Programming and Controlling
PUMA Robot Arms

1989, 1999, 2005

Gyoung H. Kim, Ph.D.
gyounghkim@open-robotics.com
I. Controlling PUMA Robot Arms with VAL, RCCL, Kali, or ALVIN, 1989.


III. Porting and Running the RCCL on a Real-time Linux/PC, 2005.
Controlling PUMA Robot Arms
With VAL, RCCL, Kali, or ALVIN

1989

Gyoung H. Kim, Ph.D
Contents

• PUMA
  - MK I controller
  - MK II controller
  - MK III controller
  - UNIVAL controller

• RCCL

• Level II

• Kali

• ALVIN
PUMA
PUMA
(Programmable, Universal Machines for Assembly)

• Vendor: Unimation, Inc.

• Arm Types:
  260 Series,
  550 Series, 560 Series, 552 Series, 562 Series
  760 Series

• Controller Types:
  Mark I, Mark II, Mark III, UNIVAL

• Robot Programming Language:
  VAL, VAL II
Unimate PUMA 500 Series, MK I System
MK I PUMA System: Information Flow
MK I Controller: Board Location
MK I PUMA Control System, Block Diagram
Block Diagram of a PUMA 550/560 Robot
MK I PUMA Controller Block Diagram
MK I PUMA Controller Block Diagram
Analog Servo Board, Block Diagram
Unimate PUMA 500 Series, MK III System
MK III PUMA System

ROBOT ARM

INTERCONNECT CABLE

CONTROLLER
CONTROL MODULE
POWER COMPONENT CHAMBER
VENTILATION & COOLING SYSTEM
I/O & CX BOARDS

PERIPHERALS

TEACH PENDANT

VDT & DISK DRIVE
Robot Arm Component Identification
Robot Arm Joint Axes and Ranges of Rotation
Robot Arm: Operating Envelope

Robot shown in righty configuration

- 150 mm radius
- Inaccessible to JTS

Model 562

- 433 mm
- 876 mm radius swept by wrist centerline
- 920 mm radius swept by mounting flange

Inaccessible area can be reached in lefty configuration.
Control Module
Control Card Set Layout
Power Card Set Layout
MK III PUMA Control System Block Diagram
PUMA MK III Controller Configuration

1. control card set

[1] LSI-11/73 CPU board
[2] 64 KW RAM board
[3] quad serial board
[4] parallel I/O board
[5] A interface card
[6] B interface card
[7] digital servo boards
[8] arm signal interface board
2. power card set

[1] C interface board
[2] high power function board
[3] power amplifier boards - PWM type
# Controller Board Comparison

<table>
<thead>
<tr>
<th></th>
<th>MK I</th>
<th>MK III</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU board</td>
<td>LSI-11/23</td>
<td>LSI-11/73</td>
</tr>
<tr>
<td>memory board</td>
<td>32KW RAM board</td>
<td>64KW RAM board</td>
</tr>
<tr>
<td></td>
<td>32KW ROM board</td>
<td></td>
</tr>
<tr>
<td>serial board</td>
<td>quad serial board</td>
<td>quad serial board</td>
</tr>
<tr>
<td>parallel board</td>
<td>(optional)</td>
<td>parallel I/O board</td>
</tr>
<tr>
<td>servo interface</td>
<td>DRV-11</td>
<td>A interface board</td>
</tr>
<tr>
<td></td>
<td>servo interface board</td>
<td>B interface board</td>
</tr>
<tr>
<td>servo boards</td>
<td>6 digital servo boards</td>
<td>6 digital servo boards</td>
</tr>
<tr>
<td></td>
<td>6 analog servo boards</td>
<td></td>
</tr>
<tr>
<td>etc</td>
<td>clock/terminator board</td>
<td>C interface board</td>
</tr>
<tr>
<td></td>
<td>paddle board</td>
<td>high power board</td>
</tr>
<tr>
<td></td>
<td>arm cable board</td>
<td>arm signal interface bd</td>
</tr>
<tr>
<td>power amplifier</td>
<td>linear amp</td>
<td>PWM amp</td>
</tr>
</tbody>
</table>
## Arm Specification Comparison

