BACHELORARBEIT

Entwurf einer grafischen und textbasierten Programmiermethode für die Ablaufsteuerung eines mobilen Roboters

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Bakkalaureus der Technischen Informatik

unter der Leitung von

Univ.Ass. Dipl.-Ing. Dr.techn. Wilfried Elmenreich
Institut für Technische Informatik 182

durchgeführt von

Christoph Martinek
Matr.-Nr. 0425174
Orchisgasse 71/4, 1220 Wien

Wien, im Oktober 2007 .....

Entwurf einer grafischen und textbasierten Programmiermethode für die Ablaufsteuerung eines mobilen Roboters

Der ct-Bot ist ein Bausatz-Roboter der zum Erlernen der Roboter-Programmierung entwickelt wurde. Die Aufgabenstellung bestand darin, den ct-Bot in dem Sinn zu steuern, dass mit der Eingabe von simplen Textbefehlen (ähnlich der Programmiersprache 'Logo') das Fahrverhalten bestimmt wird. Zusätzlich wurde noch eine graphische Eingabe implementiert, wobei eine Strecke vom Benutzer gezeichnet wird, die der Roboter anschliessend abfährt.
Es wurde darauf geachtet, dass die Textbefehle in jedem beliebigen Text-Editor geschrieben werden können. Für die graphische Eingabe wird ein Grafikprogramm benötigt, welches den Umgang mit Vektorgrafiken beherrscht.
Dieses Projekt wurde hauptsächlich entwickelt, um Kindern und Schülern einen Einblick in die Programmierung und Robotersteuerung zu geben. Es wird somit spielerisch der Umgang mit komplizierten Techniken erlernt und ein Bewusstsein für die Auswirkungen des selbst erstellten Programms geschaffen.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzungen

Das Hauptziel dieser Arbeit war die Entwicklung einer Steuerung für den vom Heise-Verlag angebotenen ct-Bot. Dadurch das der Roboter diverse Sensoren und eine Textausgabemöglichkeit in Form eines eingebauten Displays beinhaltet, bietet er somit eine Vielzahl an Einsatzmöglichkeiten und damit verbundene Applikationen.
Die Vorgabe war die Steuerung des ct-Bots möglichst einfach erlernbar zu gestalten. Es wurden dabei Befehle gewählt, die ähnlich der Programmiersprache Logo sind. Der Upload des Programms auf den ct-Bot ist durch die Eingabe eines einzigen Befehls unkompliziert gestaltet.
Die erzielten Ergebnisse stellen eine Weiterentwicklung der ct-Bot-Software dar. Dieses Projekt wurde hauptsächlich entwickelt, um Schülern der Kinderuniversität im Alter von 7 bis 12 Jahren einen Einblick in die Roboterprogrammierung zu geben und um zu veranschaulichen, wie mit den einzelnen Werkzeugen umgegangen wird. Zur Einschätzung von Entfernungen kann der ct-Bot durch ein vordefiniertes Labyrinth gelotst werden. Zusätzlich können somit Ablaufmechanismen geschult werden.

1.2 Aufbau der Arbeit

Diese Bakkalaureatsarbeit ist folgendermaßen strukturiert:

Kapitel 2 gibt einen Einblick in die Programmiersprache 'Logo' und der Anwendung 'Turtlegraphics'. Anschließend wird in diesem Kapitel der ct-Bot, welcher die Hardware- und Softwareplattform für das Projekt darstellt, näher beschrieben.

Im Kapitel 3 wird die Vorgangsweise beim Erstellen des Projekts erläutert. Zusätzliche wird auf die Problemstellung der Text- bzw. Grafik-Konvertierung, deren Programmierung und auf den Upload des Programms auf den ct-Bot näher eingegangen.

Kapitel 4 gibt einen genauen Einblick, wie die Implementierung der Programmteile stattgefunden hat. Dabei wird näher auf die Ablaufsteuerung und die darin ausgeführten Befehle eingegangen. Weiters werden die Sensoren und Motoren des ct-Bot genauer beschrieben.

Im Kapitel 5 werden Schlüsse aus den gewonnenen Erkenntnissen gezogen. Zusätzlich wird diskutiert, ob die Aufgabenstellung zufriedenstellend erfüllt wurde und welche Abweichungen festgestellt wurden.

Das letzte Kapitel beinhaltet eine Zusammenfassung der vorgestellten Arbeit und welche Ausblicke es in Zukunft in diesem Bereich geben wird.

2 Verwandte Arbeiten

2.1 Programmiersprache Logo

Die Programmiersprache Logo ist eine sehr leicht und intuitiv zu verstehende Programmiersprache und wurde vor allem für Programmier-Anfänger entwickelt. Das Grundkonzept wurde in den 1960er Jahren von Seymour Papert verwirklicht [Sch03]. Dabei stand in den Anfängen des modernen Computerzeitalters die Einfachheit der Programmierung im Mittelpunkt. Diese Sprache beinhaltete für die damalige Zeit einige relativ hilfreiche Elemente, wie zum Beispiel rekursiv aufrufende Funktionen und dynamische Datentypenerkennung. Jedoch wurde sie wegen eines scheinbaren Mangels an Leistungsfähigkeit unterschätzt und fand nicht so große Verbreitung wie zum Beispiel die Programmiersprache BASIC. Als zusätzlicher Kritikpunkt wurde früher die Rekursion empfunden, da sie weniger gut nachvollziehbar sei und mehr Rechenzeit in Anspruch nehme.

