# **Vorlesung:** Einführung in die Robotik

**Tim Baier**
tbaier@informatik.uni-hamburg.de

**Universität Hamburg**
Fachbereich Informatik
AB Technische Aspekte Multimodaler Systeme

25. November 2004





## Inhaltsverzeichnis







## Die Robot Control C-Library (RCCL)

RCCL in seiner jetzigen Version beruht im wesentlichen auf der Arbeit von John E. Lloyd und Vincent Hayward und wurde von Torsten Scherer (von dem auch diese Folien sind) an der Universität Bielefeld um die Steuerung für den MHI PA10 ergänzt.

Der *RCCL User's Guide* sagt zu RCCL:

*RCCL is a package for implementing task level robot control applications under UNIX. It provides data types useful for robotics applications, such as vectors and spatial coordinate transformations, in combination with routines to specify robot arm motions. Movements can be requested to target positions in either cartesian or joint coordinates, and several arms can be operated from the same program. Arm trajectories are created by a special background task which runs at a fixed rate. Application routines can be defined which modify the trajectories based on real-time sensor inputs.*





## Technischer Stand von RCCL

Leider ist RCCL technisch mittlerweile etwas veraltet und wird nicht mehr weiterentwickelt, aber nichtsdestotrotz ist es ein mächtiges Werkzeug für die Robotik.

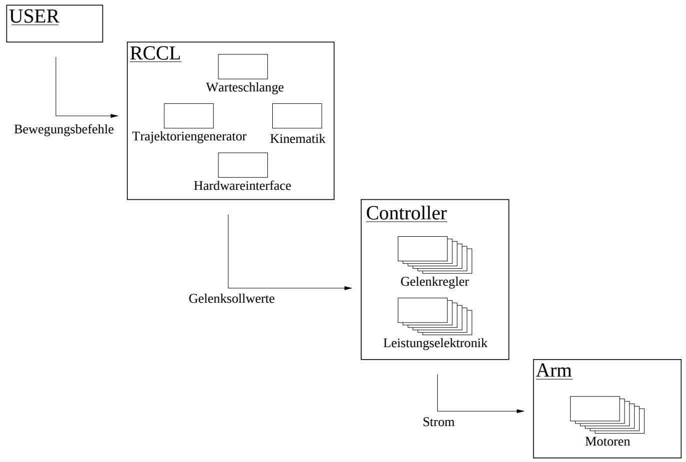
Vorteile von RCCL:

- Einfach über Bibliothek einzubinden
- Einfache Robotersteuerung
- Für alle Roboter (die unterstützt werden) gleiche Handhabung

Nachteile:

- Technisch etwas veraltet
- Es werden (bis auf den PA10) keine aktuellen Manipulatoren unterstützt
- ...

# Struktur von RCCL








# RCCL-Installation bei TAMS

RCCL steht unter

`/local/tams1.2/develop/tirccl/{bin,include,lib,man,share}`

auf allen TAMS Linux-PC's für den Simulatorbetrieb zu Verfügung.

Da auf 98 % der TAMS Rechner Linux läuft gibt es momentan keine RCCL Version für Solaris.





# Installation zu Hause

Es besteht für alle Interessierten die Möglichkeit, sich RCCL auf dem heimischen Linux-PC selber zu installieren bzw. compilieren. Erprobt und erfolgreich sind die Varianten i586-linux und m68k-linux, aber auch alle anderen Linuxe sollten kein Problem sein.

**NEIN,**
es läuft nicht unter DOS, Windows, OS/2 usw.

**NEIN,**
ich halte das auch nicht für machbar.





# Dokumentation

RCCL *itself*:
`/local/tams1.2/develop/tirccl/src/rccl.5.1.4.tar.gz`,

Intro für die Benutzung im Simulatorbetrieb und das *compilieren* unter Linux:
`/local/tams1.2/develop/tirccl/src/rccl/rccl51-howto.ps`

RCCL User's Guide (wichtig!):
`/local/tams1.2/develop/tirccl/src/rccl/doc/rcclGuide/rcclguide.ps`

RCCL Manpages:
`/local/tams1.2/develop/tirccl/man/*`

RCCL Manpages als PostScript:
`/local/tams1.2/develop/tirccl/src/rccl/doc/rcclrefman/rcclrefman.ps`
`/local/tams1.2/develop/tirccl/src/rccl/doc/rcclrefman/manpages.ps`