<table>
<thead>
<tr>
<th></th>
<th>arm</th>
<th>PUMA560</th>
<th>PUMA562</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>specification</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total inertia at motor (in-lb-s-s)</td>
<td>joint 1</td>
<td>0.149190e-01</td>
<td>0.182490e-01</td>
</tr>
<tr>
<td></td>
<td>joint 2</td>
<td>0.605801e-02</td>
<td>0.415921e-01</td>
</tr>
<tr>
<td></td>
<td>joint 3</td>
<td>0.519241e-02</td>
<td>0.663468e-02</td>
</tr>
<tr>
<td></td>
<td>joint 4</td>
<td>0.285927e-03</td>
<td>0.322934e-03</td>
</tr>
<tr>
<td></td>
<td>joint 5</td>
<td>0.300448e-03</td>
<td>0.341809e-03</td>
</tr>
<tr>
<td></td>
<td>joint 6</td>
<td>0.173137e-03</td>
<td>0.176327e-03</td>
</tr>
<tr>
<td>maximum amplifier current (amp)</td>
<td>joint 1</td>
<td>8.0</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>joint 2</td>
<td>8.0</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>joint 3</td>
<td>8.0</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>joint 4</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td>joint 5</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td>joint 6</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td>joint angular acceleration (rad/sec/sec) @SP 100</td>
<td>joint 1</td>
<td>12.82</td>
<td>8.517</td>
</tr>
<tr>
<td></td>
<td>joint 2</td>
<td>8.44</td>
<td>5.568</td>
</tr>
<tr>
<td></td>
<td>joint 3</td>
<td>38.07</td>
<td>12.6880</td>
</tr>
<tr>
<td></td>
<td>joint 4</td>
<td>71.02</td>
<td>23.6670</td>
</tr>
<tr>
<td></td>
<td>joint 5</td>
<td>37.62</td>
<td>25.0800</td>
</tr>
<tr>
<td></td>
<td>joint 6</td>
<td>142.03</td>
<td>23.6670</td>
</tr>
<tr>
<td>joint angular velocity (rad/sec) @SP 100</td>
<td>joint 1</td>
<td>1.43</td>
<td>2.3148</td>
</tr>
<tr>
<td></td>
<td>joint 2</td>
<td>0.925</td>
<td>2.3148</td>
</tr>
<tr>
<td></td>
<td>joint 3</td>
<td>2.132</td>
<td>2.3095</td>
</tr>
<tr>
<td></td>
<td>joint 4</td>
<td>3.977</td>
<td>5.2360</td>
</tr>
<tr>
<td></td>
<td>joint 5</td>
<td>4.214</td>
<td>5.2360</td>
</tr>
<tr>
<td></td>
<td>joint 6</td>
<td>3.977</td>
<td>5.2360</td>
</tr>
<tr>
<td><strong>payload (kg)</strong></td>
<td></td>
<td>2.5</td>
<td>4.0</td>
</tr>
</tbody>
</table>
UNIVAL
UNIVAL Block Diagram

- Torque Processor Board
- Servo Control Module
- Expansion Memory (Optional)
- Arm Interface Board
- Distribution Board
- Major Power Amplifier
- Minor Power Amplifier
- Signal Connection Board
- Robot Arm

Connections:
- CX Module
- VDT
- Disk Drive
- Aux. Port
- Smart I/O
- Teach Pendant
Typical Controller with Analog Torque Loop

N AXES Digital Servo and Robot Controller

Torque Loop / Amplifier

DAC

M

E

N Axes
UNIVAL Torque Processor Loop
UNIVAL Signal Loops

- Command from VAL
- Position and Velocity Servos
- Position Feedback upon Startup
- Current Feedback
- Feedback from Robot
- Arm Cable
- PWM
- AMPS
- Encoders
- Potentiometers
- ROBOT
UNIVAL Servo System Block Diagram
PUMA Control with VAL

1. The hardware does not provide sufficient processing power --- it is too slow for real-time control.

2. The system is not designed to communicate with external computer in a flexible way. The I/O module is only able to provide control signals and process elementary information received from sensors.

3. Memory resources may not be sufficient for large programs.

4. Communication with other computers is limited to interactions with VAL.

5. Inverse kinematics software is not available, hence, trajectories cannot be defined off-line.

6. Control of the arm is performed at the joint level, providing only joint position regulation.
Advantages:

1. The ability to implement closed-loop control of the manipulator in both task and joint configuration spaces.

2. The ability to use the more powerful processor for inverse kinematics, for trajectory planning, and for control in real-time.

