Developing for the OpenRTM Based Software Controller

○ Yuichiro Kawasumi, Hajime Saito (General Robotix, Inc.), Noriaki Ando (AIST), Rosen Diankov (University of Tokyo)

2Q1

IS: Advances in Open-source Robotics Tools (1/2)
Talk Overview

Development of motion control system on OpenRTM.

- Target Platform
- System Overview
- Motion Control System, Basic Philosophy
- Implementation of RTC
Target Platform
- Dual arm robot “HIRO”

- 23 DOF
  - Head 2 DOF
  - Waist 1 DOF
  - Arm 6 DOF
  - Hand 4 DOF
- 4 USB cameras
  - 2 on head
  - 2 on each hand

URL: http://nextage.kawada.jp/index.html
System Overview

VideoStream (capture) → Processing Image RTC (By user) → Python Script
  - Initialize RT System
  - use service ports

Capture command → Joint angles → Get Status

End effector position

Capture command → Joint angles → Get Status

Joint angles

State Arbi-trator → Gaze Fixer → Grasper (User /GRX)

Collision Detector

Robot Hardware

Data Logger

Coordinate Converter

ArmIK Left → Arm Sync Right → ArmIK Right → Arm Sync Left → Sequence Player

Feedforward Control RTC

Feedback Control RTC

Real-time Control Loop

Critical Path: RTC’s along this path must pass all data through
System Overview (Simplified)

- **Robot Hardware**
- **IO Program**
- **Data Logger**
- **PD Servo (Simulated Hardware)**
- **Sequence Player**
- Switch

- to simulator (OpenHRP)
- to real devices
Motion Control System, Basic Philosophy

- Binary compatibility between Simulation, Actual Robot
- Common component structure (identical I/O port)
  - Base class for Component Implementation
- Python (or Jython) operation interface
Binary Compatibility
simulation and real robot

Use same binary on upper layer
We are preparing lower layer for various robots. Use same binary on upper layer.

We are preparing lower layer for various robots.
Common component structure

Identical I/O port

struct TimedJointData
{
    Time tm;  // the time at which data created
    sequence<Time> cmd;  // the time at which command
    sequence/octet> id;  // ID of RTC
    sequence<sequence<double>> qCommand;  // reference angles
    sequence<sequence<double>> qState;  // actual angles
    sequence<sequence<double>> dqState;  // actual velocity
    sequence</octet> servoCommand;  // servo command
    sequence</octet> servoState;  // servo status
    sequence</octet> collisionFlag;  // collision status
};

→ Joint values for each part(BODY, RARM, RHAND, ...)
Service Port for Interface

To communicate from external client,
1. define the interface as idl (Interface Definition Language)
2. generate c++ template code by idl compiler
3. implement the service functions
4. compile all of them and incorporate into the RTC

```
#Sample.idl
interface SampleService
{
    void hello();
};
```
Python operation interface

- Initialize RTCs
- Use service ports

Python Script

Sequence Player

Data Logger

Robot Hardware

PD Servo (Simulated Hardware)

IO Program

Sequence Player → Data Logger → Robot Hardware

Sequence Player → PD Servo

PD Servo (Simulated Hardware) to simulator (OpenHRP)

Robot Hardware to real devices

Switch

to simulator (OpenHRP)
Implementation of RTC
Overview of Class Structure

Sample RTC

- HrprtcBase
- Sample_impl
- Sample_service_impl
- sample_core

IDL for Service Port

Sample.idl

- Not to be generated.
- To be generated as template.
Implementation of RTC Base Class (HrprtcBase)

By Common functions you can use ...

- Calculate with robot model
- the limit values for each joint
- map of joint id to part id (RARM, LARM, etc).
Implementation of RTC
Fill the template functions

ReturnCode_t Sample::onInitialize() {
    HrprtcBase::onInitialize(); // <---- model and part info. is initialized
    /* implement other Initialization */
}

ReturnCode_t Sample::onExecute(RTC::UniqueId ec_id) {
    m_jointDatIn.read(); // <--- read joint data from InPort
    /* implement main process */
    m_jointDatOut.write(); // <--- write joint data to OutPort
}
To Compile using CMake

#CmakeLists.txt

...  
set(comp_name Sample)
set(comp_sources ${comp_name}.cpp)
set(comp_sources ${comp_sources} HrprtcBase.cpp)
set(comp_sources ${comp_sources} sample_core.cpp)  ---> add your codes
...

generate_stub_skel(${comp_name}Service)
generate_hrpsys_stub_skel_wrapper(JointDataTypes)
...

target_link_libraries(${comp_name} yourlibrary)  ---> add your libraries
...
Test with OpenHRP
We introduce the robot control system for dual arm robot 'HIRO'.

We separate common functions as super class for motion control RTC, to create RTC easily for user.

You can get them from google code soon.

Latest information is available from google groups: “propelopenrtm”
Environment & Resources

- **OS:**
  - Ubuntu Linux 10.04 LTS (32bit, 64bit)
  - QNX 6.5

- **Open Source Projects on google code:**
  - openhrp-aist-grx (openhrp with collada model)
  - hrprtc-grx-video-stream (capturing by usb camera)
  - hrprtc-grx-arm-ik (Ik sample using OpenRAVE) => not yet!
  - hrprtc-grx-sample (template sample) => not yet!

- **Forum**
  - Google groups : propelopenrtm