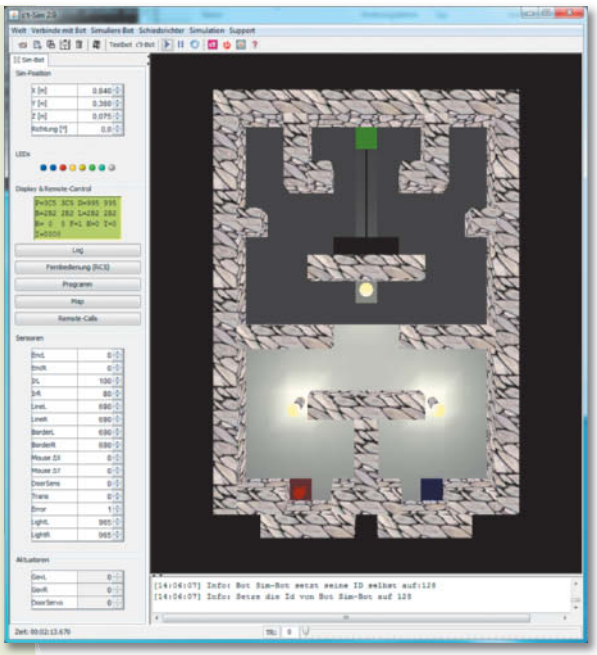
Zeitaufwand:
wenige Minuten für erste Experimente; für komplexe Roboterprogramme locker ein Wochenende

Kosten:
Alle Tools sind OpenSource und damit kostenlos.

Programmierkenntnisse:
Grundkenntnisse in C sollten Sie mitbringen.

Elektronik:
nicht erforderlich





Der Roboter-Code bekommt nicht mit, ob er auf dem Mikrocontroller eines realen Bots läuft oder innerhalb des c't-Sims in einer virtuellen Welt fährt.

byrinth zeigt. Darunter stehen Log-Meldungen, die unter anderem bestätigen, dass bereits ein erster simulierter Bot das Testlabyrinth betreten hat. Er steht auf dem roten Startfeld, trägt ein rotes Mäntelchen und schaut in Richtung der ersten Lampe.

Fertig …

Alle relevanten Daten zu diesem „Sim-Bot" zeigt die rechte Spalte. Das beginnt ganz oben mit Position und Blickrichtung in der virtuellen Welt. Mit ein paar Mausklicks kann man den Bot in der Welt hin- und herschieben. Zur Orientierung: Die X-Achse verläuft von links nach rechts, die Y-Achse von unten nach oben. Die Z-Achse verstellt man besser nicht, sonst verliert der Bot die Bodenhaftung oder versinkt.

Unterhalb der Positionsdaten folgen virtuelle Varianten für LEDs und Display des realen Bots. Weil die Simulation noch nicht begonnen hat, sind diese ebenso leer wie die Anzeigen für die Sensoren und Aktuatoren des Roboters. Das ändert ein Klick auf das kleine „Play"-Symbol am oberen Fensterrand. Dass die Simulation läuft, zeigt die laufende Uhr ganz unten. Außerdem erscheinen nun im grünen „Display" erste Ausgaben. Die einzelnen Sensor-Felder füllt der c't-Sim normalerweise automatisch anhand der Umgebung des Roboters aus. Beispielsweise liefern Linien- (Line) und Abgrundsensoren (Border) den Wert 690, solange der Roboter auf dem roten Feld steht. Schiebt man ihn etwas nach vorn, ändert sich der Wert. Auch die Abstands- (IrL und IrR) und Lichtsensoren (LightL und LightR) reagieren, sobald beim Vorwärtsschieben die Lampe in ihren Erfassungsbereich kommt.

Los!

Bisher hat der Roboter(-code) selbst noch gar nichts gemacht, sondern wurde nur herumgeschubst und bekam Sensorwerte geliefert. Setzen Sie den Bot nun wieder zurück auf die Mitte des roten Startfelds, damit es losgehen kann: Ein Klick auf die Schaltfläche „Fernbedienung (RC5)" öffnet eine virtuelle Version der IR-Fernbedienung. Die Taste „5" haben wir mit einem kleinen Demo-Programm belegt, mit dem der Bot das Labyrinth bis zum grünen Zielfeld durchquert und sich dort vor Freude einmal im Kreis dreht.

Es mag zwar ganz nett sein, dem kleinen Kerl über die Schulter zu schauen, aber nun sollen seine Geister auch nach Ihrem Willen leben. Am einfachsten geht das über die sogenannten Remote-Calls, mit denen Sie vorgefertigte Routinen aus der Ferne starten können. Das zugehörige Eingabefenster versteckt sich wiederum hinter der Schaltfläche „Remote-Calls" und zeigt auf der linken Seite eine Liste aller verfügbaren Kommandos.

Wie sie funktionieren, wird schnell klar, wenn man in der Zeile bot_turn() zum Beispiel 90 Grad einstellt und dann den Remote-Call mit dem Pfeilsysmbol an den Bot schickt. Voilá, er vollführt eine Vierteldrehung im Uhrzeigersinn. bot_goto_dist() mit den Parametern „200" und „1" lässt ihn 20 Zentimeter nach vorn fahren. Ist der zweite Wert kleiner als null, fährt der Bot rückwärts. Diese beiden Basis-Befehle reichen bereits aus, um den Bot durch das Labyrinth zu lotsen.