3. The ability to test advanced mathematical models for dynamics and real-time control.

4. The ability to connect sensory devices through serial, parallel or bus interface.

5. The ability to create a new control language which includes commands not available in VAL.

6. The ability to access to a database.
Alternative Hardware Configurations

1. The PUMA's Arm Interface Board is disconnected from the DRV-11 card and connected to the parallel interface card in the external machine (only for MK I controllers).

2. Both computers are connected through a standard DEC parallel interface, which offers higher data rates than those of a serial interface.

3. A custom-built interface for bus-to-bus connection can be used. This interface contains a FIFO (First In First Out) hardware buffer.

4. Serial interface through the DLV-11J serial card in the PUMA controller is used.
RCCL
RCCL (Robot Control "C" Library)

- RCCL, Purdue (1983)
- RCCL, McGill (1985)
- RCCL, McGill (1986)
- RCCL, JPL (1986)
- Multi-RCCL, JPL (1990)
- RWRCCL, RWU (2001)

Siblings:
- Level II, Cybotech (1987)
- Kali, McGill (1988)
- ALVIN, Purdue

Contributors:
- RCCL (Robot Control "C" Library)
Physical Implementation of RCCL

- VAX COMPUTER
  - RCCL PROGRAM
  - HIGH SPEED PARALLEL LINK
  - SERIAL LINE FOR LOADING I/O CONTROL PROGRAM

- LSI 11 CPU
  - COMMUNICATION I/O CONTROLLER

- TEACH PENDANT
- A/D CONVERTER

- 6503 JOINT CONTROLLERS

- UNIMATION CONTROLLER

- TO ROBOT
RCCL Block Diagram (Purdue)
RCCL Block Diagram
NORMAL TASK AND PROCESS EXECUTION

SWITCH TO CONTROL PROGRAM MEMORY CONTEXT

EXECUTE CONTROL: CYCLE
- read data from LSI 11
- call RCI check routine
- write commands to LSI 11
- call control function

RESTORE MEMORY CONTEXT

NORMAL TASK AND PROCESS EXECUTION

DRIVER INTERRUPT HANDLER TAKES CONTROL

INTERRUPT HANDLER RETURNS
Execution Cycle at the Control Level

- DATA FROM LSI 11
- COMMANDS TO LSI 11
- COMMAND CHECKING
- INTERNAL CHECKING
- WAIT FOR NEXT CYCLE
- NORMAL EXECUTION
- CONTROL FUNCTION
- RELEASE DIRECTIVE OR ERROR CONDITION
- OPEN DIRECTIVE
- CLOSE DIRECTIVE
- IDLE
- CONTROL DIRECTIVE
RCCL Position Equations

Position Representation:

\[ p_i = Z \cdot T_6 \cdot E \]

where

- \( Z \): the base transform
- \( T_6 \): the manipulator transform
- \( E \): the end-effector transform

Task Position Transform Graphs:
Position Equations

At point $p_1$,

\[
\begin{align*}
p_1 & = Z \ T_{61} \ E_1 \\
T_{61} & = Z^{-1} \ p_1 \ E_1^{-1} \\
& = R_1 \ p_1 \ T_1
\end{align*}
\]

At point $p_2$,

\[
\begin{align*}
p_1 & = Z \ T_{62} \ E_2 \\
T_{61} & = Z^{-1} \ p_2 \ E_2^{-1} \\
& = R_2 \ p_2 \ T_2
\end{align*}
\]

Let $^2p_1 = R_2^{-1} \ R_1 \ p_1 \ E_1 \ E_2^{-1}$

From point $p_1$ to point $p_2$,

\[
T_6 = R_2 \ ^2p_1 \ \text{Drive(s)} \ T_2
\]

where \( \text{Drive}(0) = \text{Identity} \)

\[ \text{Drive}(1) = ^2p_1^{-1} \ p_2 \]

Generally,

\[
T_6 = R \ P \ \text{Drive} \ \text{Comply} \ T
\]
The Four Levels of the Robot Programming Environment

- Device Driver
- Level II Library
- Application Program
- Menu Interface
Hardware Configuration of RC-X Controller
Kali
**Kali**

*(A creature with many arms in Hind mythology)*

Characteristics:

1. Programming and control of multiple manipulators operating in close cooperation.

2. Hybrid force/velocity task space with dynamic compensation.

3. Selection of the largest amount of off-the-shelf computing technology.

4. Open architecture design.
High Performance Computing System

Current trends:

