Eukalyptos Nutzerhandbuch

Ruprecht-Karls-Universität Heidelberg

Stand: 4. September 2006
Autoren:
Büch, Lutz
Franke, Carlos
Rieck, Bastian

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dies ist eine Nutzungsanleitung für die Benutzerschnittstelle des Projektes „Eukalyptos“.
Hier erfahren Sie alles, was es zur Benutzung zu wissen gibt, und was über die selbsterklärende grafische Oberfläche hinausgeht. Außerdem gehen wir an dieser Stelle davon aus, dass die Verbindung mit dem Serverprogramm hergestellt ist. Es geht also tatsächlich nur um das Benutzen des Programms auf dem „Steuerrechner“.

Kapitel 2

Eingabe von Befehlen

Ohne weiteres sind Befehle auszuführen, wenn man den Roboter „Gingganz“ aus unserem Projekt auf dem Testgelände im Robotiklabor verwendet. Ansonsten sind die Schritte in den Folgenden Kapiteln auszuführen.
Befehle sind im Prinzip sehr einfach aufzurufen. Im Eingabefeld können mehrere Befehle eingegeben werden, jeweils gefolgt von Klammern, die die Parameter enthalten, welche durch Kommata oder Semikola getrennt sind. Befehle untereinander werden nur durch Leerzeichen, Tabulatoren oder Zeilenumbrüche (oder auch gar nicht) getrennt. Groß-, Klein- und Gemischtschreibungen sind äquivalent. Die Parameter müssen immer vollständig sein und deren Werte sind natürliche Zahlen zwischen 1 und 250.
Ein Beispiel:

```
VORWAERTS(89) RecHTS(17) Rueckwaerts(7) LINKS(34)
```

Kapitel 3

Kartenspezifikation

Der Roboter befindet sich in Realität auf irgend einem Gelände, das nicht notwendigerweise begrenzt ist. Im Modell stellen wir dieses allerdings als rechteckiges, begrenztes Gebiet dar, das wir im Folgenden „Karte“ nennen wollen. Dessen Ausmaße müssen in der Konfigurationsdatei festlegen (diese ist erkenntlich an der Dateiendung „.config“). Zu allererst müssen Sie die Auflösung der Karte angeben. Dazu beschreiben sie in die Konstante `EPM`. Der Name `EPM` steht für „Einheiten pro Meter“. Das heißt, wenn Sie

```
EPM = 200
```

eingeben, wird ein Meter in der Wirklichkeit 200 Einheiten im Modell entsprechen. Im Folgenden werden wir diese Einheit, nur „Einheit“ nennen.
Dann bemessen Sie die Kartenbreite und -höhe so, dass das Areal in der Realität ein Stückchen größer wäre, damit es keinen Konflikt mit dem Modell gibt. Diese schreiben Sie in die gleichnamigen Konstanten in Einheiten.
Also z.B. für eine Karte von 2×2 Metern mit der Auflösung von 200 Einheiten pro Meter:

```
EPM = 200
KARTENBREITE = 400
KARTENHOEHE = 400
```

Kapitel 4

Roboterspezifikation

Es ist auch möglich, ganz einfach per Konfigurationsdatei das Programm auf einen anderen Roboter zuzuschneiden. `ROBOTERNAME` können Sie frei wählen. Er dient lediglich dazu, dass Sie mehrere Roboter verwalten können und immer wissen, welcher zu welchen Daten gehört. Aber Vorsicht: Es darf immer nur ein Roboter mit seinen Merkmalen in der Konfigurationsdatei stehen. Die anderen können Sie aber in anderen Textdateien abspeichern und bei Bedarf wieder dort hineinkopieren.
Der Roboter wird im Modell als rechteckig dargestellt. Da kaum ein Roboter tatsächlich diese Form hat, müssen Sie dasjenige Rechteck in der Konfigurationsdatei angeben, das den Roboter am besten annähert. Ein Um-Rechteck sozusagen.
Die Konstante `ROBOTER_LAENGE` gibt die Länge des Roboters in Einheiten an. Damit ist natürlich das Ausmaß in Richtung des Vorwärtsfahrens gemeint. `ROBOTER_BREITE` gibt das andere Ausmaß an, ebenfalls in Einheiten.
Für ein möglichst genaues Modell wird noch der Drehmittelpunkt des Roboters gebraucht. Führen Sie dazu eine relativ lange Drehung per `RECHTS(250)` aus und beobachten Sie von senkrecht über dem Roboter, welcher Punkt des Roboters sich nicht bewegt. Markieren Sie diesen Punkt und messen sie nun relativ zu der hinteren linken Ecke des Roboters die folgenden beiden Größen: `ROBOTER_MITTE_X`, das ist die Koordinate „in Richtung der Breite“ und `ROBOTER_MITTE_Y`, das ist die Koordinate „in Richtung der Länge“, jeweils angegeben in Einheiten.
Mit dem gleichen gedachten Koordinatensystem, diesmal aber relativ zum gerade ermittelten Drehmittelpunkt geben Sie nun die Position der Berührungssensoren an, jeweils mit der Nummer desjenigen Dateneingangs auf dem RCX-Block, an dem der Sensor angeschlossen ist. Dann geben Sie `SENSOR_1_BREITE` an, ebenfalls in Einheiten. Analog für Sensor 2 und 3. Für die Sensoren fehlt jetzt noch die Sensorrichtung. Damit ist die Richtung gemeint, in die der Sensor „zeigt“. In die Gegenrichtung wird der Sensor nachgeben, wenn der Roboter anstößt. Dies ist wiederum ein Vektor aus zwei Komponenten. Obwohl es ein „Richtungsvektor“ ist, braucht er nicht normiert zu sein. Seine Richtung soll relativ zur Richtung des Roboters angegeben werden. Für einen Sensor, der frontal am Roboter angebracht ist, und am Dateneingang 2 angeschlossen ist, wird man

```
SENSOR_2_RTG_X = 1
```

```
SENSOR_2_RTG_Y = 0
```