Ein paar einfache Demo-Programme:
`/local/tams1.2/develop/tirccl/src/rccl.demos/*.c`

# Benutzung von RCCL im Simulatorbetrieb

- siehe
  `/local/tams1.2/develop/tirccl/src/rccl/rccl51-howto.ps`

aber hier eine kleine Zusammenfassung:
Bevor es losgeht, müssen ein paar Environmentvariablen gesetzt werden. Praktischerweise sollte das in der jeweiligen Startupdatei der Loginshell geschehen. Anschliessend lässt sich der Simulator starten.





# Umgebungsvariablen - I

So sieht es z.B. bei mir für Linux-PC's für die `tcsh` (`.tcshrc`) aus:

```
setenv TIRCCL /local/tams1.2/develop/tirccl
setenv PATH $PATH':'$TIRCCL/bin
setenv MANPATH $MANPATH':'$TIRCCL/man
setenv RCCL_LIB_GENERIC $TIRCCL/share
setenv RCCL_PATH_Linux $TIRCCL/share
setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH':'$TIRCCL/lib
```





# Umgebungsvariablen - II

Und für die `bash` (`.bashrc`):

```
export TIRCCL=/local/tams1.2/develop/tirccl
export PATH=$PATH:$TIRCCL/bin
export MANPATH=$MANPATH:$TIRCCL/man
export RCCL_LIB_GENERIC=$TIRCCL/share
export RCCL_PATH_Linux=$TIRCCL/share
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$TIRCCL/lib
```

Ähnlich für alle weiteren shells.





# Bedeutung der Variablen

- `TIRCCL` wird nur als Abkürzung verwendet
- `PATH` wird benötigt, um die RCCL-Programme (`robotsim`, `move`, `teachdemo`, ...) zu finden
- `MANPATH` wird benötigt, damit `man` die Manpages findet
- `RCCL_LIB_GENERIC` und `RCCL_PATH_Sun4sol` braucht RCCL intern um diverse Konfigurationsdaten (z.B. die Roboterbeschreibungen `puma260.{jls,kyn,pos}`) zu finden
- `LD_LIBRARY_PATH` wird benötigt, damit die Programme die *shared library* `librccl.so` finden, da diese in einem Pfad (`/local/tams1.2/develop/tirccl/lib`) steht, in dem der dynamische Linker sie normalerweise nicht sucht.

# Compilieren von RCCL Programmen

Nach dem Setzen der Umgebungsvariablen Programme unter Verwendung des Präprozessorflags
`-I/local/tams1.2/develop/tirccl/include` mit:

```
#include <rccl.h>
```

*compiliert* und unter Verwendung des Linkerflags
`-L/local/tams1.2/develop/tirccl/lib` mit:

```
-lrccl -lsocket -lnsl -lm
```