1. Super or mini-super computers.

2. Workstations with special very high performance floating point accelerations.

3. Special purpose chips such as DSPs (AT&T, TI., etc).

4. Many single board computers operating in parallel.
Kali src Directory
Kali Block Diagram

- SUN 3 workstation
- ethernet link
- ethernet board
- CPU 1: 68020
- CPU 2: 68020
- CPU 3: 68020
- CPU 4: 68020
- CPU 5: 68020
- memory board
- VME BUS
- DAC board
- robot I/O board
- parallel I/O board
- force/torque sensor
- encoder pulses
- brake release
- power on/off
- power amplifier
- PUMA 560 ROBOT ARM
Run-Time Structure

1. Synchronous processes
   i. Trajectory generator
   ii. Servo control
   iii. I/O process

2. Asynchronous process
   i. User process
   ii. Dynamic computations

Processor Communication

1. Message passing
2. Shared memory
1. Software support

Host: Unix-based workstation
Target: real-time operating system (VxWorks)
Target-host communication - ethernet

2. Hardware support

Servo CPUs:
1 ms servo rate PID control algorithm
One CPU for three joints (CPU3 and CPU4)

Computational CPUs:
I. User program, trajectory generator, kinematics (CPU1)
ii. Dynamic computation (CPU2)
iii. Supervisor - all I/O information handling (CPU0)
Kali Graphic Simulator
Kali Graphic Simulator
Kali Graphic Simulator
Spatial Relationships
(Ring Structures)

$B \cdot M \cdot T \cdot A \cdot D \cdot C = \text{Identity}$

where

$B$ : the Manipulator base transform

$M$ : the manipulator transform

$T$ : the tool transform

$A$ : the accommodation transform

$D$ : the drive transform

$C$ : the goal position of the control frame
Two Arms Manipulating a Common Object

Transformation Graph for Two Arms Manipulating a Common Object
Proposed Kali Configuration for Low Level of JPL Telebot
ALVIN
ALVIN
(ALmost Very Intelligent, but Not)

- Purdue's new robot system
- Named by Prof. C S G Lee
- Advanced system in hardware and software
# System Comparison

<table>
<thead>
<tr>
<th></th>
<th>Condore</th>
<th>Chimera</th>
<th>Kali</th>
<th>Alvin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Univ.</td>
<td>MIT</td>
<td>CMU</td>
<td>McGill</td>
<td>Purdue</td>
</tr>
<tr>
<td>Host</td>
<td>SUN 3</td>
<td>Sun 3</td>
<td>SUN 3</td>
<td>SUN 3</td>
</tr>
<tr>
<td>Host Interface</td>
<td>bus adaptor</td>
<td>SUN backplane</td>
<td>ethernet</td>
<td>ethernet</td>
</tr>
<tr>
<td>role of the host</td>
<td></td>
<td>development</td>
<td>environments</td>
<td></td>
</tr>
<tr>
<td>interprocess or communicat ion</td>
<td>dual-port memory and mailbox interrupt</td>
<td>VSB</td>
<td>VSB</td>
<td></td>
</tr>
<tr>
<td>target CPU</td>
<td>68020</td>
<td>68020, 20MHz</td>
<td>68020, 20MHz</td>
<td>68030, 33MHz</td>
</tr>
<tr>
<td>No. of CPUs</td>
<td>4</td>
<td>1 or more</td>
<td>5 or more</td>
<td>5 or more</td>
</tr>
<tr>
<td>math Processor</td>
<td>68881</td>
<td>68882</td>
<td>68881</td>
<td>68882</td>
</tr>
<tr>
<td>CPU board</td>
<td>Ironics</td>
<td>Ironics IV-3204</td>
<td>Heurikon HKV2F</td>
<td>Heurikon HKV3E</td>
</tr>
<tr>
<td>Target arm</td>
<td>Utah-MIT hand MIT DD Arm</td>
<td>CMU DD Arm</td>
<td>PUMA 560</td>
<td>PUMA 562</td>
</tr>
<tr>
<td>servo rate</td>
<td>400 Hz</td>
<td>N/A</td>
<td>1 KHz</td>
<td>1 KHz(?)</td>
</tr>
<tr>
<td>kernel</td>
<td>MIT’s own</td>
<td>CMU’s own</td>
<td>VRTX-32</td>
<td>WRS kernel</td>
</tr>
<tr>
<td>OS</td>
<td>SUN OS</td>
<td>SUN OS</td>
<td>VxWorks V3.2</td>
<td>VxWorks V4.0</td>
</tr>
</tbody>
</table>
### PUMA Arms and Controllers

