Object-Oriented Techniques in Robot Manipulator Control Software Development

2001 American Control Conference

M. Loffler, D. Dawson, E. Zergeroglu, N. Costescu

Department of Electrical and Computer Engineering, Clemson University
Outline

• Introduction
  – Robotic Control Systems at Clemson’s CRB Group
  – Review of Robot Control Platforms
  – Focus of this Research

• The QMotor Robotic Toolkit (RTK)
  – Overview
  – Object-Oriented Design
  – Run-Time Issues
  – QMotor
  – Class Design
  – GUI Components

• Conclusions

• Future Research
Introduction

• Robotic Control Systems at Clemson’s CRB Group
• Review of Robot Control Platforms
• Focus of this Research
Introduction - Robotic Control Systems at CRB – towards Flexibility

Before 1997
- RCCL Mark II Controller
- UNIX/workstation based
- Procedural
- No servo control flexibility
- Uses Puma 560

1997
- RCCL Servo Control on PC
- UNIX/QNX/PC based
- Servo control flexibility
- No hard real-time in trajectory generation

1998 1999
- Operator Interface Robot Simulator
- ARCL Servo Control on PC
- QNX/Windows/PC based
- Disassembly system
- Inhomogeneous components

2000
- QMotor Robotic Toolkit (RTK)
- Modular
- Object-oriented
- Data logging/plotting
- Control tuning
- Puma, WAM, IMI
Introduction – Review of Robot Control Platforms

Robot Control Languages
- VAL II
- AML
- V+
- IRDATA

Platforms for Servo Control Development
- MATLAB/Simulink
- RTWT
- RTLT
- D-Space
- WinCon
- QMotor

Full Scale Robot Control Libraries
- RCCL
- OSCAR
- ARCL

Problems:
- Proprietary
- Very limited flexibility
- Hardware integration

Problems:
- Need to start at the lowest level
  *Simulink solutions:*
  - Limited flexibility
  - Complex block diagrams
  - High computational burden

Problems:
- Too complex
- Limited flexibility
- Expensive hardware platforms required
**Introduction – Focus of this Research**

**Modularity**
- Independent components
- Easy to (re-) configure
- Easy Extension by adding new components
- Don’t need to understand the whole platform for extensions

**Flexibility**
- Easy extension/modification
- Extend/modify on all system levels
- Reuse code

**Real-Time Support**
- Establish hard real-time
- No special hardware (e.g., DSP) required
- Debug real-time code
- Log and plot control signals
- Parameter tuning
The QMotor Robotic Toolkit (RTK)

- Overview
- Object-Oriented Design
- Run-Time Issues
- QMotor
- Class Design
- GUI Components
The QMotor Robotic Toolkit

- Ready-to-execute programs and libraries
- Uses QMotor for real-time execution, logging, plotting and tuning
- Works only on the joint level
- Contains servo control, trajectory generator, teachpendant and utility programs
Advantages:

- Intuitive modeling of the physical system
- Multiple physical objects through multiple software objects
- Inheritance
Object-Oriented Design

- Code reuse
- No redundancy

Class Hierarchies

- `ManipulatorControl`
  - `PumaControl`
  - `WAMControl`
Common Functionality of all Manipulators

- Communication with the motion control board
- Setting output torques by setting voltages of the digital to analog converters (DACs)
- Position readings through encoders
- Enabling/disabling arm power by setting digital outputs
- PD position control
- Determining velocities and accelerations by backwards difference and filtering
- Communication with client tasks (e.g., to receive a desired trajectory)
- Switching between control modes (e.g., zero gravity mode/position control mode)
- Safety checks for joint and torque limits
- Manual calibration of the manipulator to a new (known) position
- Generation of a simple test mode trajectory
QMotor RTK – Object-Oriented Design / Deriving Manipulator Classes

class ManipulatorControl

control() → getCurrentPosition() → calculatePositionDerivates() → calculateControlLaw() → checkTorqueLimits() → setJointTorques()

Uses A/D readings as joint angles

Sets D/A channels to torque values

Derive class

class WAMControl

calculteControlLaw() → checkTorqueLimits() → setJointTorques()

Coupling Effects
Design Concepts – **Specific Manipulator Control Classes**

- **ControlProgram**
  - **ManipulatorControl**
    - **WAMControl**
    - **IMIControl**
    - **PumaControl**

### Specific WAM Functionality
- Automatic encoder calibration
- Joint coupling
- Gravity compensation
- Torque ripple compensation
- Damping control instead of disabling the arm power

### Specific IMI Functionality
- No arm power control

### Specific Puma Functionality
- Automatic encoder calibration
- Joint coupling
- Gravity compensation
How can an object-oriented design execute on a real machine?

Concurrency
- Some components need to execute concurrently
- PC is fast enough (No need for special processors/hardware)
- Components run as separate programs
  (Easy reconfiguration)

Communication
- Client/Server architecture
- Generic components

Real-Time Performance
- Need Real-Time Operating System -> QNX4
- C++ overhead can be neglected
- Need to be careful with dynamic memory allocation
- QMotor for control parameter tuning and data logging/plotting
The QMotor Graphical User Interface

- Provides an intuitive user interface
- Provides flexible real-time data plotting
- Provides control tuning

Main Window

Plot Window

Control Parameter Window
QMotor RTK – GUI Components

Manipulator Control Panel

Teachpendant

Manual Move Utility
Conclusions

• Concluding Remarks
• Future Research
The QMotor Robotic Toolkit

- Lightweight modular platform, entirely implemented as PC software
- Object-oriented homogeneous design allows code reuse and easier extension for new hardware and new algorithms
- Addresses the issues of concurrency and real-time
- Data logging, control tuning and plotting from the QMotor GUI
- GUI components (Teachpendant, control panels)

Code Reuse

- 63% of the system is independent of the manipulator type, i.e., it can be reused for new manipulators
- Common code is well tested
- Implementation of new manipulators is very quick
Disadvantages of the QMotor Robotic Toolkit

- Does not work in Cartesian space
- No 3D robot simulator
- Startup/Shutdown inconvenient with many components