unter Solaris *gelinkt* werden. Bei Linux reicht:

```
-lrccl -lm
```





# Makefiles

```
CC = gcc
CPPFLAGS = -I/local/tams1.2/develop/tirccl/include
CFLAGS = -Wall -O2
LDFLAGS = -L/local/tams1.2/develop/tirccl/include
LDLIBS = -lrccl -lm -lsocket -lnsl

all: program

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

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

Dokumentation zu der Syntax und Semantik von Makefiles gibt es z.B. im GNU-make Paket, hier von uns zur Verfügung gestellt unter:
`/local/tams1.2/develop/tirccl/share/doc/make-3.80.ps`





# Der Simulator - I

Der Simulator kann mit der Kommandozeile:

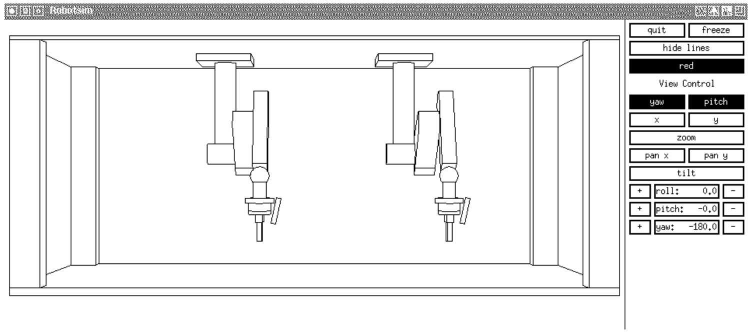
```
robotsim [-s] [-i] [-r <interval>] [-noauto] [-ng] [-X11]
[-bg <background>] [-fg <foreground>] [-wf] [-nocfg] [-f
<configFile>] [-pendant] [ <robotName> [-nc] [-p
<position>] [-clf <CartesianLogFile>] [-jlf <jointLogFile>]
]
```

gestartet werden.





# Der Simulator - II




**Simulanten** sollten in Ihren Programmen grundsätzlich *vor allen anderen* RCCL-Aufrufen die Zeile

`rcclSetOptions (RCCL_SIMULATE);`

stehen haben.

# Offene Probleme

- Signalhandling beim Simulatorbetrieb: Der Simulator benutzt intern `SIGALRM`, darum sind Aufrufe von `sleep()` verboten.
- mehr als ein Roboter in einem Simulator *kann* sich aufhängen, *muß* aber nicht

Ganz allgemein gilt: *Since RCCL/RCI is licensed free of charge, it is provided "as is", without any warranty, including any implied warranty of merchantability or fitness for a particular use.* Zitatende.

Wer also merkwürdiges Verhalten beobachtet und sich sicher ist das nicht selbst verursacht zu haben, kann sich gerne per *email* an mich (`tbaier@`), wenden, oder mich F316 heimsuchen - vielleicht kann ich ja helfen?





# Bewegungstypen in RCCL

Es gibt zwei große Gruppen von Bewegungstypen in RCCL:

1. Gelenkinterpolierte Bewegungen (`movej(), setMod('j')`) und
2. kartesisch interpolierte Bewegungen (`move(), setMod('c')`).





# Gelenkinterpolierte Bewegungen...

...sind technisch relativ einfach, da jedes einzelne Gelenk unabhängig von allen anderen von seinem Ist- zu seinem Zielwert bewegt wird. Das heißt:

- Jede einzelne Gelenktrajektorie ist eindimensional und damit trivial,
- die resultierende Bewegung des Armes als Überlagerung der einzelnen Gelenkbewegungen ist aber sehr unübersichtlich (große "Schlenker")
- und tendiert daher zu Kollisionen mit der Umgebung.

Gelenkinterpolierte Bewegungen sollte man daher nicht benutzen um große Raumabschnitte zu überwinden und sich nicht absolut sicher ist, daß die Zielposition sicher erreicht werden kann.





# Kartesisch interpolierte Bewegungen...

...sind in unserer 3d-Welt gradlinige Bewegung von einem Ist- zu einem Zielpunkt und viel komplizierter:

- Die Bewegungen der einzelnen Gelenke sind über die Kinematik des Roboters gekoppelt und *nicht* mehr eindimensional und
- alle Bedingungen sind in einem anderen Koordinatensystem formuliert als dem, in dem letztendlich geregelt wird.

Mögliche Probleme sind:

- Verlassen des Arbeitsraumes (nach außen und innen)
- Singularitäten (Problemstellen der inversen Kinematik, das *mapping* zwischen Kartesischem und Gelenkraum ist nichtlinear)
- Mehrdeutigkeiten aufgrund unterschiedlicher Konfigurationen

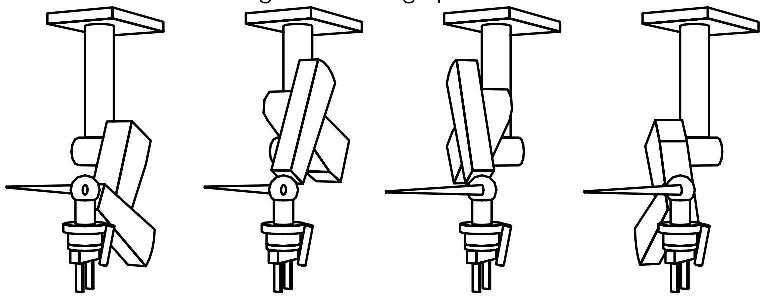
Kartesisch interpolierte Bewegungen sollte man daher nicht benutzen um große Raumabschnitte zu überwinden und sich nicht absolut sicher ist, daß die Zielposition sicher erreicht werden kann. Und damit hätten wir ein Problem...