Nicht alle Remote-Calls in der Liste eignen sich für den direkten Aufruf oder funktionieren nur unter bestimmten Voraussetzungen. Etwas spielen kann man jedoch auch noch einmal mit bot_drive_square(), bot_solve_maze().

Hintergrund

Was der Bot während seiner Reise mit seinen beiden Abstandssensoren sieht, enthüllt ein Klick auf die Schaltfläche „Map". Dort markiert der Bot Hin-

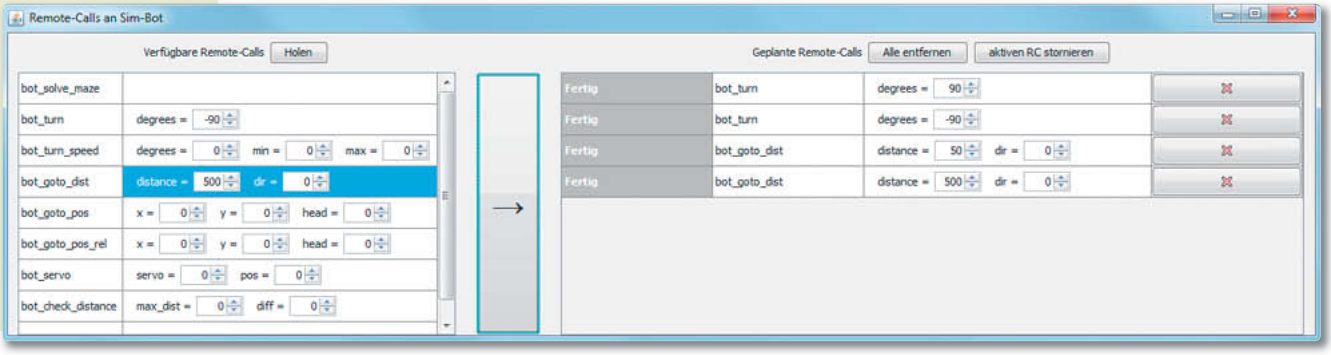
TIPP

Startet der c't-Sim nicht, fehlt Ihrem Windows vermutlich das 3D-API DirectX, das Microsoft zum Download anbietet.

TIPP

Die genaue Beschreibung aller Funktionen aus dem Code von c't-Bot und c't-Sim liegen im HTML-Format in den Unterverzeichnissen Documentation/api der jeweiligen Teilprojekte.

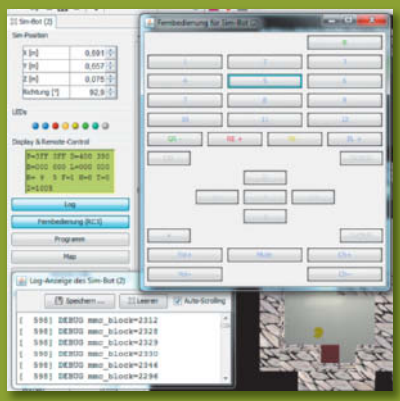
Die Remote-Calls ermöglichen den direkten Aufruf von Funktionen im Steuercode des Roboters.



Debug-Tipps

Sowohl der c't-Sim als auch das Display des realen Bot können Debug-Ausgaben darstellen. Im Code verwendet man dazu einfach die Funktion LOG_DEBUG(), die dieselbe Syntax verwendet wie printf(). Wo die Debug-Ausgaben landen und ob sie überhaupt mit kompiliert werden, legen #define-Zeilen in ct-Bot.h fest.

Das Display kennt unterschiedliche „Screens", die man mit der Taste TV/VCR der Reihe nach durchschalten können. Die globale Tastenbelegung der Fernbedienung erfolgt über die Datei rc5.c. Wer möchte, kann auch eigene Screens definieren und diese mit dem Befehl „register_screen()" am Framework anmelden. Der dabei übergebene Screenhandler kann nicht nur die Anzeige füllen, sondern auch auf RC5-Tasten-Codes reagieren. Deshalb können sich die Tastenbelegungen auch von Screen zu Screen unterscheiden.



Der grün unterlegte Bereich des c't-Sim zeigt exakt das an, was auch der reale Bot auf seinem Display präsentieren würde. Für ausführlichere Log-Meldungen gibt es einen eigenen Logging-Mechanismus.

Eclipse-Tipps

- Halten Sie die Strg-Taste gedrückt, so werden Datentypen, Variablen, Include-Anweisungen und Funktionsnamen zu Links, auf die Sie mit der Maus klicken können, um zu ihrer Definition zu springen.
- Strg+Tab wechselt zwischen Header- und Sourcecode-Datei hin und her.
- Strg+Shift+L blendet eine Liste der Shortcuts ein.
- Eclipse kann Funktionen und Schleifen einklappen und so für mehr Übersicht sorgen. Einstellungen dazu stehen unter Einstellungen/C/C++/Editor/Folding.
- Wo überall eine Funktion ein Makro oder eine Variable genutzt wird, verrät das per rechter Maustaste erreichbare Hilfsmittel References/Project. So findet man unter anderem schnell Beispiele, wie eine Funktion zu verwenden ist oder wer alles eine globale Variable verändert.
- Ebenfalls im Content-Menü versteckt sich Refactor/Rename, mit dem man Variablen oder Funktionen umbenennen kann. Dabei werden automatisch alle Referenzen mit geändert.
- Per „Team/Synchronize with Repository", „Compare With" oder „Replace With" findet man schnell alle Änderungen zu lokal oder im Repository gespeicherten (Vorgänger-) Versionen.
- Team/Update bringt den lokalen Code auf den neuesten Stand aus dem Repository, ohne dabei eigene Änderungen zu überschreiben.