2.2 Turtlegraphics

Hierbei handelt es sich um eine Applikation, die auf der Sprache Logo aufbaut. Es können auf einer zweidimensionalen Oberfläche ein bzw. mehrere Schildkröten mittels Steuerbefehle bewegt werden. Die Wege, die die Schildkröten gegangen sind, werden mit Linien gekennzeichnet und es entsteht dadurch eine Zeichnung. Die wichtigsten Turtlegraphics-Befehle (die in [Wik07] genauer erklärt werden) werden hier kurz aufgeführt:

- penup (pu): Stift auf Zeichenfläche aufsetzen
- pendown (pd): Stift von der Zeichenfläche anheben
- home: Schildkröte in der Mitte der Zeichenfläche hinaufschauend platzieren
- clean: Bildschirm wird gelöscht
- textscreen (ts): wechselt zu Text-Eingabe-Modus

- fullscreen (fs): wechselt zu Grafik-Modus
- right (rt + Argument): Rechtsdrehung der Schildkröte um angegebene Gradanzahl
- left (lt + Argument): Linksdrehung der Schildkröte um angegebene Gradanzahl
- forward (fd + Argument): Vorwärtsbewegung der Schildkröte um angegebene Streckenlänge
- back (bk + Argument): Rückwärtsbewegung der Schildkröte um angegebene Streckenlänge

2.3 ct-Bot

2.3.1 Allgemeines

Der ct-Bot ist ein Baukasten-Roboter, dessen Design im ct-Magazin [Ben06] vorgestellt wurde. Dieser wird mit allen benötigten Bauteilen und Platinen geliefert.
Neben der Hardware wird vom Heise-Verlag ein Framework mit allen für den Roboter notwendigen Funktionen auf der Homepage als Download angeboten[1]. Weite Teile dieses Frameworks wurden jedoch in diesem Projekt nicht verwendet, da die Software zu umfangreich wäre. Welche Software-Teile genau verwendet wurden, wird in Kapitel 4 genauer erklärt.
Es wird darüberhinaus auch eine Software-Simulation angeboten, bei der das Verhalten des ct-Bot am Rechner simuliert werden kann. Dies ist sehr hilfreich, da das ständige Downloaden des selbst geschriebenen Programms und Nachvollziehen der Bewegungen des Roboters viel Zeit in Anspruch nehmen und die Mechanik des ct-Bots einer Dauerbelastung aussetzen würde. Diese Simulationssoftware ist in der Programmiersprache Java verfasst worden und ist daher plattformunabhängig.

2.3.2 Technische Daten

Motoren

Die Fortbewegung und dadurch die Haupteigenschaft des Roboters werden von zwei Gleichstrommotoren der Firma FTB Faulhaber, Modell: 2619-006SR

[1] ct Projekte - ct-Bot und ct-Sim, Downloads; `http://www.heise.de/ct/projekte/ct-bot/download.shtml`

Abbildung 2.1: Der ct-Bot

[Fau07] erledigt. Diese beiden Motoren, die über eine Spannung von 6V versorgt werden, haben ein Übersetzungsgetriebe mit dem Verhältnis 33:1. Dadurch wird weniger Geschwindigkeit, jedoch mehr Drehmoment (insgesamt 0,03 Nm) erreicht. An der Hinterseite des Roboters befindet sich ein abgerundeter Kunststoffteil, dass zur Stabilisierung beiträgt. Es kann bei einem Raddurchmesser von 57mm und einer Umdrehungsgeschwindigkeit von maximal 151 U/min eine Höchstgeschwindigkeit (jedoch nur bei unbelastetem Roboter) von 0,45 m/s, das entspricht 1,62 km/h, erreicht werden.
Die Räder sind aus Aluminium gefertigt und zur Gewichtsreduzierung wurden 6 Löcher aus den Rädern ausgefräst. Um genug Bodenhaftung zu gewährleisten, wurde ein O-Ring aus Kunststoff auf die Scheiben gezogen.

Rad-Encoder

Um die Bewegung der Räder zu überwachen und daraufhin zu steuern werden Reflexlichtschranken der Firma Vishay, Modell CNY70 [Vis00], verwendet. Auf den Rädern wurden auf den Innenseiten Kodierscheiben mit abwechselnd schwarzen und weißen Feldern aufgeklebt. Durch die Detektion der Abwechslung zwischen schwarz und weiß kann die Drehgeschwindigkeit und dadurch auf die zurückgelegte Strecke ermittelt werden.

Display

Das Display der Firma Hitachi (Modell: HD44780U (LCD-II) [Hit98]) ist ein Punkt-Matrix-LCD-Display, mit dem in 4 Zeilen jeweils 20 Zeichen ausgegeben

werden können. Es beinhaltet einen Controller, über den die einzelnen Characters (5x8 bzw. 5x10 Punkte) angesprochen werden.

Prozessor

Der ursprünglich für den ct-Bot vorgesehene Mikrokontroller (ATmega16) wurde durch den pinkompatiblen ATmega32 [Atm07] ersetzt, welcher mehr Speicher bietet. Der verwendete Mikrokontroller besitzt 32K Bytes In-System Programmable Flash-Speicher und verfügt zusätzlich über 1024 Bytes EEPROM und 2K Byte Internal SRAM.
Der Mikrokontroller bietet zwei 8-bit und einen 16-bit-Timer, eine Programmable Serial USART-Schnittstelle und einen 10-bit ADC mit 8 Kanälen.

Die nun folgenden Sensoren werden in diesem Projekt nicht verwendet, befinden sich jedoch ebenfalls auf dem ct-Bot und werden somit kurz erklärt:

Abstandssensoren

Zwei Abstandssensoren (GP2D12 [Sha97]) der Firma Sharp geben Aufschluss über die Entfernung von Objekten zum ct-Bot. Dabei wird von einer Infrarotdiode Licht auf das Objekt ausgestrahlt und später von diesem auf den Sensor zurückreflektiert. Wie weit das Objekt vom Roboter entfernt ist, wird durch den Winkel des reflektierten Lichtstrahls berechnet. Der Sensor liefert eine analoge Spannung abhängig von der Entfernung des Objektes und kann somit Entfernungen von 10 bis 80 cm detektieren.

Liniensensoren

Die schon erwähnten Lichtschrankensenoren von Vishay (CNY70) werden auch auf der Unterseite des ct-Bot verwendet. Diese liegen eng beieinander und können somit eine schwarze Linie, die auf dem Boden gezeichnet ist, problemlos detektieren.

Abgrundsensoren

Auf der vorderen Seite des Roboters sind nach unten gerichtet wiederum Lichtschrankensensoren (CNY70) angebracht. Diese können feststellen, ob vor dem ct-Bot eine schwarze Fläche oder ein Abgrund vorhanden ist. Durch diese Sensoren kann der Roboter vor Schäden bewahrt werden.

Photosensoren

Nach vorne gerichtet sind zwei Photosensoren (LDR), die eine Lichtquelle erkennen können. So könnte zum Beispiel eine getragene Taschenlampe vom Roboter verfolgt werden.

Maussensor

Auf der Unterseite des ct-Bot befindet sich noch zusätzlich ein Maussensor (ADNS2610), der die genaue Position, unterstützend mit den Rad-Encodern ermitteln kann.