<table>
<thead>
<tr>
<th>Controller</th>
<th>Arm</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unimation MK I</td>
<td>PUMA 560</td>
<td>POTR B20</td>
</tr>
<tr>
<td>Unimation MK II</td>
<td>PUMA 260</td>
<td>EE Teaching Lab</td>
</tr>
<tr>
<td>Unimation MK III</td>
<td>PUMA 562</td>
<td>MGL 1336</td>
</tr>
</tbody>
</table>

### RCCL (Purdue) and Kali (McGill)

<table>
<thead>
<tr>
<th></th>
<th>RCCL</th>
<th>Kali</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Processor</td>
<td>VAX 780</td>
<td>MC68020</td>
</tr>
<tr>
<td>Communication Processor</td>
<td>LSI-11/23</td>
<td>MC68020</td>
</tr>
<tr>
<td>Servo Control Processor</td>
<td>six 6503's</td>
<td>MC68020</td>
</tr>
<tr>
<td>Power AMP</td>
<td>Unimation Linear</td>
<td>Contraves Goertz PWM</td>
</tr>
</tbody>
</table>
ALVIN Block Diagram (Stage I)
ALVIN Block Diagram (Stage II)

Engineering Computer Network

SUN workstation

VMEbus System
- Ethernet Card
- CPU 0 (68040)
- CPU 1 (68040)
- CPU 2 (68040)
- CPU 3 (68030)
- CPU 4 (68030)
- Shared Memory
- VMEbus Adaptor
- VMEbus Adaptor

Unimation MK III Controller
- Qbus Adaptor
- ADC
- A-interface
- B-interface
- Digital Servo
- Paddle
- Arm Cable

PUMA 562 Robot Arm

Unimation MK I Controller
- Qbus Adaptor
- ADC
- DRV-11
- Arm Interface
- Digital Servo
- Clock / term.
- Analog Servo
- Paddle
- Arm Cable

PUMA 560 Robot Arm
Digital servo boards

Power amplifier

Servomotors

Potentiometers

Encoders

Arm signal board

High power function bd.

Quad serial board

CPU board

I/O interface

Operator panel

User connect

Disk driver

Teach pendant

Aux.

Super.

Alter

Digimig

Spare

CMOS memory board

I/O PCA

CX PCA

PUMA MK III CONTROLLER
PUMA MK III Controller
Modified for ALVIN I
PUMA MK III Controller
Modified for ALVIN II
ALVIN Block Diagram (Stage III)
References

VME bus / VSB

VxWorks, VRTX
[vrtx 3] S. J. Doyle and P. Bunce, "Real-Time Multiprocessing Requirements,“ WESCON '86
PUMA/VAL

[puma 1] B. Fisher and V. Hayward, "Communication Routines Between the LSI 11-03 and the 6503's", Purdue University, (undated, unpublished)

[puma 2] B. Fisher and V. Hayward, "Robot Controller," Purdue University, (undated, unpublished)

(undated, unpublished, company confidential)

[puma 4] R. Vistnes, "Breaking Away From VAL, or How to use your PUMA without using VAL,“ Stanford University, (undated, unpublished)


Unimation Inc., (company confidential)


Part 1 - Control from the System terminal
Part 2 - Communication with a Supervisory System
Part 3 - Real-Time Path Control


IEEE Intl Conf on Robotics, Mar. 1984


LEVEL II

RCCL (a Robot Control C Library)
School of Electrical Engineering, Purdue Univ., Oct. 1983.
[recl 2] V. Hayward, Introduction to RCCL: A Robot Control 'C' Library, TR-EE 83-43,
School of Electrical Engineering, Purdue Univ., Oct. 1983.
School of Electrical Engineering, Purdue Univ., Oct. 1983.
[recl 4] V. Hayward, RCCL Version 1.0 and Related Software Source Code, TR-EE 83-47,
School of Electrical Engineering, Purdue Univ., Oct. 1983.
[recl 5] J. Roger, "VAX-LSI Interprocessor FIFO", Purdue University, (undated, unpublished)