dernisse dunkel und freie Flächen hell; unbekanntes Terrain bleibt grau. Je mehr der Bot seinen Beobachtungen für einen Punkt in der Karte traut, desto intensiver färbt er ihn ein. Dass die Karte nur teilweise ausgefüllt ist, liegt daran, dass nur der Simulator die gesamte Welt kennt, dem Roboter-Code davon aber immer nur das präsentiert, was auch ein realer Bot an der jeweiligen Stelle wahrnehmen würde.

Das Software-Framework des c't-Bot kümmert sich aber nicht nur um Sensorauswertung und Kartografie, sondern regelt auch die Motoren und sorgt so dafür, dass der Bot sauber geradeaus fährt. Eine Notfallroutine verhindert, dass er abstürzt, wenn er an eine Tischkante heranfährt. Das können Sie ausprobieren, wenn Sie den Bot bei pausierter Simulation vorsichtig an einen Abgrund (schwarzes Feld) heranschieben. Dabei dürfen nur die beiden vorderen Zacken knapp über die Kante hinausragen. Dort sitzen die „Border"-Sensoren und schauen nach unten. Sobald die Simulation wieder startet, fährt der Bot rückwärts in Sicherheit. Fährt der Bot doch einmal zu weit, so blockiert ihn der Simulator, sobald seine Räder über dem Abgrund sind. Der Roboter-Code bekommt davon nichts mit, der Anwender erkennt es an einem Farbwechsel.

Das Software-Framework des c't-Bot erfüllt sehr unterschiedliche Anforderungen:

- In allererster Linie soll die Programmierung eines Roboters möglichst einfach sein. Sprich: Der Einsteiger bleibt von Assembler-Code, PID-Reglern, Interrupts, I/O-Ports, Timern und vielen weiteren Interna verschont – zumindest zu Beginn.
- Alle Sensorwerte stehen als globale Variablen zur Verfügung. Eine handvoll Funktionen und Variablen gewährt Zugriff auf die Aktuatoren.
- Der Code ist plattformunabhängig, sodass er sowohl im Simulator (auf dem PC) als auch dem realen Bot läuft, ohne dass man ihn anpassen muss.
- Einmal entwickelte Unterprogramme von ganz rudimentären wie `bot_turn()` bis zu komplexen wie `bot_solve_maze()` lassen sich weiterverwenden, mit Freunden austauschen und in andere Routinen einbauen.

TIPP

Verfolgen Sie, wie sich die Sensorwerte während der Fahrt in Abhängigkeit von der Umgebung verändern.

- Verschiedene Aufgaben – wie etwa die erwähnten Notfallmechanismen – können im Hintergrund oder parallel ablaufen.
- Der Code lässt sich über ein paar Konfigurationsoptionen so weit abspecken, dass er auch auf kleine Mikro-Controller passt.

Die Kehrseite der Medaille ist allerdings, dass ein paar Konventionen gelten, die dem gestandenen Mikrocontroller-Programmierer befremdlich vorkommen könnten. Auch stehen die für den Einstieg relevanten Programmteile nicht etwa in main.c, wie man bei einem einfachen C-Programm vermuten würde, obwohl der gesamte Bot-Code in Standard-C geschrieben ist. Im Laufe des c't-Bot-Projektes gab es immer mal wieder Kritik, das Framework sei mit seinen über 250 C- und Header-Dateien, zahlreichen Hilfsprogrammen, Konfigurationsdateien, Makefiles und so weiter viel zu komplex. Wir sind anderer Meinung und wollen anhand von ein paar kleinen Beispielen zeigen, wie sehr eine gut ausgebaute Infrastruktur die Entwicklung eigener Roboterprogramme erleichtert – wer Auto fahren will, fängt ja auch nicht jedes Mal mit dem Fundament seiner Straße an.

Werkzeugkasten

Bevor es aber an die Programmierung geht, noch ein paar Worte zu den benötigten Werkzeugen: Das ganze Projekt setzt komplett auf Open Source – nicht nur beim Code, sondern auch bei den Tools. Im Hintergrund übersetzen verschiedene gcc-Derivate, während im Vordergrund Eclipse als Entwicklungsplattform dient.