# Konfigurationen - I

Ein Roboter kann eine bestimmte Position mit unterschiedlichen Gelenkwinkel-Stellungen erreichen. In diesem Zusammenhand wird auch von verschiedenen Roboter-Konfigurationenen gesprochen.



Bestimmte Bewegungen können zu sehr „verrenkten" Stellungen (Stellungen nahe einem Gelenklimit) führen, wo dieselbe Stellung in einer anderen Konfiguration viel entspannter wäre.





# Konfigurationen - II

Da es *keine* Möglichkeit gibt, bei einer *kartesischen* Bewegung die Konfiguration anzugeben, bleibt der Trajektoriengenerator von RCCL zwangsweise immer in derselben Konfiguration wie die Startposition der Bewegung war. Wenn ihn das in eine "verrenkte" Stellung treibt: Pech gehabt! :-(

Einzige Möglichkeit: So oft wie möglich (mit der Aufgabe vereinbar) den Roboter mit gelenkinterpolierten Bewegungen an bekannte „gutartige" Stellungen fahren.





# Trajektoriengenerierung in RCCL - I

Jede Bewegung im kartesischen Raum kann zerlegt werden in:

1. Eine Translation entlang eines Vektors $\vec{p}$ und
2. eine Rotation um einen Winkel $\alpha$ um einen Vektor $\hat{v}$.

RCCL berechnet und benutzt diese Parameter, um mit den Benutzerangaben zu Sollbeschleunigung und -geschwindigkeit die Gesamtdauer der Bewegung, die *motion time* $t_m$ zu berechnen.

RCCL läßt dann während der Bewegung eine normierte Zeit $t = 0..1$ mitlaufen und berechnet in jedem Zyklus die momentane Sollposition über:

$$\begin{aligned} \vec{p}(t) &= t \cdot \vec{p} \\ \alpha(t) &= t \cdot \alpha \end{aligned}$$

# Trajektoriengenerierung in RCCL - II

Eine Positionsgleichung in RCCL gibt aber nur die Ziel*position* an, *nicht* die *Trajektorie* dahin! Man kann eine Rotation um $\hat{z}$ um einen Winkel $\alpha \in [0..2\pi]$ z.B. auf zwei Arten erreichen:

1. Über eine Rotation um $\alpha$ um $\hat{z}$:

$$R_{\hat{z}}(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Über eine Rotation um $2\pi - \alpha$ um $-\hat{z}$:

$$R_{-\hat{z}}(2\pi - \alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$





# Trajektoriengenerierung in RCCL - III

RCCL kann aus so einer Transformation die gewünschte Drehrichtung *nicht* mehr zurück berechnen und wählt die Drehachse statt dessen immer so, daß der erforderliche Rotationswinkel $\alpha < \pi$ ist. Das ist so im Trajektoriengenerator fest eingebaut und nicht zu verändern und kann sehr leicht zu Problemen führen, wenn man sich dessen nicht bewusst ist.

Mögliche Abhilfen:

- Vorgabe mehrerer Hilfspunkte entlang der Drehrichtung (funktioniert nur bedingt)
- Erzeugung der Bewegung mit einem beweglichen Ziel (funktioniert, ist aber komplizierter)





# Bewegliche Ziele in RCCL

Die einzige wirkliche Möglichkeit, Einfluß auf die Trajektoriengestaltung zu nehmen, sind bewegliche Ziele (s. *RCCL User's Guide*, Kapitel 5). Wenn eine Positionsgleichung eine veränderliche Transformation enthält,

- dann löst RCCL die Gleichung nicht a priori komplett auf,
- sondern berechnet zwar noch die Bewegungsparameter,
- reagiert aber in jedem Schritt auf Veränderungen des Zieles.

Damit RCCL die Bewegungsparameter nicht jedesmal neu berechnen muß, bleibt er im Prinzip bei einer gradlinigen Bewegung und führt jede Änderung des Zieles in einem Schritt noch zusätzlich aus. Der *RCCL User's Guide* sagt dazu:

*If a variable transform contained in a target position changes suddenly by a large amount, the robot will try to leap across the workspace, which is a bad thing, because if the robot is large, it might succeed.*





# Beispiele

line.c

box.260.c

box.pa10.c

## The End

Das war's, noch Fragen?