Kali
V. Hayward, L. Daneshmend, A. Nilakantan, and A. Topper, A Selection of Papers"
End of Part 1
RCCL Variants

1999

Gyoung H. Kim, Ph.D
RCCL-related Systems

- RCCL v1.0-based
- Multi-RCCL
- RWRCCL
- Qrobot
- ARCL
RCCL v1.0
Configurations
RCCL (Purdue, 1983)
RCCL Architecture

- User process
  - World modelling
  - Task planning
- Trajectory generator
  - Get request
  - Calculate setpoints
- Realtime control
  - Decode setpoints
  - Collect arm state
- Global variables
  - Motion queue
  - Motion specification
- Clock interrupt
- Unimation Controller
- Robot Arm
Routine Explanation

(1) main(): The main program.
(2) startup():
   Connect "setpoint_n()" to the clock interrupt.
   Initialize and check hardware connection.
(3) move(park):
   "park" is a built-in position. Usually the robot starts from this position.
(4) pumatask():
   The actual user process. Users write only this routine.
(5) release(): Stop the trajectory generator.
(6) setpoint(): The trajectory generator.
(7) jnsend_n():
   Send the setpoints to the robot controller in the global variable "chg".
(8) getobsj_n():
   Get the actual robot position from the global variable "how".
(9) getobst_n():
   Get the arm currents from the ADC from the global variable "how".
ALVIN-RCCL (Purdue, 1991)

68030s/VxWorks

VMEBus

CPU #1
CPU #2
Shared Mem.
Bus adapter

CRT

CLOCK

DRV-11

SERVO INTERFACE

Digital Servo

Analog Servo

Power AMP

Arm Cable

encoder pulses

current

PUMA 560 ARM
RCCL-ALVIN (Purdue, 1998)

Host PC/QNX

PCI Bus

Bus adapter

Qbus

Bus adapter

CLOCK

DRV-11

SERVO INTERFACE

Digital Servo

Analog Servo

Power AMP

Arm Cable

encoder pulses
current

Unimation Controller

PUMA 560 ARM
Software Architecture of ALVIN-RCCL
RCCL / RCI (Standard)
Multi-RCCL

• Multi-RCCL v5.0
  John Lloyd and Vincent Hayward, 1992.
  Official release of Multi-RCCL.

• Multi-RCCL v5.1
  Simulation mode on Linux.

• Multi-RCCL v5.1.4
  Torsten Scherer, 1999.
  Unofficial modifications for Linux.
Hardware Architecture of Multi-RCCL
• **RWRCCCL (Roger Williams RCCL):**
  RCCL modified by Matthew Stein at Roger Williams University in 2000.

• **Configuration**
  – **ARM:** a single puma560 only.
  – **Hardware:** PC + Trident TRC 004/006 boards.
  – **Software:** RCCL ported to rtlinux.  
    Position control is added to RCCL.
RWRCCL (TRC Boards)

Host PC/Linux

ISA bus

TRC004 board

Unimation Controller

TRC006 Board

Power amplifier

Arm Cable

PUMA ARM
Hardware Architecture of RWRCCL
Software Architecture of RWRCCL
Graphic Simulator of RWRCCCL
QRobot

- A multitasking QNX/PC-based robot control system.
- Developed at Clemson University in 1998.
- Target arm: PUMA 560.
- Hardware: PC + MultiQ board + Unimation controller.
RCCL-QRobot (MultiQ Board)

Host PC

PCI bus

MultiQ board

Unimation Controller

Preamplifiers
filtering circuits

Power amplifier

Arm Cable

PUMA ARM
ARCL
(Advanced Robot Control Language)

• Developed by Peter I. Corke, 1993.

• Inspired by an early version of RCCL.

• Monitor and Interpreter to run VAL-II programs.
Structure of ARCL
End of Part 2
Porting and Running the RCCL on a Real-time Linux/PC

2005

Gyoung H. Kim, Ph.D
RCCL Poring

- Porting RCCL v1.0 to Linux.
- Developing a graphic simulator for RCCL v1.0.
- Porting RCCL v1.0 to Linux/RTAI.
- Porting RCCL v1.0 to Linux/Xenomai.
- Patching Multi-RCCL for a newer Linux.
- Patching RWRCCL for a newer Linux.
- Porting RWRCCL to Linux/RTAI.
Test Environment