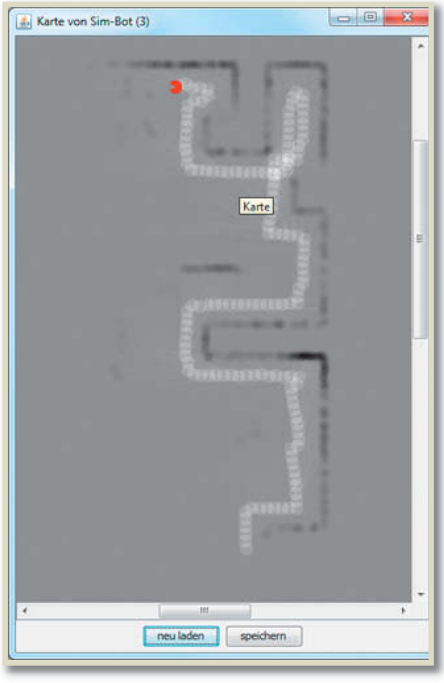
Für Windows haben wir die aktuelle Eclipse-Version mitsamt allen benötigten Plugins sowie den gcc-Toolchains für x86- und Atmel-Prozessoren auf die Heft-DVD gepackt. Es reicht ein Doppelklick auf „eclipse.bat" – sofern Sie das Verzeichnis „ct-Bot" zuvor auf ein wiederbeschreibbares Laufwerk extrahiert haben. Linuxer halten sich am besten an die Eclipse- und gcc-Versionen ihrer Distribution. Eine ausführliche Installationsanleitung für Linux und Mac OS X finden Sie im c't-Bot-Wiki.

Sobald Eclipse läuft, schließen Sie den Begrüßungsdialog und lassen den Blick über den Bildschirm schweifen: Das große Fenster in der Mitte ist der eigentliche Editor. Links daneben blendet der Project-Explorer die zum jeweiligen Projekt gehörenden Dateien ein. Unter dem Editor-Fenster informiert Eclipse über etwaige Probleme, den Fortschritt des Build-Vorgangs (Console) oder Suchergebnisse. Das rechte Fenster gewährt Schnellzugriff auf die Definitionen innerhalb des aktiven Editorfensters. Keine Panik, wenn Eclipse plötzlich ganz anders aussieht. Die Entwicklungsumgebung kennt sogenannte „Perspectives", also voreingestellte Fensteraufteilungen. Diese können Sie rechts oben umschalten. Für das c't-Bot-Projekt ist die Ansicht „C/C++" ideal. Hat man eine Perspektive kaputt gespielt, kann man sie über den Menüpunkt „Window/Reset Perspective" rekonstruieren.

Eclipse ist ein ungeheuer mächtiges Werkzeug, dass viele Programmiersprachen unterstützt und dem Entwickler viel Arbeit abnehmen kann, aber auch ein wenig Eingewöhnung erfordert. Ein paar Eclipse-Tricks und die wichtigsten Tastenkürzel haben wir im Kasten und im Wiki zusammengetragen.

TIPP

Eclipse kompiliert den Code automatisch jedes Mal, wenn Sie eine Datei speichern (Strg+s). Den Fortschritt zeigt die „Console". Das hat den großen Vorteil, dass man Fehler schnell bemerkt und unmittelbar im „Problems"-Fenster sieht.



Je länger der Roboter herumfährt, desto genauer lernt er die Umgebung kennen. So markiert er ursprünglich unbekanntes Terrain (grau) als befahrbar (weiß) oder durch Hindernisse blockiert (schwarz).

Abholmarkt

Den Quelltext für c't-Bot und c't-Sim haben wir bereits für Sie aus dem SVN-Repository geholt. Im Weiteren geht es um das Teilprojekt „ct-Bot". Mit ein paar Mausklicks können Sie es selbst übersetzen: Klicken sie es dazu im „Project Explorer" (links) an und dann im Menü auf „Project/Clean" und schon legt der Compiler los und erstellt nach einiger Zeit im Unterverzeichnis Debug-W32 eine neues ct-Bot.exe. Das testen Sie am besten gleich einmal im c't-Sim (Simulator-Schaltfläche „c't-Bot"). Dafür eignen sich die oben beschriebenen Schritte, denn Ihr Binary sollte in etwa so reagieren wie das von uns mitgelieferte, das der c't-Sim automatisch startet.

Kompiliert der Code aufgrund eigener Veränderungen nicht mehr kann man das Projekt jederzeit per Rechtsklick und „Team/Update" wieder vom SVN-Server herunterladen. Einen detaillierten Vergleich der Änderungen erlaubt indes die Option „Compare with".

Selbstbau

Nun steht das erste eigene Verhalten an. Verhalten – im Code „behaviours" genannt – sind kleine Anwendungsprogramme für den c't-Bot, die das Framework ausführt. Sie sind immer nach dem gleichen Schema aufgebaut und daher flexibel miteinander kombinierbar – doch dazu später mehr. Für den Anfang enthält das Projekt bereits einen Prototypen namens behaviour_simple.c im Unterverzeichnis bot-logic.

Ignoriert man im Listing rechts erst einmal das Vorgeplänkel und den Übergabeparameter, wird schnell klar, dass die eigentliche Verhaltensfunktion

im Wesentlichen alte Bekannte wie bot_turn() und bot_goto_dist() – die Sie zuvor schon als Remote-Calls getestet haben – hintereinander reiht. Einziger Clou dabei: Das Verhalten speichert in der Variable simple_state den aktuellen Zustand. Diesen schaltet es nach jedem abgesetzten Kommando weiter und beendet sich dann sofort. Das Framework packt den Funktionsaufruf in seinen Stack, arbeitet ihn ab und stellt sicher, dass bot_simple_behaviour() danach erneut ausgeführt wird. Im ersten Durchgang lässt der Beispiel-Code den Roboter folglich ein paar Zentimeter vorwärts fahren. Im zweiten dreht er sich um 90 Grad und dann geht es von vorne los. Sprich der Roboter fährt ein Quadrat ab.

