RCCL 5.1.4“HowTo”
Version 1.4

Torsten Scherer*
Technische Fakultät
AG Technische Informatik
Universität Bielefeld

1. Juli 2003

1 Preface - About this HOWTO

Dieses HOWTO soll sowohl den Umgang mit RCCL auf den Rechnern der TechFak (öffentliche Solaris-Maschinen als auch Linux-Maschinen der AG-TI) erläutern, als auch die Installation und den Umgang mit RCCL unter Linux für diejenigen, die es gerne privat zu Hause nutzen möchten.

Es soll **nicht** die Programmierung mit RCCL erläutern – dafür sei auf die entsprechenden Veranstaltungen verwiesen, oder auf den `RCCL User's Guide` in dem `doc`-Unterverzeichnis.

2 Introduction

RCCL ist ein Softwarepaket von Dr. John E. Lloyd et.al. zum steuern und/oder simulieren von Robotern von einer “normalen” Workstation aus. RCCL darf für nicht-kommerzielle Zwecke frei benutzt werden, unter Reglementierungen die in diversen `COPYRIGHT`-Dateien nachzulesen sind. RCCL steht für “Robot Control C-Library”, womit schon mal feststeht was auf einen zukommt, nämlich Programmierung in C. Die meisten Robotersysteme sind Eigenentwicklungen mit nicht nur eigenständiger Hardware, sondern auch Software, so dass es durchaus einen Vorteil (oder zumindest Komfortgewinn) darstellt, das alles mal von C aus auf einer “normalen” Maschine laufen lassen zu können. Soviel zur Motivation von RCCL und den Gedanken, die dahinter stehen.

Die von uns hier im Moment verwendete Version – ich nenne sie 5.1.4– ist eigentlich nur eine Vorabversion von 5.0.0, speziell von John Lloyd für uns für eine Vorlesung von Dr. Jianwei Zhang zur Verfügung gestellt. Wir arbeiten zwar mittlerweile sehr lange erfolgreich damit und ich hoffe auch die schlimmsten Fehler behoben zu haben, aber trotzdem kann sie noch ein paar Macken haben kann. Falls also Teile von RCCL nicht so reagieren wie es die Manuals nahelegen, so kann das durchaus ein Bug sein. Zum anderen ist RCCL sowieso nur *“as is”* zur Benutzung freigegeben, und eine Garantie gibt es für nichts. Daher kann in jedem Fall eine kurze EMail an mich nur nützlich sein, und wenn es nur ist, damit ich ein *feature* als solches bestätigen kann.

RCCL in seiner jetzigen Form ist sowieso laut Aussagen von John Lloyd auf dem aussterbenden Ast. Zitat: „Es ist einfach an der Zeit, das alles mal gründlich aufzuräumen. Und dann kann ich es gleich nochmal neu und besser machen. Objektorientiert vielleicht.“ Das ist auch der Grund weswegen ich mir die Freiheit herausnehme, abweichende Versionsnummern für unsere internen Versionen zu vergeben und meine eigene, komfortablere (wie ich finde) Makefilestruktur über den Sourcebaum drüber zu legen. Die folgenden Abschnitte beschreiben unsere Version 5.1.4, und nicht das Original. An Stellen wo von dem Original abgewichen wird, sollte jeweils ein kleiner Kommentar stehen. Diese Änderungen sind insbesondere zu beachten, wenn diese Hinweise mit denen aus der offiziellen RCCL-Dokumentation verglichen werden.

3 Configuration, Compilation & Installation

Hinweis: Die Teilabschnitte *Configuration* und *Compilation* sind nur für diejenigen interessant, die RCCL selbst auf einem Linux-PC compilieren wollen, der Teilabschnitt *Installation* aber für alle.

* email: itschere@techfak.uni-bielefeld.de

3.1 Configuration

Die normale Konfiguration von RCCL setzte das Setzen einiger Environmentvariablen, das Editieren eines Konfigurationsfiles, ein `xmkmf` und anschließendes `make Makefiles` voraus – also im Prinzip alles, was so rund um `Imakefiles` gehört.

Ich persönlich fand das reichlich kompliziert und hatte in der Vergangenheit auch des öfteren Probleme damit gehabt. Deswegen verwende ich dieses Schema nicht mehr, sondern nur noch das im nächsten Abschnitt beschriebene.