- RCCL: v1.0 (1983)
- Simderella: v2.0.2 (1995)
- gcc: 2.9.x, 3.0.4, 3.2
- Linux distribution:
  - Redhat 7.3, Redhat 8.0
  - Debian woody, Debian sarge
- CPU: Pentium III, Pentium 4, VIA C3
- RTAI
  - Linux kernel: 2.4.19 (only for RTAI)
  - RTAI distribution: 2.4.11
  - RTAI modules used by RCCL:
    rtai, rtai_sched, rtai_fifo, rtai_shm, rtai_libm
RCCL v1.0 on Linux
Running Modes of RCCL

(Modes are specified at h/switch.h)

1. PLAN mode (no signal or interrupt)
   - setpoint_n() is repeatedly called until completed = 0.

2. FAKE mode (signal – simulated interrupt)
   - At every 28ms, clock() generate a signal.
   - Signal handler calls setpoint_n().

3. REAL mode (hardware interrupt)
   - At every 28ms, a hardware interrupt signal is generated from
     the Unimation controller.
   - Interrupt service routine calls setpoint_n().
Running RCCL on Linux
(BSD mode with lib5)

- RCCL v1.0 written in pre-ANSI C run on BSD Unix.
- While older Linux distributions with lib5 were BSD-flavored, recent Linux distributions with lib6 are SYSV-flavored.
- To port RCCL v1.0 to Linux,
  1. Install BSD headers and libraries.
     On debian, apt-get install altgcc
  2. Do the following modifications:
     - Change <signal.h> and <sgtty.h> to <bsd/signal.h> and <bsd/sgtty.h>, respectively.
     - Implement nap( ) with usleep( ).
     - Insert more FAKE and REAL mode switches for cleaner compiling.
     - Correct up some K & C C-language syntax.
     - Remove malloc_l( ) and free_l( ) routines.
  3. Link the BSD library with the option, -lbsd.
Running RCCL on Linux
(SYSV mode with libc6)

(1) Change from BSD stuffs to SYSV
   - sgtty to termio
   - nap( ) to usleep( )

(2) Change pre-ANSI C stuffs to gcc
   - const( ) to rccl_const( )

(3) Change clock( ) location
    - In RCCL v1.0, separated clock.c and vfork( ) were used.
    - Add clock( ) to main.c and replace vfork( ) with fork( ).

(4) Do the following minor modifications:
    - Add print.c for more printouts.
    - Add printst( ) to jnsend_n( ) of misc.c
    - Create linux directory for examples.
    - Move main.c from src to linux directory.
    - Adjust the delay parameter of usleep( ).
    - Add FAKE_DEBUG for better debugging.
Simderella
(A general-purpose robot simulator for kinematics)

Current joint angles

Simderella

conel
(robot controller)

Desired joint angles, velocities, accelerations

simmel
(forward kinematics calculator)

Homogeneous matrices of robot links

bemmel
(X-windows drawing program)

Graphic visualization
RCCL + Simderella (on-line)
(Simderella as a graphic simulator of RCCL)

For on-line simulation,
1. Modify connel/main.c so that desired joint angles are from \( j_6 \), a global variable of RCCL.
2. Combine the main.c of RCCL and the main.c of connel. After setpoint_n( ), call user_move_robot( ) at each clock signal.
3. Modify user_move_robot( ) and move_robot( ) of connel/moving.c to bypass some dynamics and kinematics calculations.
5. Link RCCL libraries when compiling connel/main.c.
6. Run Simderella.
Right window: connel’s window printing out joint angles from a RCCL program.
Left window: bemmel’s window drawing a PUMA arm with the joint angles.
For off-line simulation,
1. Modify connel/main.c so that desired joint angles are read from an external file, @.out and they are fed into user_move_robot( ).
2. Modify user_mode_robot( ) and move_robot( ) of connel/moving.c to bypass some dynamics and kinematics calculations.
3. Adjust D-H parameters of PUMA arms.
5. Run a RCCL program with “-d” option.
6. Copy @.out to connel directory.
7. Run Simderella.

RCCL + Simderella (off-line)
RCCL v1.0 on Linux/RTAI
Porting RCCL to RTAI

Software Architecture

Kernel space

RT Task #0 (user process)

FIFO handler

RT Task #1 (TG + RTC + moper)

User space

User command monitor

Shared Memory

Data sent to Unimation controller

Completed

FIFO

FIFO

FIFO

FIFO
Software Components

1. User-space task
   (1) User command monitor:
   - Send start/stop/break/resume command to RT tasks in the kernel space.
   - Receive fifo messages from the RT tasks
   - Print out the content of the shared memory.