2.3.3 Simulation

Um möglichst schnell die Auswirkungen und das Verhalten des Roboters zu betrachten, gibt es auf der Heise-Homepage[2] eine Simulatorumgebung. Dieser Simulator ist in der Programmiersprache Java geschrieben und ist dadurch plattformunabhängig.
Zusätzlich wird eine Java-Laufzeitumgebung und die Java-3d-Bibliothek benötigt. Die generierten ausführbaren Dateien *ct-Bot.exe* in Windows oder *ct-Bot.elf* in Linux, die die Routinen für die Steuerung des ct-Bot beinhalten, werden nach der Ausführung von *ct-Bot.jar* in die Simulation importiert. Dies geschieht dadurch, dass sich der ct-Bot über eine TCP/IP-Verbindung beim Simulator unter der lokalen IP 127.0.0.1 auf dem Port 10001 anmeldet.
Im Simulatorfenster werden sowohl die genaue Position des Roboters, als auch alle Sensorwerte und Displayausgaben angezeigt. Zusätzlich können eigene Welten mit dem Weltgenerierungs-Programm erschaffen und die Reaktionen des ct-Bot genau beobachtet werden. Die folgende Abbildung zeigt das Simulationsfenster:

[2]Heise ct-Bot-Simulator, `http://www.heise.de/ct/projekte/ct-bot/ct-sim.shtml`

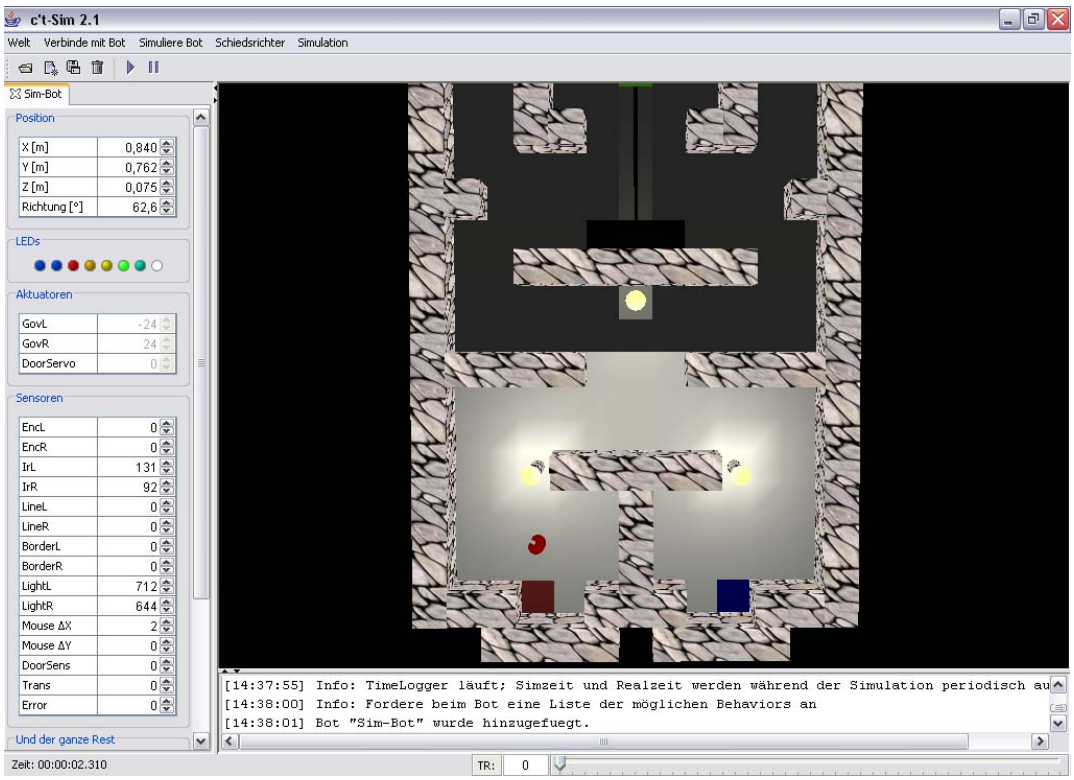

Abbildung 2.2: ct-Simulatorfenster

3 Entwicklungsansatz

3.1 Programmierung mittels textueller Eingabe

Hierbei wird eine ähnlicher Syntax wie bei der Sprache Logo verwendet. Insgesamt stehen fünf Befehle zur Auswahl:

- FORWARD (Vorwärts) = F + 'Argument'
- BACKWARD (Rückwärts) = B + 'Argument'
- LEFT (Links) = L + 'Argument'
- RIGHT (Rechts) = R + 'Argument'
- STOP (Stopp) = S

Diesen folgen die entsprechenden Argumente; bei Vor- bzw. Rückwärtsbewegung wird die Entfernung in Zentimeter (cm) und bei Links- bzw. Rechtsdrehungen wird der Drehwinkel in Grad angegeben. Eine Liste von Befehlen werden untereinander (Befehl und Argument getrennt durch Leerzeichen) geschrieben. Am Ende jedes Programms sollte ein Stopp-Befehl stehen, welcher anzeigt, dass das Programm beendet ist.
Das Programm wird in die Datei *logo*, welche sich im Hauptverzeichnis befindet, mit jedem beliebigen Text-Editor geschrieben. Wie ein solches Programm ungefähr aussehen könnte, sieht man in der Abbildung 3.1.

Dieses Programm lässt den Roboter 'eine Treppe mit drei Stufen' nach rechts und anschließend zu seiner Ausgangsposition zurückfahren.
Anschließend muss als Kommandozeile (ausgehend vom Hauptverzeichnis des ct-Bot) der Befehl ***make text*** eingegeben werden, um die erforderlichen Dateien zum Download zu generieren.
Mit dem Befehl ***make text_install*** wird das aktuelle Programm ebenfalls übersetzt und zusätzlich auf den ct-Bot geladen.