Interessierten an dem Originalablauf seien hiermit auf den *install guide* im `doc`-Verzeichnis verwiesen, wodrin diese ganze Prozedur (und noch viel mehr) beschrieben steht. Die Lektüre dieses *guides* ist ebenfalls unvermeidlich wenn RCCL auf eine neue Architektur portiert werden soll, weil dann noch eine Menge Handarbeit zu verrichten ist. Glücklicherweise hat das bis jetzt noch keiner gewollt...

3.2 A more comfortable Configuration

Mein Ansatz lautet, dass ich die ganzen Konfigurationsmöglichkeiten, die in den `Imakefile`s stecken, auch durch etwas fortgeschrittene Makefile-Programmierung direkt erreichen kann – die `Makefile`s in den allermeisten Verzeichnissen stammen daher von mir. Die Original-`Imakefile`s sind inzwischen unwiderruflich gelöscht...

Eine wichtige **Vorbedingung** für das Funktionieren meiner `Makefile`s ist, das die Environmentvariable `MYARCH` existiert und den Namen der jeweils zu compilierenden Architektur gemäß der Ausgabe des Shellskiptes `config.guess` aus näherungsweise irgendeinem beliebigen, vielleicht nicht allzu altem GNU-Paket enthält. Für eine Auflistung der schon unterstützen Architekturen siehe die entsprechenden `Makevars.$MYARCH` im Hauptdirectory. In diesen Files sind alle Konfigurationsmöglichkeiten enthalten (z.B. die Optimierungsoption `-mcpu=xxx`, die mindestens `gcc-2.8.0` voraussetzt), wenn also jemand lieber mit anderen Optimierungsflags und/oder Debuggingflags compilieren möchte...

3.3 Compilation

Wenn die Konfiguration in Ordnung ist, kann man mit

```
make veryclean
make libs
make bins
```

oder einfach – wenn alles schon *clean* ist – mit

```
make
```

den ganzen Sourcebaum durchcompilieren. Als Ergebnis entstehen in `lib.$MYARCH` die Libraries und in `bin.$MYARCH` die Binaries. Die anderen `lib`- und `bin`-Verzeichnisse stammen von der Original-Prozedur und sind nicht mehr von Bedeutung.

Die einzelnen Kommandos sind auch in den einzelnen Unterverzeichnissen ausführbar, können dort aber z.B. im Falle von `make libs` nur noch die Objektfiles übersetzen, aber nicht mehr die ganze Library neu erstellen.

3.3.1 A Note about Solaris

Es scheint so zu sein, dass der Linker von Solaris nicht die gleichen Parameter versteht wie der Linker von GNU, was dazu führt, das man *shared libraries* nicht so erstellen kann, wie die *info pages* von `gcc` das vorschlagen, und man es z.B. von Linux gewöhnt ist. Daher sind die Solaris Binaries mit den GNU Binutils compiliert. Das wiederum hat zur Folge, das beim Linken der so erstellten Library mit dem Solaris Linker eine Ausgabe „`ld: warning: file /vol/tirccl/lib/librccl.so: section .stabstr: malformed string table, initial or final byte`“ kommt. Dies ist eine harmlose Warnung und ignoriert werden.

3.4 Installation

Ob selber compiliert oder der Distribution entnommen, können die Libraries aus `lib.$MYARCH` und die Binaries aus `bin.$MYARCH` prinzipiell an jede beliebige Stelle kopiert werden, es sollte aber die Environmentvariable `LD_LIBRARY_PATH` existieren und den Librarypfad enthalten, sonst laufen die Programme nicht. Es gibt jetzt nur noch eine Library, `librccl.{a|so}`, und nicht mehr zwei, `librciUser.a` und `librciCtrl.a`, wie früher. Das ist als einer der Unterschiede zur Originalversion zu beachten!

Zusätzlich müssen noch ein paar Konfigurationsdaten in ein beliebiges Verzeichnis kopiert werden, wo sie von RCCL mit einer Environmentvariablen gefunden werden können (siehe unten). Für den reinen Simulatorbetrieb brauchen nur diejenigen Dateien bzw. Verzeichnisse, auf die die Links `conf.rciparams`, `conf.robotData`, `conf.robots` und `conf.robotsim` zeigen, installiert zu werden. Wer will, kann auch noch `conf.defaultRobot` installieren, aber das ist nicht zwangsweise nötig.