2. Kernel-space tasks
   (1) RT task #1: the main.c of RCCL (non-periodic)
   (2) RT task #2: the TG + RTC + moper of RCCL (periodic task)
   (3) fifo handler:
   - Send commands to the RT tasks.
   - Receive commands from User command monitor

3. Shared memory: Store all the output data from RT tasks.
RTAI Porting Procedure

1. Kernel memory allocation
   
   Change malloc() and free() to kmalloc() and kfree(), respectively.

2. File output/stdout/stderr
   
   Make shm_printf() and redirect all the file output/stdout/stderr to the shared memory.

3. Modify kernel vprintf() to handle float variables.

4. Replacement of glibc
   
   (1) sprintf – kernel’s lib.a
   (2) strcat, strcpy, strlen - #include <rtai.h>
   (3) math library – rtai_libm.o module
   (4) others – gcc’s bootstrap libgcc.a

5. Follow the compiling/linking options of RTAI.

6. Distribution- or CPU-specific options:
   
   (1) Redhat 7.3: use kgcc without -mpreferred-stack-boundary=2.
   (2) Redhat 8.0: use gcc with -mpreferred-stack-boundary=2.
   (3) VIA EPIA 800 board: use –m586
RTAI Running Procedure

1. Install rtai modules
   modprobe rtai
   modprobe rtai_sched
   modprobe rtai_fifos
   modprobe rtai_shm
   modprobe rtai_libm

2. Install the RCCL realtiime module
   insmod rccl.o

3. Start the “user command monitor” as
   ./rccl_app

4. Type s/q/b/r keys at the prompt of “rccl_app” to
   start/quit/break/resume realtime RCCL tasks.

5. After the RCCL tasks are finished, joint angles and other
   information are saved to file “@.out”
RCCL Porting Results

- **RCCL on BSD- and SYSV-mode Linux**
  - PLAN/FAKE/REAL modes can be compiled.
  - PLAN and FAKE modes run correctly.
  - Clock intervals less than 1 ms work.
- **RCCL + Simderella in on-line mode**
  - FAKE mode with 30ms clock interval work.
  - Faster than 30ms is impossible due to the slow socket communication of Simderella.
- **RCCL on RTAI**
  - Clock rate 50Hz (20ms) works without any problem.
  - All the examples which do not need a real arm can be compiled and run correctly.
  - A faster clock rate is possible when moper functions are needed.
  - For the Unination controller still with the digital servo boards, clock interrupt signals should be received from the Unimation controller to prevent clock drifts.
RWRCCL on Linux/RTAI
Analysis Results of RWRCCL

• Routines are fully documented.

• Trace and analysis of the program execution flow are done.

• Documentation on the program execution flow is almost done.
Updating RWRCCL to a newer RTLinux

• Environment:
  - debian sarge 3.1r1
  - gcc 3.3 6
  - glibc 2.3.2

• RTLinux:
  - rtlinux-3.2-rc1
  - kernel 2.4.27

• Patches
  (1) gcc-dependent: newline in a literal string, sys_errlist, varargs.h
  (2) imake configuration bug.
Porting Results of RWRCCL-rtlinux

• Trajectory generation sampling interval: 27ms

• Servo sampling interval: 0.9 ms
  (position control mode, PID control)

• Simulation mode needs a slower sampling rate due to the slow X11 graphics.

• Dry-running mode in real-time is added.
Porting RWRCCL to RTAI

- RTAI:
  - rtai-3.2
  - kernel 2.4.27

- Patches
  (1) rtlinux-dependent directories:
      $RCCL/jls (kernel modules)
      $RCCL/puma (user-space routines)
  
  (2) Utilities for the puma interface card:
      $RCCL/rtlinux

  (3) Conflicting macro definition:
      FREE(x) is changed to RCCL_FREE(x)
      (FREE(x) is also defined at rtai_lxrt.h)
Ongoing Projects

- Modifying RCCL for running various arms.
- Porting RCCL v1.0 to Xenomai.
- Developing a front-end interpreter for RCCL.
- Modifying RCCL v1.0 for running multiple arms in independent and/or coordinative modes.
- Running RCCL on Non-X86 CPUs.
End of Part 3