eingeben müssen.
Sehr wichtige Daten für die Simulation, aber auch für das Kompilieren des Roboterprogramms sind `METERZEIT` und `PERIODENDAUER`. Beide Konstanten beinhalten eine Formel. Die Syntax einer Formel, die das Programm versteht, ist die Folgende: Man kann alle Grundrechenarten sowie das Potenzieren (mit dem Rechenzeichen „^“) benutzen. Als Operanden kann man einerseits ganze Zahlen benutzen. Um auch gebrochene Zahlen verwenden zu können, bildet man ganz einfach einen Bruch. Andererseits kann man auch Variablen verwenden. Deren Zahl hängt von der Formel ab. In diesen beiden genannten Formeln wird genau ein Parameter erwartet, nämlich die Batteriespannung in mV. Im Allgemeinen kann eine Formel aber mehrere oder gar keinen Parameter erwarten. Diese werden dann durchnummeriert. Die Namen lauten dann „p_0, p_1, p_2...“. In einer Formel kann man alles beliebig klammern, aber „Potenz-vor-Punkt-vor-Strich-Rechnung“ wird schon standardmäßig beachtet.
Zurück zu `METERZEIT` und `PERIODENDAUER`. `METERZEIT` gibt die Zeit in Hundertstelsekunden an, die der Roboter für einen Meter Strecke benötigt, in Abhängigkeit von der Batteriespannung in mV. `PERIODENDAUER` gibt die Zeit in Hundertstelsekunden an, die der Roboter für eine komplette Umdrehung (also 360 bzw. 2π) benötigt, in Abhängigkeit von der Batteriespannung in mV. Hierzu ist zunächst jeweils eine Messreihe durchzuführen, in der bei verschiedenen Batteriespannungen die gefahrenen Strecken (bzw. Winkel) bei fester Fahrdauer zu messen sind. Dabei ist es am leichtesten, sich einen Testbefehl zu schreiben, der den Roboter eine gewisse Zeit nach vorn fahren (bzw. drehen) lässt. Die Strecke (bzw. den Winkel) kann man direkt messen und die aktuelle Batteriespannung wird ja nach allen ausgeführten Befehlen angezeigt. Für das Schreiben von Befehlen lesen Sie bitte das Kapitel 5 „Implementierung von Befehlen“. Die Abhängigkeit der Größen kann man dann beispielsweise durch eine Ausgleichsgerade annähern. Hiermit haben wir im Test gute Erfahrungen gemacht. Es kann hier aber auch jegliche andere Formel in der genannten Syntax angegeben werden. Und auch ein konstanter Wert ist eine Formel, eignet sich also zumindest als Testkandidat.

Kapitel 5

Implementierung von Befehlen

Es ist ganz einfach, neue Befehle einzubinden. Grundvoraussetzung ist natürlich das Beherrschen der vom Programm verwendeten Lego-Programmiersprache. Die von uns implementierte heißt NQC. Eine deutsche Anleitung findet sich z.B. unter http://lug.fh-swf.de/lego/.
Hat man diese Hürde genommen, fehlen nur noch wenige Schritte. Zuerst überlegt man sich einen geeigneten Namen für den Befehl. Dann vergibt man eine Codenummer. Bei beiden Schritten bitte beachten, dass beides noch nicht vergeben ist. Weiter sollte der Bezeichner nicht die Sonderzeichen „#“, „=“ oder „(“ enthalten. Schwierigkeiten mit anderen Sonderzeichen sind nicht bekannt, werden aber nicht grundsätzlich ausgeschlossen. Groß-, Kleinschreibungen und deren Mischformen sind als äquivalent zu betrachten. Und die Codenummer ist unbedingt aus dem Bereich 1-251 zu wählen.
Ein Beispiel für das bis jetzt beschriebene:

```
TANZ = 123
```

Als nächstes sollte man sich überlegen, von wievielen Parametern der Befehl abhängen soll. Diese Zahl trage man unter Hinzunahme des Präfix „DIM_“ (für „Dimension“) ein:

```
DIM_TANZ = 123
```