Wer die Datei behaviour-simple.c in Eclipse öffnet und genau hinschaut, erkennt an der grauen Hinterlegung, dass das gesamte Beispielverhalten (per #ifdef-Block) noch deaktiviert ist. Folglich müssen Sie es noch freischalten. Entfernen Sie dazu die beiden Kommentarzeichen vor der Zeile:

```
#define BEHAVIOUR_SIMPLE_AVAILABLE
```

in der Datei include/bot-logic/available_behaviours.h. Nun sollte der Code in behaviour_simple.c nicht mehr ausgegraut sein. Wenn alle Dateien gespeichert sind, übersetzt Eclipse den Code automatisch und Sie können das neue Binary ausprobieren. Hat alles geklappt, fängt der Roboter sofort an, im Quadrat zu fahren, weil wir den Aufruf für bot_simple_behaviour() an anderer Stelle bereits vorbereitet haben, doch dazu später mehr.

Achtung: Der Roboter kollidiert dabei unter Umständen mit Wänden, schieben Sie ihn daher an eine freie Stelle in der virtuellen Welt.

Verfeinerung

Ohne Sensorauswertung, völlig blind in der Gegend herumzufahren ist aber nicht gerade die hohe Schule der Robotik. Im zweiten Beispiel soll der Roboter zum Licht fahren, aber cleverer als eine Motte kurz vorher anhalten. Lichtquellen oder genauer die Intensität von Licht kann der Bot über zwei nach vorne gerichtete Sensoren erkennen. Die beiden Distanzsensoren warnen, wenn er dabei einer Wand zu Nahe kommt. In erster Näherung lautet der Lichtsuch-Algorithmus nur:

1. Drehe den Bot in 10°-Schritten und speichere dabei die Richtung mit maximaler Lichtintensität.
2. Fahre so lange in diese Richtung, bis die Distanzsensoren ein Hindernis näher als 12 Zentimeter melden.

Die Umsetzung als Verhalten ist wiederum sehr einfach: Die beiden Stufen des Algorithmus werden jeweils zu einem Zustand. Dazu kommen noch zwei weitere zum Initialisieren und Aufräumen von Variablen. Auch die Abfrage der Sensoren erfordert keinerlei Hexerei, wohl aber etwas Interpretation: So steht in der globalen Variablen sensLDRL zwar der Messwert des linken Helligkeit-Sensors, doch der wird bei völliger Dunkelheit maximal und geht bei starkem Lichteinfall gegen Null. Die Variablen sensDistL und sensDistR enthalten zwar aufbereitete Werte in Millimetern, aber die Sensoren haben einen beschränkten Erfassungsbereich: So liefern sie nur zwischen 10 und 69 Zentimetern Distanz zuverlässige Werte. Das fängt

Listing

```
void bot_simple_behaviour(Behaviour_t * data) {
  switch (simple_state) {
     case 0:
       bot_drive_distance(data, 0, BOT_SPEED_MAX, 14);
       simple_state = 1;
       break;
     case 1:
       bot_turn(data, 90);
       simple_state = 0;
       break;
     default:
       return_from_behaviour(data);
     break;
  }
}
```

Dieses kleine Anwendungsprogramm – alias behaviour – lässt den c't-Bot im Quadrat fahren.

Code-Archiv

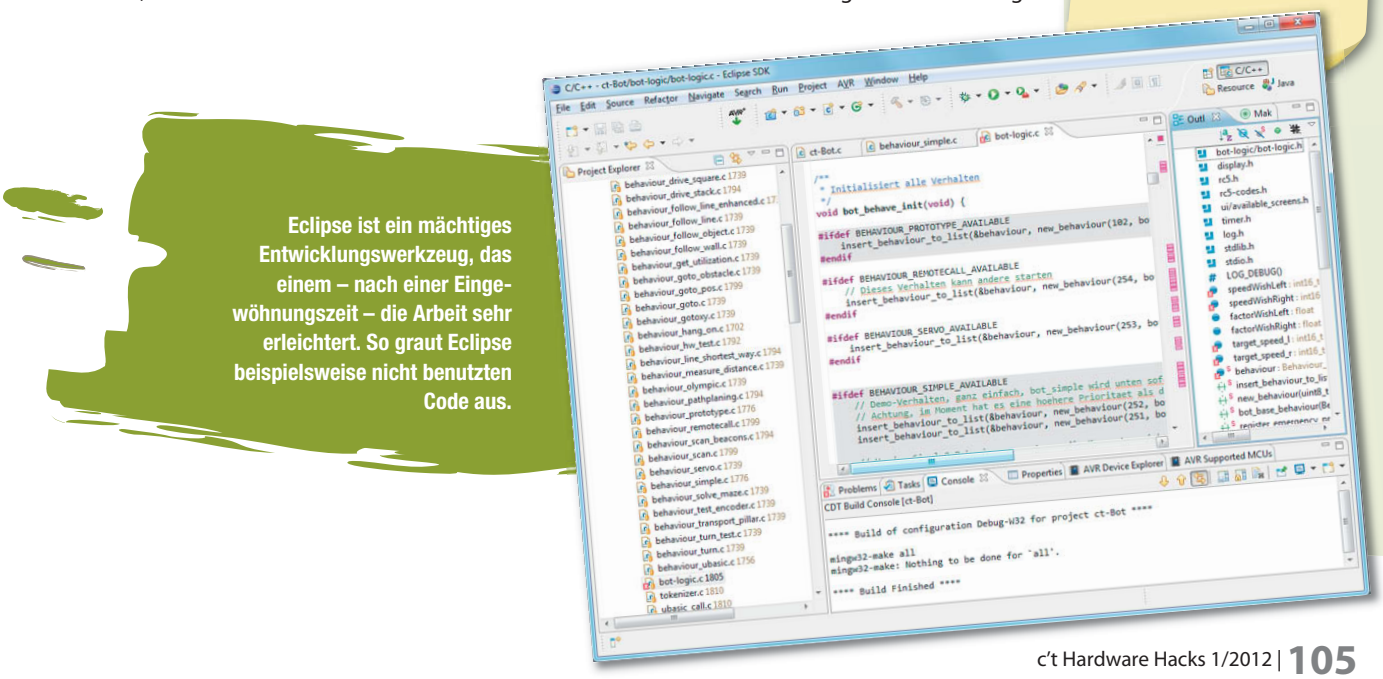
Der gesamte Quelltext der beiden Teilprojekte c't-Bot und c't-Sim ist über ein SVN-Repository zugänglich, auf das Eclipse direkt zugreifen kann. Somit können Sie jederzeit Ihren eigenen Code mit dem offiziellen Vergleichen oder Änderungen zwischen Versionen verfolgen.