Für den Betrieb echter Roboter müssen darüber hinaus noch die Dateien bzw. Verzeichnisse der Links `conf.robots`, `conf.robotData` und `conf.robotDataTab` editiert werden. Wie das zu geschehen hat, ist der Originaldokumentation im `doc`-Verzeichnis zu entnehmen, die ich mit dieser Kurzanleitung keinesfalls als ersetzt betrachtet wissen möchte.

Die mitgelieferten Versionen aller dieser Dateien spiegeln unser ganz spezielles Setup wieder und weichen an einigen Stellen von den in der Originalversion enthaltenen Dateien ab. Sie erlauben aber auf jeden Fall den Simulatorbetrieb ohne weitere Änderungen.

Die Manpages und Includefiles müssen ebenfalls an die gewünschte Stelle kopiert werden, aber hierbei gibt es nichts weiter zu beachten.

Zusammengefasst sieht unsere Installation z.Z. ungefähr so aus:

```
> cd /vol/tirccl ; find
bin
bin/RCISimMuxd
bin/move
bin/robotsim
bin/teachdemo
bin/pumacal
bin/forceCal
bin/free
bin/potcal
bin/power
bin/primecal
bin/zerograv
include/*
lib
lib/librccl.a
lib/librccl.so -> librccl.so.5.1.4
lib/librccl.so.5.1.4
man/*
share
share/robotData
share/robotData/puma*.[jls,kyn,pos]
share/.rciparams
share/.robotsim
share/robotIO.cfg
src/rccl.5.1.4   <you are here :-)>
```

Von dieser Installation stellt diese „Distribution“ nur das Sourceverzeichnis `rccl.5.1.4` dar, der Rest muss alles von Hand so erstellt werden, da es (immer) noch keine richtige Installationsprozedur gibt – ich bitte um Entschuldigung...

3.5 Simulator Configuration

Als letztes muss noch der Port für die Simulatorverbindungen konfiguriert werden. Hierzu muss einmal in die Datei `/etc/services` (oder in die NIS-Datenbank, wenn der Rechner NIS im Netzwerk benutzt) die Zeile

```
RCISimMux       5347/tcp                        # RCI Simulator port
```

eingefügt werden. Falls wider Erwarten diese Portnummer schon belegt sein sollte, so kann auch irgendeine andere genommen werden, allerdings gibt es dann Probleme bei Simulatorverbindungen zwischen unterschiedlichen Rechnern. Das ist aber auch schon alles an Konfigurationsarbeiten.

4 Usage

4.1 Environment Settings

Zum Benutzen von RCCL muss man ein paar Environmentvariablen setzen, was der Einfachheit halber aus der entsprechenden Initialisierungsdatei der jeweiligen Shell heraus geschehen kann/sollte. So z.B. sieht es bei mir für die `tcsh` unter Solaris aus:

```
setenv TIRCCL /vol/tirccl
setenv PATH $PATH':'$TIRCCL'/bin'
setenv MANPATH $MANPATH':'$TIRCCL'/man'
setenv RCCL_LIB_GENERIC $TIRCCL'/lib'         # moper
setenv RCCL_PATH_Sun4sol '.:'$TIRCCL'/lib'    # config files
```

und so für Linux:

```
setenv TIRCCL /vol/tirccl
setenv PATH $PATH':'$TIRCCL'/bin'
setenv MANPATH $MANPATH':'$TIRCCL'/man'
setenv RCCL_LIB_GENERIC $TIRCCL'/share'       # moper
setenv RCCL_PATH_Linux $TIRCCL'/share'        # config files
```

Die Variablen `$PATH` und `$MANPATH` sind hoffendlich selbsterklärend, die Variable `$RCCL_LIB_GENERIC` ist für den reinen Simulatorbetrieb eigentlich überflüssig (hier wird das Steuerprogramm `moper` für die Roboter gesucht), und die Variable `$RCCL_PATH_Sun4sol` bzw. `$RCCL_PATH_Linux` gibt an, wo die Parameterdateien der Roboter gesucht werden.