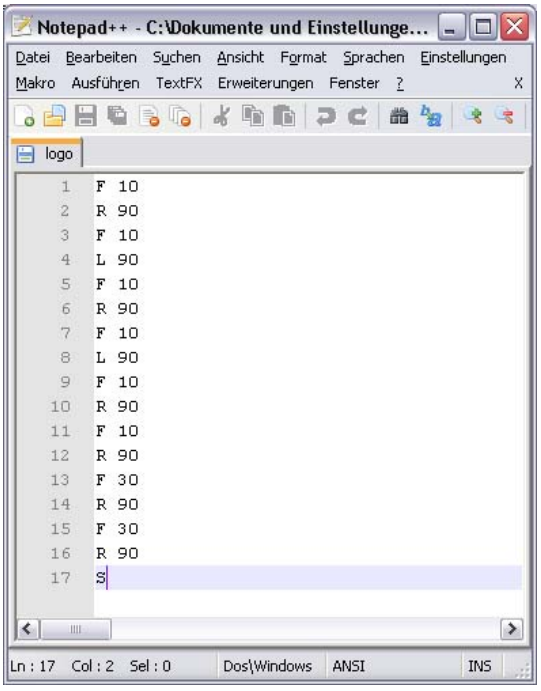

Abbildung 3.1: Logo-Programm für den ct-Bot

3.2 Programmierung mittels grafischer Eingabe

Bei dieser Möglichkeit den Ablaufalgorithmus des Roboters zu generieren, wird eine grafische Oberfläche verwendet. Es kann auf einer Fläche von 250x140 cm die Bewegung des Roboters an Hand einer Linie vorgegeben werden. Dabei steht ein Raster von 10x10 cm zur Verfügung. Diese wird in einer svg-Datei abgespeichert und generell mit dem 'Bezier-Werkzeug' gezeichnet. Dabei ist zu beachten, dass eine Linie nicht auf zwei oder mehr Schritte gezeichnet werden darf, sondern eine Strecke in einem Vorgang zu zeichnen ist. Der Startpunkt des gezeichneten Wegs ist frei zu wählen. Als mögliches Programm für die grafische Eingabe steht **Inkscape**[3] zur Verfügung. In diesem Programm wird die Linie mit dem 'Bezier-Kurven und gerade Linien zeichnen'-Werkzeug gezeichnet. Durch die Tastenkombination 'Umschalt-F6' kann diese sofort aktiviert werden.
Die gezeichnete Linie wird in der Datei *graph.svg* im Hauptverzeichnis des ct-Bot abgespeichert. Abbildung 3.3 zeigt das Grafik-Eingabe-Fenster (rechts) inklusive Texteingabe (links) und Kommandozeile (unten):

Mit dem Befehl ***make graph*** wird die Datei *graph.svg* nach einer zu fahrenden Linie durchsucht und in die dadurch extrahierten Befehle in die Datei *logo* gespeichert. Anschließend werden aus den Befehlen der Datei *logo* alle zum

[3]Inkscape Projekt und Download, `http://www.inkscape.org/`

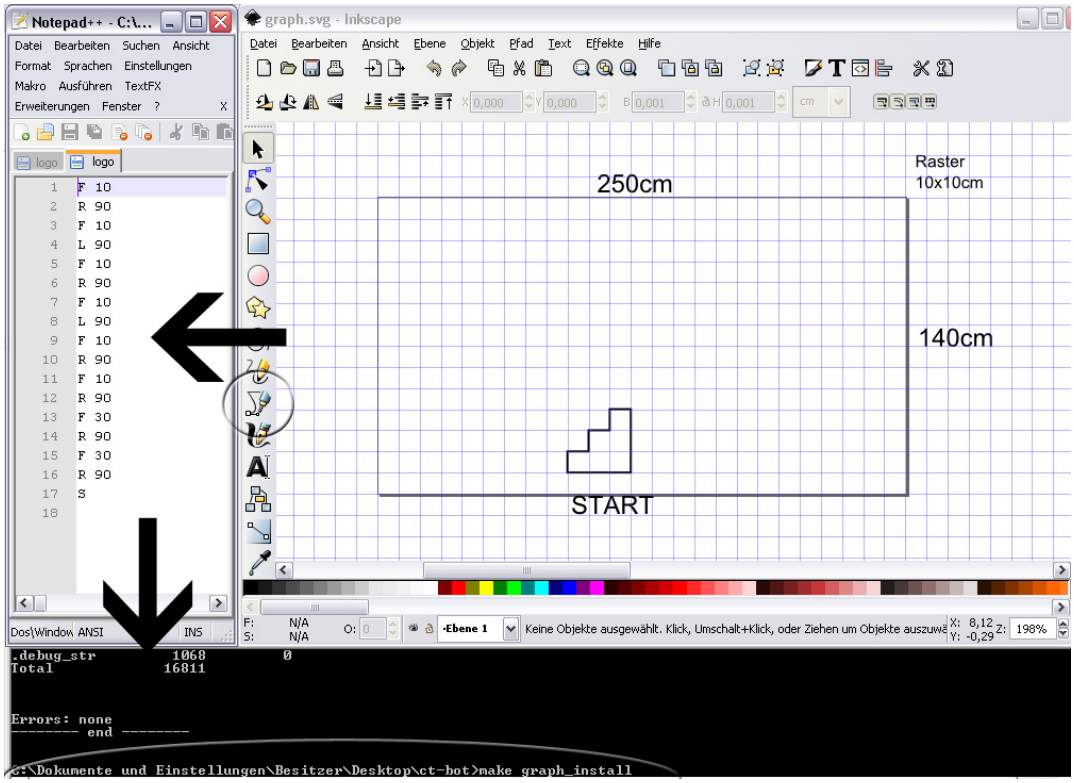

Abbildung 3.2: Programm-Screen

Upload notwendigen Files erzeugt.
Wiederum kann mit dem Befehl ***make graph_install*** die Ablaufsteuerung für die gezeichnete Fahrlinie direkt auf den ct-Bot geladen werden.

3.3 Laden des Programms auf den ct-Bot

Der Upload wird automatisch mit den Befehlen ***make text_install*** und ***make graph_install*** ausgeführt. Dabei wird das Programm **Zeusprog** aufgerufen. Der genaue Befehl für den Upload lautet: ***zeusprog -f i -m f -p 0 bot.hex***. Dabei wird das generierte hex-File in den Flash-Memory des ATmega32 des ct-Bots geladen.