Im Repository gibt es für jedes der beiden Teilprojekte zwei Zweige: Unter stable finden Sie immer eine gut getestete Version, der aber womöglich einige neue schicke Funktionen fehlen. Für den Einstieg in das Projekt sind Sie hier genau richtig. In den devel-Zweig fließen indes schon mal Routinen ein, die noch nicht so ganz funktionieren und an denen die Entwickler noch feilen.

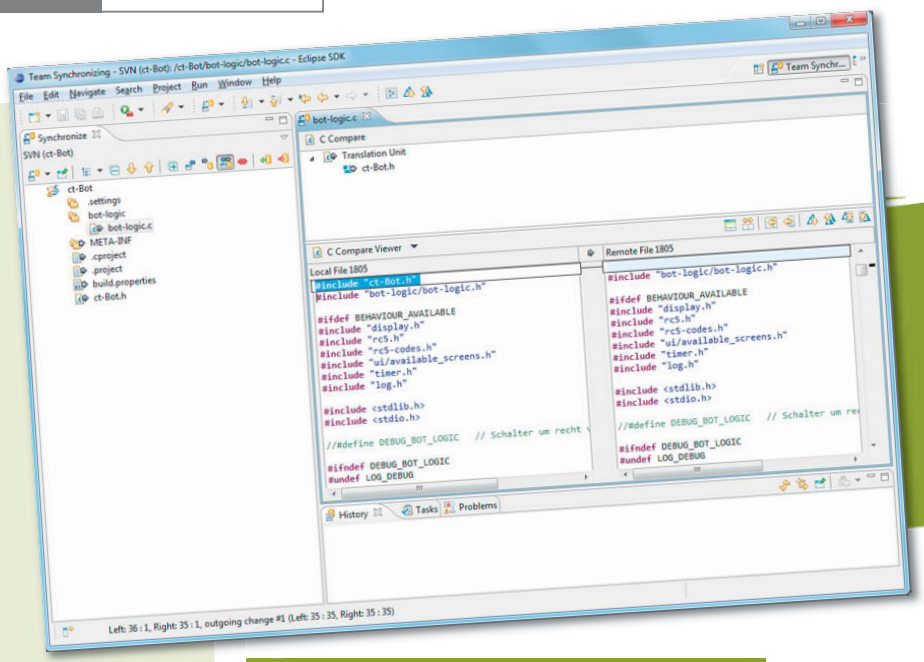
Im Entwicklungspaket auf der Heft-DVD ist übrigens der stable-Zweig voreingestellt.

TIPP

Die Power-Taste der Fernbedienung (rechts oben) bricht alle laufenden Aktionen ab und hält den Bot an.



Eclipse ist ein mächtiges Entwicklungswerkzeug, das einem – nach einer Eingewöhnungszeit – die Arbeit sehr erleichtert. So graut Eclipse beispielsweise nicht benutzten Code aus.



Unterschiede zwischen Code-Versionen zeigt Eclipse detailliert an, wenn man per Rechtsklick „Compare With“ oder „Team/Synchronize with Repository“ aufruft.

Profi-TIPP

Wem das Übersetzen des Codes zu lange dauert und mindestens einen Dual-Core-Prozessor besitzt, kann Eclipse auch mehrere Compiler-Instanzen parallel anstoßen lassen. Die Option dazu heißt „Use parallel Builds“ und versteckt sich unter Project/Properties/C/C++ Build/Behaviour.

TIPP

Wem die eingebauten Vergleichsfunktionen zu komplex sind, kann auch mehrere Kopien des Projektes aus dem Repository in den Project-Explorer laden und dann lokal per Hand vergleichen.