Dann schreibe man den vorgesehenen Quelltext in die entsprechende Konstante mit Präfix „1_NQC_“. Hierbei kann man alle anderen Befehle, die bereits implementiert sind, mit Präfix „DO_“ aufrufen. Die Parameter können, wie in den angesprochenen Formeln, mit „p_0, p_1“ usw. referenziert werden. Auch muss man beachten, dass man ja einen Quelltext in eine Konfigurationsdatei schreibt, die eine eigene Syntax hat. Daher sind neue Zeilen mit „~’ zu beginnen. Weiterhin müssen die Zeichen „=“ und „#“ mit

```
@@~~~~$$aequumaequum$$^^^^@@
```

und

```
@@~~~~$$cruxcrux$$^^^^@@
```

maskiert werden. Der Quelltext, den man abspeichert, wird sich nach dem Programmstart schon in einer Funktionsumgebung mit Deklaration des Names und aller Parameter befinden. Man muss also nur noch den „Funktionsrumpf“ programmieren.
Wieder zum Beispiel:

```
1_NQC_TANZ = ~ DO_VORWAERTS(p_0); DO_RECHTS(45);
~ DO_VORWAERTS(p_0*(14142 / 10000)/2); DO_RECHTS(90);
~ DO_VORWAERTS(p_0*(14142 / 10000)/2); DO_RECHTS(135);
~ DO_VORWAERTS(p_0); DO_LINKS(135);
~ DO_VORWAERTS(p_0*(14142 / 10000)); DO_LINKS(135);
~ DO_VORWAERTS(p_0); DO_LINKS(135);
~ DO_VORWAERTS(p_0*(14142 / 10000)); DO_LINKS(135);
~ DO_VORWAERTS(p_0);
```

Abschließend muss nur noch durch eine weitere Konstante für die Simulation festgehalten werden, welche Bewegungen bei dem Befehl ausgeführt werden. Diese müssen ja durch das Modell dargestellt werden können, das aber die Bedeutung des Quelltextes nicht kennt.
Diese Information wird unter Verwendung des Präfix „`CMD_`“ abgespeichert, und zwar in einer ähnlichen Form, wie man Befehle im Programmfenster eingibt. Der Unterschied ist, dass man nur die Elementarbefehle `VORWAERTS, RUECKWAERTS, RECHTS` und `LINKS` verwenden kann, deren Bedeutung dem Modell bekannt sind. Allerdings kann man statt konstanten Werten, wie bei der Eingabe ins Programmfenster, hier auch die bereits erklärten Formeln benutzen. Dabei sind dieselben Parameter zu verwenden, wie beim Quelltextes des Befehles.

```
CMD_TANZ = VORWAERTS(p_0) RECHTS(45) VORWAERTS(p_0*(14142 / 10000)/2)
RECHTS(90) VORWAERTS(p_0*(14142 / 10000)/2) RECHTS(135) VORWAERTS(p_0)
LINKS(135) VORWAERTS(p_0*(14142 / 10000)) LINKS(135) VORWAERTS(p_0)
LINKS(135) VORWAERTS(p_0*(14142 / 10000)) LINKS(135) VORWAERTS(p_0)
```

Das waren alle nötigen Eingriffe in die Konfigurationsdatei Jetzt muss nur noch in dem Roboterprogrammgerüst (erkenntlich an der Dateiendung .nqc) irgendwo zwischen den Marken `//DEFINE-ANFANG` und `//DEFINE-ENDE` ein leeres „Define“ eingefügt werden:

```
#define TANZ
```

Damit ist der neue Befehl komplett implementiert und kann wie jeder andere Befehl in der Eingabe im Programmfenster aufgerufen werden.

5.1 Makros

Makros sind keine eigenen Befehle, sondern sind lediglich ein programm-internes Synonym für einen oder mehrere Befehle. Da diese Befehle von Parametern abhängen können, kann auch das Makro Parameter haben. Deren Anzahl ist wieder mit „DIM_“ anzugeben. Zum eigentlichen Inhalt: Ein Makro kann in genau derselben Syntax verfasst werden wie der Eintrag mit „`CMD_`“. Allerdings ist dieser im Falle des Makros mit dem Präfix „`MAKROS_`“ zu versehen:

```
DIM_RECHTECK = 2
MAKRO_RECHTECK = VORWAERTS(p_0) RECHTS(90)
VORWAERTS(p_1) RECHTS(90) VORWAERTS(p_0)
RECHTS(90) VORWAERTS(p_1) RECHTS(90)
```

Neben größeren Makros kann es auch zweckmäßig sein, einzelne Befehle durch ein Makro „Umzubenennen“ So können die Elementarbefehle, die in der Konfigurationsdatei nicht geändert werden dürfen, verkürzt werden, um schneller Befehle aufrufen zu können:

```
DIM_VW = 1
MAKRO_VW = VORWAERTS(p_0)
```