4 Implementation

4.1 Text-Konvertierung

Bei diesem Ansatz werden die Logo-Befehle in eine Ablaufsteuerung (die genauer noch weiter unten erklärt wird) in die Datei *befehle.*c konvertiert. Dies wird dadurch erreicht, dass im Programm **convert_text** die Datei *logo* geöffnet und nach den definierten Befehlen analysiert wird. Danach wird das dazugehörigen Argument aus der Angabe extrahiert. Diese Daten werden in der Struktur 'prog[i].Befehl' und 'prog[i].Argument' abgespeichert, wobei i die Befehlsnummer angibt.
Anschließend werden die aus dem Parsen gewonnene Daten in Form einer State-Machine in die Datei *befehle.c* geschrieben, die wiederum vom Hauptprogramm ausgeführt wird.

4.2 Grafik-Konvertierung

4.2.1 Konvertierung

Hierbei wird eine Linie, die mit dem Programm **Inkscape** gezeichnet wurde, zur Befehlsgenerierung analysiert, um daraus die einzelnen Befehle zu gewinnen. Dies wird damit bewerkstelligt, indem die Datei *graph.svg* mit dem Programm **convert_graph** geöffnet und nach dem String gesucht wird, der die Informationen und Koordinaten der Linie beinhaltet. Anschließend werden die Koordinaten miteinander verglichen und dadurch können die gewünschten Bewegungen des Roboters nachvollzogen werden. Nun folgt eine Winkelberechnung für die Drehungen und danach werden diese Daten in 'Winkel[i]' und 'Distanz[i]' (i bezeichnet wiederum die Befehlszahl) abgespeichert. Zum Schluss werden Winkel und Distanzen in die benötigte Befehlsform umgewandelt und in das File *logo* geschrieben.

4.2.2 Problematiken

Bei dieser Aufgabe, war es besonders schwierig die Koordinaten aus der Grafikdatei herauszufiltern. Weiters war es kein Leichtes die Koordinaten so aus-

zuwerten, dass daraus die Befehle extrahiert werden konnten.
Anzumerken ist, dass die Konvertierung aus dem Grafikprogramm nur eine Drehung von 90 Grad unterstützt.

4.3 Ablaufsteuerung

Die Ablaufsteuerung wird in der Datei *befehle.c* ausgeführt. Diese zeichnet sich dadurch aus, indem sie wie eine State-Machine realisiert wurde. Das bedeutet, dass ein Zustand dem nächsten folgt und jeder Befehl (jeweils getrennt durch einen Warten-Befehl) fortschreitend ausgeführt wird. Diese Datei wird von **convert_text** erzeugt und wird bei jedem neuen Befehlssatz (gleich ob von der Text- oder Grafikeingabe) generiert. Dabei ist zu beachten, dass die gewünschten Argumente zu den Bewegungen (Zentimeter oder Grad) zuerst aus den Rad-Encoder-Sensor-Werten berechnet werden müssen.
Die Zentimeterangaben werden so umgesetzt, dass ein gefahrener Zentimeter 3.2 Steps des Rad-Encoders entspricht. Der Wert für den Rad-Encoder wird über den Mittelwert von beiden Rad-Encoder-Sensoren berechnet.
Für die Winkelangaben werden ähnliche Berechnungen verwendet. Bei der Drehung werden die Motoren gegengleich angesteuert und wieder die Rad-Encoder zur Kontrolle des Winkels herangezogen. Dabei entsprechen 100 Schritte einer Vollumdrehung des ct-Bot.

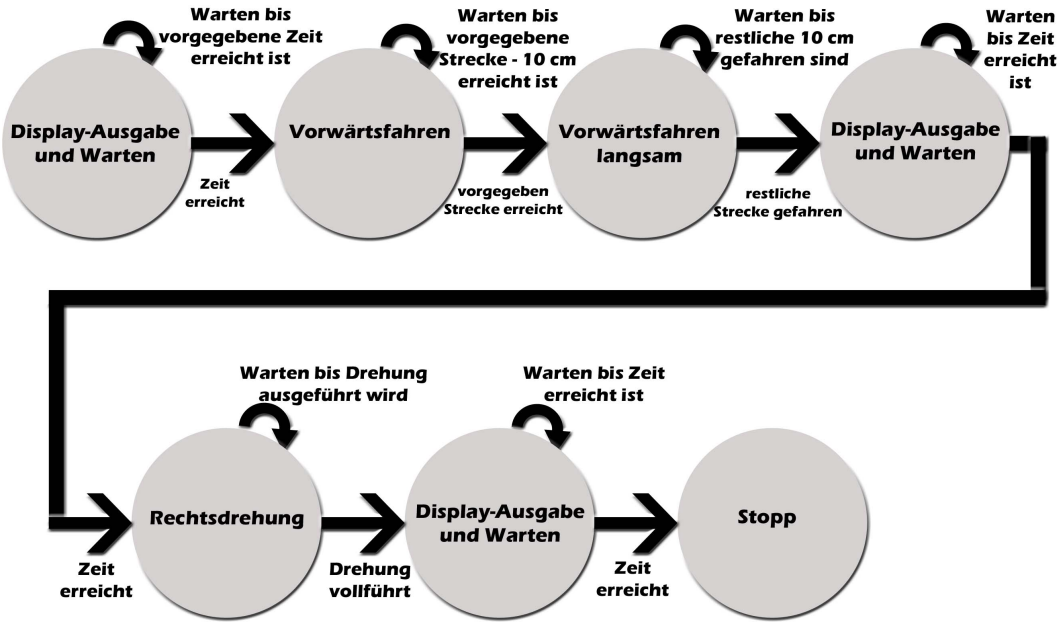

Abbildung 4.1: State-Machine

Diese Ablaufsteuerung wird aus dem Hauptprogramm in einer ISR (Interrupt Service Routine) in regelmäßigen Abständen ausgeführt.
Als Beispiel dient ein kleines Programm indem die drei Befehle ausgeführt werden. Die folgende Abbildung zeigt die Reihenfolge der Abarbeitung:

Es werden im folgenden die drei zugehörigen Zustandsbeispiele kurz beschrieben:

4.3.1 Vorwärts

```
/*Display-Ausgabe und Warten*/
case 0: motor_set(0,0); wait++;
if (wait == 100){
display_clear();
display_cursor(1,2); display_printf("1.Command from 3");
display_cursor(3,2); display_printf("FORWARD   100 cm");
display_cursor(4,2); display_printf("gefahren:   0 cm");
}
if (wait == 5000){wait = 0; state++; return 1;}
break;
```