Listing

```
void bot_simple2_behaviour(Behaviour_t * data) {
  #define STATE_SIMPLE2_INIT 0
  ...
  static uint16_t max_dir, dir;
  uint16_t light=(sensLDRL+sensLDRR)/2;
  ...

  switch (simple2_state) {
     case STATE_SIMPLE2_INIT:
        max_light = INT16_MAX;
        max_dir=0;      dir=0;
        simple2_state=STATE_SIMPLE2_SEARCH;
        break;
     case STATE_SIMPLE2_SEARCH:   // drehen und Maximum suchen
        if (light < max_light){
           max_dir=dir;            max_light=light;
        }
        if (dir < 360) {    // Noch nicht ganz rum?
           bot_turn(data, 10);   // drehen
           dir+=10;        // neue Position sichern
        } else { if (simple2_light == 0)
           bot_turn(data, max_dir);    // zum Licht drehen
        else
           bot_turn(data, max_dir-180);  // vom Licht wegdrehen
           simple2_state=STATE_SIMPLE2_DRIVE;
        }
        break;
     case STATE_SIMPLE2_DRIVE:    // fahren, bis Hinderniss
        free = (sensDistL < sensDistR) ? sensDistL : sensDistR;
        free = (free > SENS_IR_MAX_DIST) ? SENS_IR_MAX_DIST : free;
        if (free > SENS_IR_SAFE_DIST) {
           bot_goto_dist(data,free -- SENS_IR_SAFE_DIST,1);
           // simple2_state = STATE_SIMPLE2_INIT;
        } else
           simple2_state=STATE_SIMPLE2_DONE;
        break;
     default:
        return_from_behaviour(data);
        break;
  }
}
```

der Code im Listing-Kasten und bedient sich dazu vordefinierter Konstanten wie SENS_IR_MAX_DIST. Apropos Sensoren: Die Variablen für die einzelnen Sensoren sind in sensor.h beschrieben. Allgemeine Definitionen zu den Abmessungen und Eigenschaften des Bots stehen in bot-local.h.

In der Praxis verfehlt der Bot mit dem im Kasten abgedruckten Code leider oft die Lichtquelle, weil die Messung des Winkels zu Beginn nicht genau genug ist. Eine hemdsärmlige Lösung dafür versteckt sich in der auskommentierten Zeile im dritten Case-Abschnitt. Sie setzt nach der Fahranweisung den Status des Verhaltens zurück und sorgt so dafür, dass der Bot regelmäßig neu auf die Lichtquelle einschwenkt.

Sie müssen diesen Code übrigens nicht abtippen, weil er bereits in der Routine bot_simple2_behaviour() steht. Allerdings startet diese nicht wie das erste Beispiel automatisch und kommt folglich erst einmal nicht zum Zuge. Um die Lichtsuche zu starten gibt es verschiedene Optionen: Mit der Power-Taste der virtuellen Fernbedienung können Sie das automatisch startende bot_simple() abbrechen und dann bot_simple2() als Remote-Call anwerfen. Dass der Remote-Call nur bot_simple2() und nicht etwa bot_simple2_behaviour() heißt ist kein Tippfehler sondern entspricht dem Namensschema für Verhalten, doch dazu gleich mehr.

Interna

Alternativ zum Aufruf per Remote-Call oder von einem anderen Verhalten aus, möchten wir zeigen, wie man das neue Verhalten automatisch startet und dabei noch ein wenig auf das Verhaltens-Framework eingehen. Dessen Herzstück ist die Datei bot-logic.c. Dort registriert die Funktion bot_behave_init() alle Verhalten in der Taskliste und kann sie auch gleich starten. Für die Beispielverhalten finden Sie dort folgende Einträge:

```
insert_behaviour_to_list(&behaviour, 7
        new_behaviour(252,bot_simple_behaviour, BEHAVIOUR_ACTIVE);
insert_behaviour_to_list(&behaviour, 7
      new_behaviour(251,bot_simple2_behaviour, BEHAVIOUR_INACTIVE);
//activateBehaviour(NULL, bot_simple2_behaviour);
```

insert_behaviour_to_list() fügt einen neuen Eintrag in die globale Verhaltensliste „behaviour“ ein. Die Funkti-

on zum Erzeugen der Einträge erwartet neben dem Namen der Verhaltensfunktion und dem Hinweis, ob das Verhalten sofort loslegen soll, auch dessen Priorität. Verhalten mit hoher Priorität können die Steuerungsanforderungen von solchen mit niedriger Priorität überschreiben. Daher sind die Werte oberhalb von 245 eigentlich für Notfallroutinen – wie die Abgrundvermeidung – reserviert. Dass die beiden simple-Verhalten diese aushebeln, ist eine nur für erste Gehversuche sinnvolle Ausnahme. Eigentlich gehören solche High-Level-Verhalten in den Bereich um 100.

Ein Verhalten, das zu Beginn nur in die Liste gehängt, aber nicht aktiviert wurde, kann man jederzeit per activateBehaviour() starten. Das reine Entfernen der Kommentarzeichen vor diesem Befehl reicht jedoch in diesem Fall nicht aus, weil das höher priorisierte bot_simple_behaviour() ewig läuft und niemand anderen mehr zum Zuge kommen lässt. Initialisieren Sie Letzteres doch einfach mit BEHAVIOUR_INACTIVE.