Für die `bash` sieht die Syntax ein klein wenig anders aus, dort muss man

```
export TIRCCL=/vol/tirccl
```

usw. schreiben. Für die ebenfalls weit verbreitete `rc` muss jeder dieser Schritte in zwei Teile wie in

```
TIRCCL=/vol/tirccl
export TIRCCL
```

zerlegt werden. Im Zweifelsfall einen Blick in die Manpage der Shell werfen!

4.2 Makefiles

Von unserer Version 5.0.0 an ist RCCL so zu behandeln wie beliebige andere Softwarepakete mit externen Libraries auch, man benötigt also nur ein paar Änderungen an den Makefiles (in Version 4.X war das alles um einiges aufwendiger). Die offizielle Dokumentation ist aber immer noch auf dem alten Stand und also nicht mehr korrekt. Insbesondere die Hinweise auf das Compiler-Frontend `rcc` und den magischen Parameter `CTRL` in seiner Kommandozeile sind obsolet – es gibt es nicht mehr.

So könnte z.B. ein Makefile für ein RCCL-Programm namens `example.c` aussehen:

```
#
# Makefile fuer example.c
#

TIRCCL = /vol/tirccl

CC = gcc
CPPFLAGS = -I$(TIRCCL)/include
CFLAGS = -Wall -O2 -g
LDFLAGS = -L$(TIRCCL)/lib -g
LDLIBS = -lrccl -lm          # für Linux
# LDLIBS += -lsocket -lnsl   # zusätzlich für Solaris
# LDLIBS += -lthread         # zusätzlich für echte Roboter unter Solaris
```

```
all: example

clean:
        $(RM) *.o *~

example: example.o
        $(CC) $(LDFLAGS) $< $(LDLIBS) -o $@

example.o: example.c
        $(CC) $(CPPFLAGS) $(CFLAGS) -c $< -o $@
```

Wer das hier mit cut&paste ausschneidet muss aufpassen, dass die Einrückung in der Zeile unter „clean:“ keine 8 Spaces sondern ein `TAB` ist – `make` wird sich sonst beschweren, „missing separator“ oder so.

Der Makefile-*Crack* wird natürlich sofort erkennen, dass man das noch kompakter schreiben kann. Für alle anderen sei auf die bei GNU make mitgelieferte Dokumentation zu Syntax und Semantik von Makefiles unter `/vol/tirccl/share/doc/make-3.76.1.ps` verwiesen.

5 Programming

Neee, das müsst Ihr selber lernen... :-)

6 Literatur

Davon gibt’s natürlich eine ganze Menge, teilweise ganz allgemeine zu Robotersteuerungen ([Paul], siehe Vorlesung), die einem aber dennoch beim Verständnis von RCCL helfen können, weil sich John Lloyd größtenteils sehr an deren Notation gehalten hat. Zum einen wäre da der InstallGuide, der neben dem Setup der Software nochmal die Sache mit dem Environment und den Libraries beschreibt (aber in der alten Version):

```
$RCCL/doc/installGuide/install.ps
```

Des weiteren gibt es das eigentliche Handbuch zu RCCL:

```
$RCCL/doc/rcclGuide/rcclguide.ps
```

Da gibt es dann auf ca. 200 Seiten eine stufenweise Einführung in die Datentypen und Funktionen von RCCL. Die Kapitel 2 und 3 kann man auch “einfach mal so nebenbei” lesen. In Kapitel 4 werden dann ein paar einfache Beispiele erklärt, die man ruhig mal ausprobieren sollte. Danach kann man eigentlich schon loslegen zu programmieren und muss nur mehr ab und an mal etwas spezielles nachschlagen. So jedenfalls ist es mir gegangen. Viel Spaß beim lesen! Es gibt auch eine PostScript-Version der Manpages in:

```
$RCCL/doc/rcclrefman/rcclrefman.ps
$RCCL/doc/rcclrefman/manpages.ps
```

Wer will und kann hat damit eine schöne und schnelle Übersicht über alle Funktionen von RCCL. Mir ist sowas jedenfalls immer noch lieber als endloses suchen in Online-Manpages.

Die Beispielprogramme gibt es zum ausprobieren auch noch unter:

```
$RCCL/src/demo.rccl
```

Viel Spaß!