Dieses Codestück zeigt das anfängliche Stillstehen des ct-Bot und Ausgabe des nächsten Befehls auf dem Display. Dabei wird in der ersten Zeile das jeweilige Kommando in Bezug auf die Gesamtanzahl der Kommandos ausgegeben `display_printf("1.Command from 3")`. Die dritte Zeile des Displays beinhaltet den Befehl (also in diesem Fall FORWARD und das dazugehörige Argument 100 cm). Die letzte Zeile zeigt die bereits zurückgelegte Strecke in Zentimeter an. Anschließend wird der nächste Zustand mit `state++` aktiviert. In diesem Zustand beginnt der Roboter sich geradeaus zu bewegen `motor_set(85,86)`. Um eine gerade Vorwärtsfahrt zu ermöglichen, wird der rechte Motor um einen geringen Teil schneller bewegt als der linke, da die Geschwindigkeit der Motoren bei dem verwendeten Roboter leicht abweicht. In der nächsten Zeile wird überprüft, ob schon die geforderte Strecke weniger zehn Zentimeter gefahren wurde. Wenn dies der Fall ist, wird wiederum der nächste Zustand aktiviert. In diesem werden die letzten zehn Zentimeter in langsamer Fahrt fortgesetzt und die Display-Ausgabe des aktuellen gefahrenen Werts angezeigt. Die letzten zehn Zentimeter werden in langsamer Geschwindigkeit gefahren, da sonst der Roboter über den vorgegebenen Sollwert hinaus weiterfahren würde, weil er nicht aus einer höheren Geschwindigkeit sofort stehenbleiben könnte.

```
/*Drive Forward*/
case 1: motor_set(85,86);
if (cm_ausgabe >= 100-10) {state++; return 1;}
display_cursor(4,12); display_printf("%3d", cm_ausgabe);
break;

/*Drive Forward slow*/
case 2: motor_set(55,55);
if (cm_ausgabe >= 10) {state++; return 1;}
display_cursor(4,12); display_printf("%3d", (100-10)+cm_ausgabe);
break;
```

4.3.2 Rechtsdrehung

Hier wird zuerst wieder die Displayausgabe mit Anführen des Kommandos, Argument und bereits gedrehter Umdrehung ausgegeben. Es wird wieder eine gewisse Zeit gewartet, bis der Roboter stillsteht und der nächste Befehl am Display abzulesen ist.

```
/*Display-Ausgabe und Warten*/
case 3: motor_set(0,0); wait++;
if (wait == 100){
display_clear();
display_cursor(1,2); display_printf("2.Command from 3");
display_cursor(3,4); display_printf("RIGHT     90 grad");
display_cursor(4,2); display_printf("gedreht:   0 grad");
}
if (wait == 5000){wait = 0; state++; return 1;}
break;
```

Daraufhin wird die Drehung des Roboters eingeleitet. Dies geschieht, indem die Motoren mit der selben Geschwindigkeit gegengleich angesteuert werden. Dadurch wird erreicht, dass sich der ct-Bot an der gleichen Position dreht. Folgend ist zu erkennen, dass sich der Roboter so lange dreht, bis die gewünschte Gradanzahl erreicht ist. Daraufhin wird der Zustand in der State-Machine wieder inkrementiert.

```
/*Turn Right*/
case 4: motor_set(80,-80);
if (grad_ausgabe >= 90) {state++; return 1;}
display_cursor(4,11); display_printf("%3d", grad_ausgabe);
break;
```

4.3.3 Stopp

Der letzte Zustand der beschrieben wird, ist der Endzustand den der Roboter am Schluss des Programms ausführt. Auf dem Display wird der Befehl STOP ausgegeben und die Motoren werden deaktiviert. Anschließend wird der Timer2, in der die Ablaufsteuerung aufgerufen wird, deaktiviert und die rote LED aktiviert:

```
/*Display-Ausgabe und Warten*/
case 5: motor_set(0,0); wait++;
if (wait == 100){
display_clear();
display_cursor(1,2); display_printf("3.Command from 3");
display_cursor(3,3); display_printf("-----STOP-----");
LED_on(LED_ROT); TIMSK &= ~(1<<OCIE2); LED_off(LED_GRUEN); }
if (wait == 5000){wait = 0; state++; return 1;}
break;

/*Stop*/
case 6: motor_set(0,0); wait++;
if (wait == 5000){wait = 0; state++; return 1;}
break;
```

4.4 Verwendete Programme

Es wurden Programmteile vom ct-Bot-Downloadverzeichnis[4] verwendet. Diese sind Low-Level-Befehle für die Motoren- und Displaysteuerung und wurden für den Zweck dieses Projektes modifiziert. Welche Programmteile und inwiefern diese verändert wurden wird folgend genauer erläutert:

1. C-Files

   - delay.c:
     Beinhaltet Hilfsroutinen für Warte-Befehle, Delay-Funktionen für 5 und 10 ms eingeführt.
   - display.c:
     Routinen zur Steuerung des Displays, in der Routine 'display_clear()' ein Delay von 5 ms eingefügt und die Steuerungsbefehle für die

[4] Heise ct-Bot-Downloadverzeichnis, `https://www.heise.de/ct/projekte/machmit/ctbot/browser/stable/ct-Bot`

Fernsteuerung entfernt, da es sonst hier zu Konflikten kam. In 'display_init()' wiederum die Befehle 'delay_5ms()' bzw. 'delay_10ms()' eingefügt. In 'display_printf()' Steuerungsbefehle für die Fernbedienung entfernt.

- led.c:
  Programm zur Ansteuerung der LEDs, unverändert übernommen.
- motor.c:
  Beinhaltet Routinen zur Ansteuerung der Motoren, unverändert übernommen.
- motor-low.c:
  Low-Level-Befehle zur Motorsteuerung, unverändert übernommen.
- sensor-low.c:
  Programm zur Sensorsteuerung. Sämtliche, für dieses Projekt unnötigen, inkludierten Headerfiles und Pinbelegungen der anderen nicht verwendeten Sensoren wurden entfernt. In der Routine 'bot_sens_init()' die Initialisierung aller nicht verwendeten Sensoren und Initialisierung des Timer2 deaktiviert. Die Routine 'bot_sens_isr()' gelöscht und 'bot_encoder_isr()' für linken und rechten Rad-Encoder aufgesplitet und in 'bot_encoder_rechts()' und 'bot_encoder_links()' umbenannt. In diesen Routinen wurde das Rückzählen des Encoderstandes beim Rückwärtsfahren entfernt.
- shift.c:
  Programm enthält die Ansteuerung für Shift-Register. Headerdateien, vordefinierte Konstanten und Routinen, die für die Simulation am PC notwendig sind, wurden entfernt, weil durch Änderung des Timer2 und der Zeitsteuerung die Simulation am PC nicht verfügbar ist.
- timer.c:
  Routinen für Einstellungen und Initialisierungen der Timer, unverändert übernommen.