Außerhalb von bot_behave_init() ergibt der direkte Aufruf von activateBehaviour() nur in wenigen Ausnahmen Sinn. Viel eleganter klappt das über Botenfunktionen, die es für jedes Verhalten gibt. Einge Beispiele davon haben Sie bereits kennengelernt – etwa bot_turn() oder bot_goto_dist(). Ihnen gemeinsam ist, dass der Zusatz „_behaviour" fehlt. Dieser ist für die eigentlichen Verhalten reserviert, die aber nur vom Framework selbst aufgerufen werden und in eigenem Code – oder anderen Verhalten – nichts zu suchen haben.

Die Botenfunktionen versetzen die Verhalten in einen sinnvollen Startzustand, verarbeiten Übergabeparameter und aktivieren dann das Verhalten in der erwähnten Taskliste. Auch für die Simple-Verhalten gibt es solche Aufrufe, die im Wesentlichen nur den Zustand zurücksetzen und dann das Framework per switch_to_behaviour() zu einem eleganten Taskwechsel veranlassen. So ist sichergestellt, dass der Aufrufer nach Beendigung auch wieder an die Reihe kommt.

Übrigens: Ob das aufgerufene Verhalten erfolgreich war oder nicht, hinterlegt das Framework im Feld subResult der Datenstruktur Behaviour_t data, die jedes Verhalten übergeben bekommt. Die möglichen Werte stehen in bot-logic.h.

Kochrezept

Beim Erstellen eines ganz neuen Verhaltens hilft folgendes Rezept:

1. Fertigen Sie von den Dateien behaviour_prototype.c und behaviour_prototype.h in den jeweiligen Verzeichnissen Kopien an.
2. Ersetzen Sie sowohl im Dateinamen als auch im Quelltext jedes Vorkommen von „prototype" respektive „PROTOTYPE" durch den Namen Ihres Verhaltens.
3. Leiten Sie aus der Zeile #define BEHAVIOUR_PROTOTYPE_AVAILABLE in available_behaviours.h eine für das neue Verhalten passende ab und fügen Sie die neu erstellte Header-Datei zu den Includes am Ende von available_behaviours.h hinzu.
4. Ergänzen Sie bot_behave_init() in bot-logic.c. Die in der dortigen Kopiervorlage eingetragene Priorität von 102 ist ein guter Ausgangswert.

Das war es eigentlich auch schon – es sei denn das neue Verhalten soll auch als Remote-Call zur Verfügung stehen. Dann fehlt noch ein Eintrag in der Konstante remote_call_beh_list in der Datei behaviour_remote_calls.c.

Eine Übersicht über alle bereits verfügbaren Verhalten sowie c't-Artikel, die den Aufbau und die Funktionsweise des Frameworks im Detail beschreiben, finden Sie im Wiki.

Ausblick

Läuft der Code erst einmal im Simulator, sollte er auch schnell auf einen bereits fertig aufgebauten und kalibrierten realen Bot übertragbar sein. Die Entwicklungsumgebung auf der Heft-DVD enthält bereits einen AVR-Compiler und für Eclipse haben wir die dafür nötigen „Build Configurations" bereits angelegt. Wer seinen Teilesatz gerade erst auf dem Tisch ausgebreitet hat, sollte jedoch erst einmal in Ruhe die Aufbauanleitung abarbeiten. Für den Test der Hardware gibt es vorkompilierte Prüfprogramme, die mit Display und LEDs zeigen, ob alle Sensoren funktionieren. (bbe)

www.ct.de/cs1120100

Wichtige Code-Dateien

Datei	Beschreibung
include/ct-bot.h	Globale Konfigurationsoptionen; An- und Ausschalten von Code-Teilen
include/bot-logic/available_behaviours.h	An- und Ausschalten von Verhalten
bot-logic/bot-logic.c	Initialisieren von Verhalten
include/bot-local.h	Globale Definitionen
include/sensor.h	Deklaration der Sensorvariablen
bot-logic/behaviour_simple.c	Beispielverhalten für erste Tests
bot-logic/behaviour_prototype.c	Vorlage für eigene Verhalten
include/bot-logic/behaviour_prototype.h	Rohling für eigene Verhalten
ui/rc5.c	Belegung der Fernbedienungstasten

c't-Sim

Die von uns vorbereitete Eclipse-Umgebung auf der Heft-DVD enthält neben dem Roboter-Code auch den des Simulators. Ein klick auf das kleine „Play"-Symbol in der Menüleiste und dann auf ct-Sim startet ihn und lädt auch gleich ihr selbst kompiliertes ct-Bot.exe.

Im Unterverzeichnis parcours des Projektes ct-Sim befinden sich die Beschreibungen der Testwelten für die Bots. Weil es sich dabei um einfache XML-Dateien handelt, können sie mit wenigen Handgriffen eigene Testparcours erschaffen. Normalerweise wird automatisch testparcours2.xml geladen, aber das können Sie in der Datei config/ct-sim.xml leicht ändern.

TIPP

Hat ein neuer Bot erst einmal anhand der Testprogramme seine Feuertaufe bestanden, kann man den c't-Sim auch benutzen, um ihn fernzusteuern und Debug-Ausgaben anzuzeigen. Die Verbindung kann per WLAN oder USB erfolgen.