2. Header-Files

- bot-local.h:
  Dieses Headerfile wurde unverändert übernommen.
- command.h:
  Dieses Headerfile wurde unverändert übernommen.
- ct-Bot.h:
  Dieses Headerfile wurde unverändert übernommen.

- delay.h:
  Delay-Funktionen für 5 und 10 ms wurden eingefügt, unverändert übernommen.
- display.h:
  Dieses Headerfile wurde unverändert übernommen.
- global.h:
  Dieses Headerfile wurde unverändert übernommen.
- led.h:
  Dieses Headerfile wurde unverändert übernommen.
- motor.h:
  Dieses Headerfile wurde unverändert übernommen.
- motor-low.h:
  Dieses Headerfile wurde unverändert übernommen.
- sensor.h:
  Dieses Headerfile wurde unverändert übernommen.
- shift.h:
  Dieses Headerfile wurde unverändert übernommen.
- timer.h:
  Timer2 Initialisierung wurde entfernt, da dieser Timer für die Ansteuerung der Ablaufsteuerung in bot.c genutzt wird, sonst unverändert übernommen.

5 Experimente und Auswertung

5.1 Roboterbewegungen ohne Hindernisse

Die Roboterbewegungen wurden auf einer A0-großen Fläche (841 x 1189 mm) mit einem 10 x 10 cm Raster vollführt. Dabei war zu erkennen, dass bei leichten Unebenheiten und Verschmutzungen der Fahroberfläche die Räder zu leichtem Durchdrehen neigten. Dies führte wiederum zum Problem, dass dadurch die Rad-Encoder-Sensoren, welche die zurückgelegte Strecke messen, inkorrekte Daten erhielten und dadurch die zu fahrende Strecke zu früh abbrachen.
Zusätzlich konnte, wie schon in Kapitel 4.3.1 erwähnt, ein 'Hinterherlaufen' des rechten Motors festgestellt werden. Dies wurde damit ausgeglichen, dass am rechten Motor die Geschwindigkeit ein wenig erhöht wurde.
Bei der Vor- bzw. Rückwärtsbewegung konnte eine Ungenauigkeit von plusminus 1 cm auf einer Strecke von 1 m festgestellt werden. Dies ist die Folge von dem schon beschriebenen Durchdrehen der Räder auf der einen und bei zu wenig Haftung beim Abbremsen auf der anderen Seite.
Bei Drehungen konnte keine nennenswerte Abweichung von den Sollwerten festgestellt werden.

5.2 Roboterbewegungen mit Hindernissen

Wie in der folgenden Abbildung zu sehen ist, wurde ein zu durchdringendes Labyrinth mit Hilfe von Hindernissen aufgestellt.

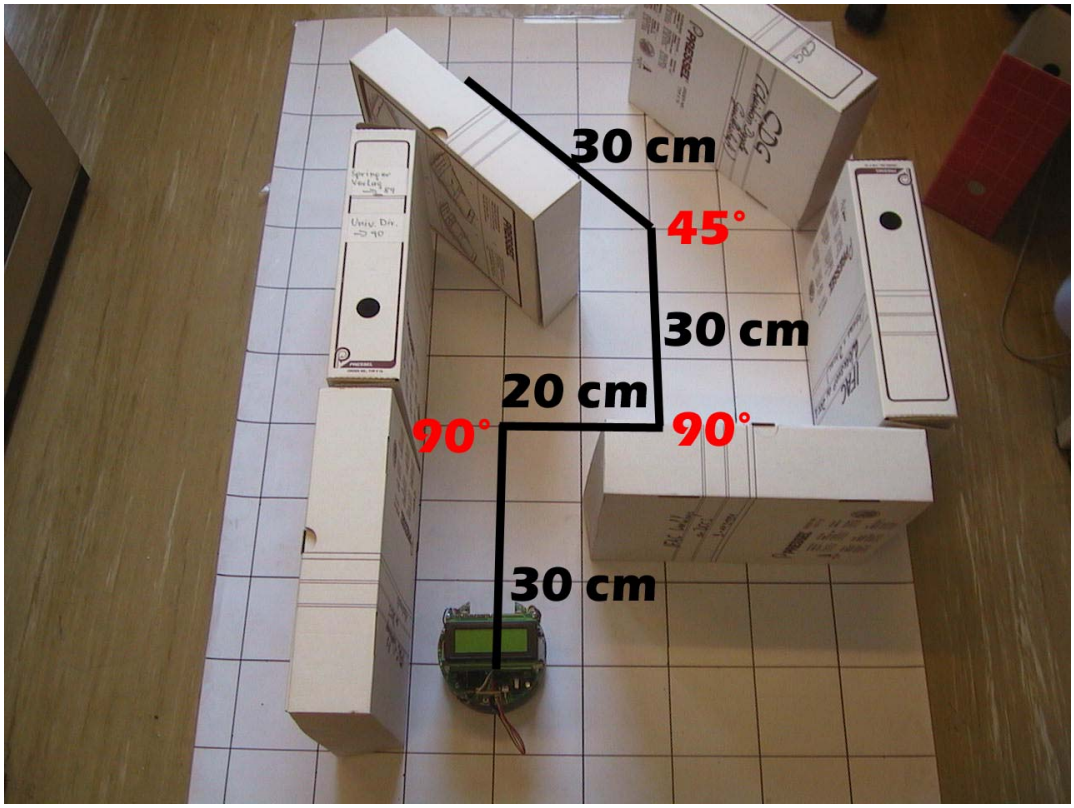

Abbildung 5.1: Labyrinth

Zur Durchquerung dieses Labyrinths muss der Roboter mit den richtigen Steuerungsbefehlen programmiert werden. Die Befehle für den Weg aus dem Parkur lauten wie folgt:

```
F 30
R 90
F 20
L 90
F 30
L 45
F 30
S
```

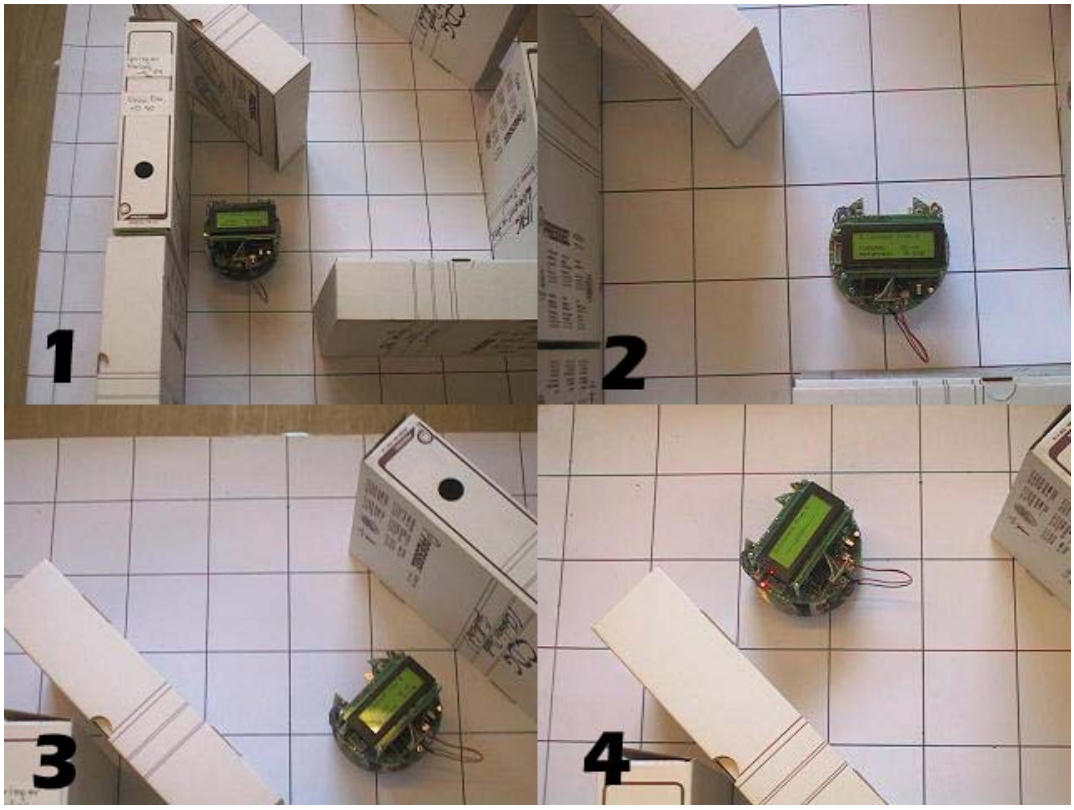

Abbildung 5.2: Weg aus dem Labyrinth

Dadurch wird erreicht, dass der Roboter, ohne an einem Hindernis anzukommen in ungefähr der Mitte des zur Verfügung stehenden Weges das Labyrinth durchfährt.
Es wurde bei der Vorwärtsbewegung wie oben schon beschrieben festgestellt, dass es bei Verschmutzungen und Verunreinigungen des Bodens bei mehrmaligem Wiederholen des Experimentes zu einem leichten Durchdrehen der Räder kommt und dadurch der zurückgelegten Weg beeinflusst wird. Wie man jedoch anhand von Abbildung 5.2 erkennen kann, absolviert der ct-Bot den Parkur fehlerlos.

6 Zusammenfassung

In dieser Arbeit wurde eine Steuerung eines mobilen Roboters mittels einfachen und leicht zu erlernenden Befehlen entworfen und implementiert. Zusätzlich kann dieser ct-Bot noch mit Hilfe einer grafischen Benutzeroberfläche und den daraus resultierenden Befehlen gesteuert werden. Durch 'Lotsen' des Roboters durch einen Hindernisparkur können verschiedene Fähigkeiten, u.a. räumliches Darstellungsvermögen und Entfernungseinschätzungen geschult werden.
Bei der konkreten Realisierung des Projekts konnte festgestellt werden, dass die Oberflächenbeschaffenheit der Fahrstrecke des Roboters wesentlich für die Ausführungsgenauigkeit ist. Auf einer Fahrstrecke von 1 m konnte, auf einem mit Plane aus Papier ausgelegtem Untergrund, eine Ungenauigkeit von 1 cm festgestellt werden. Für den geplanten Einsatzbereich ist diese Genauigkeit ausreichend.

Literatur- und Quellenverzeichnis

[Atm07] Atmel. *8-bit Microcontroller with 32K Bytes In-System Programmable Flash*. USA, 2007. Available at `http://www.atmel.com/dyn/resources/prod\_documents/doc2503.pdf`.

[Ben06] Benjamin Benz, Carl Thiede, Thorsten Thiele. Spielgefährten. Roboter für Löter, Simulator für Soft-Werker. *ct*, pages 130–135, Februar 2006. Available at `http://www.heise.de/ct/06/02/006/`.

[Fau07] FTB Faulhaber. *DC-Getriebemotoren mit eisenlosem Flachläufer, Serie 2619 ... SR*, Februar 2007. Available at `http://www.faulhaber-group.com/uploadpk/d_2619SR_FTB.pdf`.

[Hit98] Ltd. Hitachi. *Reflective Optical Sensor with Transistor Output*. Japan, 1998. Available at `http://web.mit.edu/6.115/www/datasheets/44780.pdf`.

[Sch03] Günther Schättiger. Einstieg in Logo. *Uni-Hamburg*, Februar 2003. Available at `http://www.informatik.uni-hamburg.de/~schaetti/LOGOeinstieg.html`.

[Sha97] Sharp. *GP2D12/GP2D15 Distance Measuring Sensors*, May 1997. Available at `http://personal.telefonica.terra.es/web/x-robotics/downloads/datasheets%/gp2d12.pdf`.

[Vis00] Vishay. *Reflective Optical Sensor with Transistor Output*. Germany, April 2000. Available at `http://personal.telefonica.terra.es/web/x-robotics/downloads/datasheets%/cny70.pdf`.

[Wik07] Wikipedia. Logo (Programmiersprache). *wikipedia*, September 2007. Available at `http://de.wikipedia.org/wiki/Logo_%28Programmiersprache%29`.