IST-2001-37652

Hard Real-time CORBA

Title

D3.7
Robot Control Testbed
Documentation

Authors

Klas Nilsson (klas@cs.lth.se)
Mathias Haage (mathias@cs.lth.se)

Reference IST37652/090 Deliverable D3.7
Date 2003-10-23
Release 1.0
Status Final
Clearance Consortium

Partners

Universidad Politécnica de Madrid
Lunds Tekniska Högskola
Technische Universität Wien
SCILabs Ingenieros

Summary Sheet

IST Project 2001-37652
HRTC
Hard Real-time CORBA

D3.7 Robot Control Testbed
Documentation

Abstract:

The Robot Control Testbed (RCT) for Hard Real-Time CORBA (HRT CORBA) was designed and implemented to permit experiments with hard and soft real-time communication transports with CORBA. To fully accomplish HRT CORBA, the ORB and the application would need to execute on a RTOS. Alternatively, only the execution of the threads that have to fulfil HRT requirements are run on an RTOS, while the establishment of object connections and the soft RT threads can run on an ordinary OS. The developed testbed is based on this alternative approach, using Linux for the soft RT part and the RTAI in Linux kernel space for the hard RT parts. Other platforms are possible, but not within the scope of the efforts reported here. A robot system, including a HRT transport in terms of the ThrottleNet RT Ethernet protocol for the servo control, is used for illustrating the CORBA and communication techniques. In documented application, the robot uses stereo vision to see and catch a thrown ball.

HRTC Partners:

Universidad Politécnica de Madrid
Lunds Tekniska Högskola
Technische Universität Wien
SCILabs Ingenieros.

Release Sheet (1)

Release:	**0.1 Draft**
Date:	2003-09-29
Scope	Initial version
Sheets	All

Release:	**1.0 Final**
Date:	2003-10-23
Scope	Final version
Sheets	All

Table of Contents

1 Introduction

This document describes the testbed experiment (catching a thrown ball) together with all equipment used for the experiment. It does not describe the software used in the experiment (HRT software D3.5, SRT software D3.4) and the different tests and measurements that have been made on the testbed (testing D3.6).

2 Testbed experiment

The purpose of the experiment is to catch a thrown object using a 6-DOF industrial robot. A stereo vision system estimates the ball trajectory and a predicted catch point. The experiment was originally developed (without HRTC extensions) as a PhD student project at Dept. of Automatic Control by PhD Bo Lincoln (bo.lincoln@control.lth.se) and PhD student Johan Bengtsson (johan.bengtsson@control.lth.se).

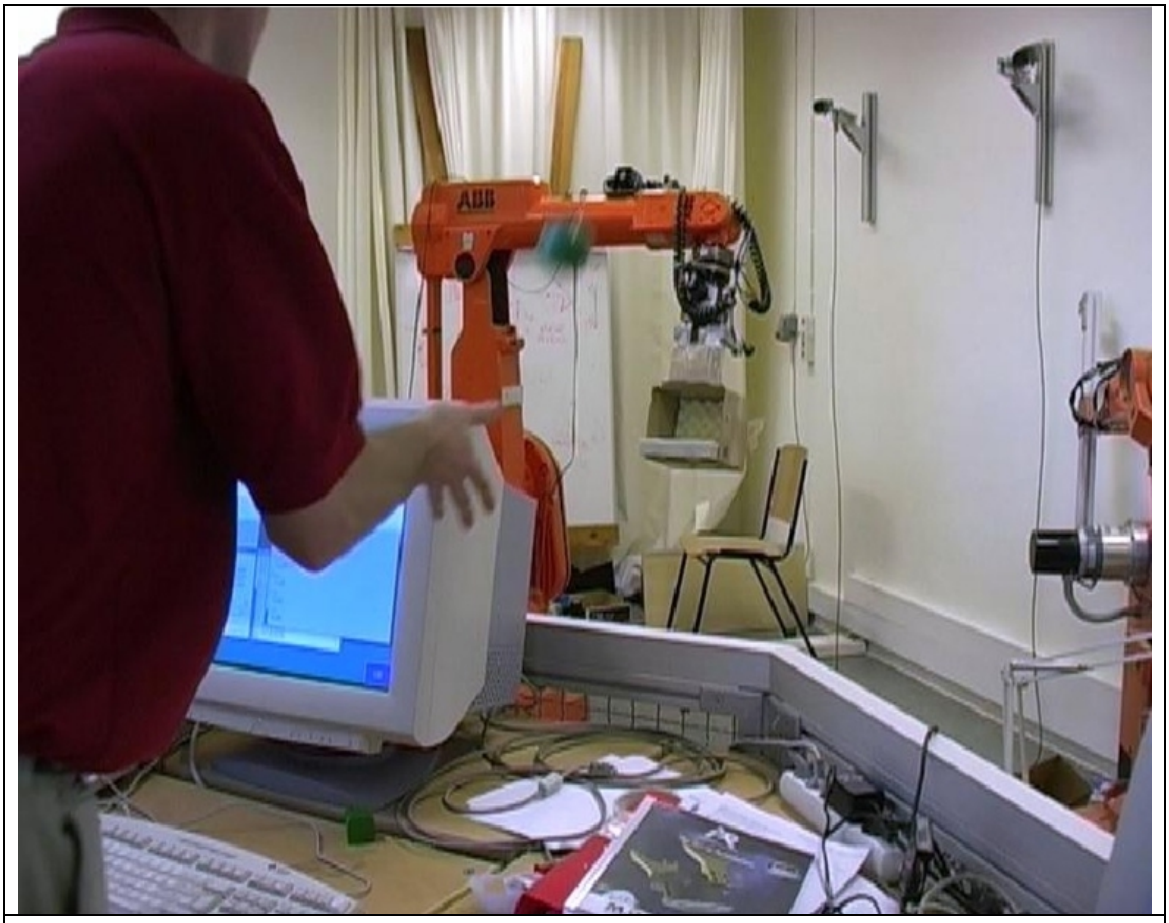

Figure 1: Experimental setup showing the two digital cameras mounted on the wall with the Irb-2000 robot ready to catch the thrown ball.

The experiment setup consists of an ABB Irb-2400 industrial robot and two digital video cameras calibrated to Cartesian space. The two cameras are placed on a wall behind the robot in stereo configuration, facing the throwing person. Included in the vision subsystem are three computers, each with one dedicated task. The images from the cameras are sent through an IEEE 1394 network to the low-level vision computer. From this computer, image feature points are sent to the ball trajectory estimation computer, located on a TCP/IP network. This computer in turn sends a predicted catch point to a third computer which calculates a robot reference trajectory. Finally, the trajectory is sent to the robot control system through a TCP/TN network bridge.

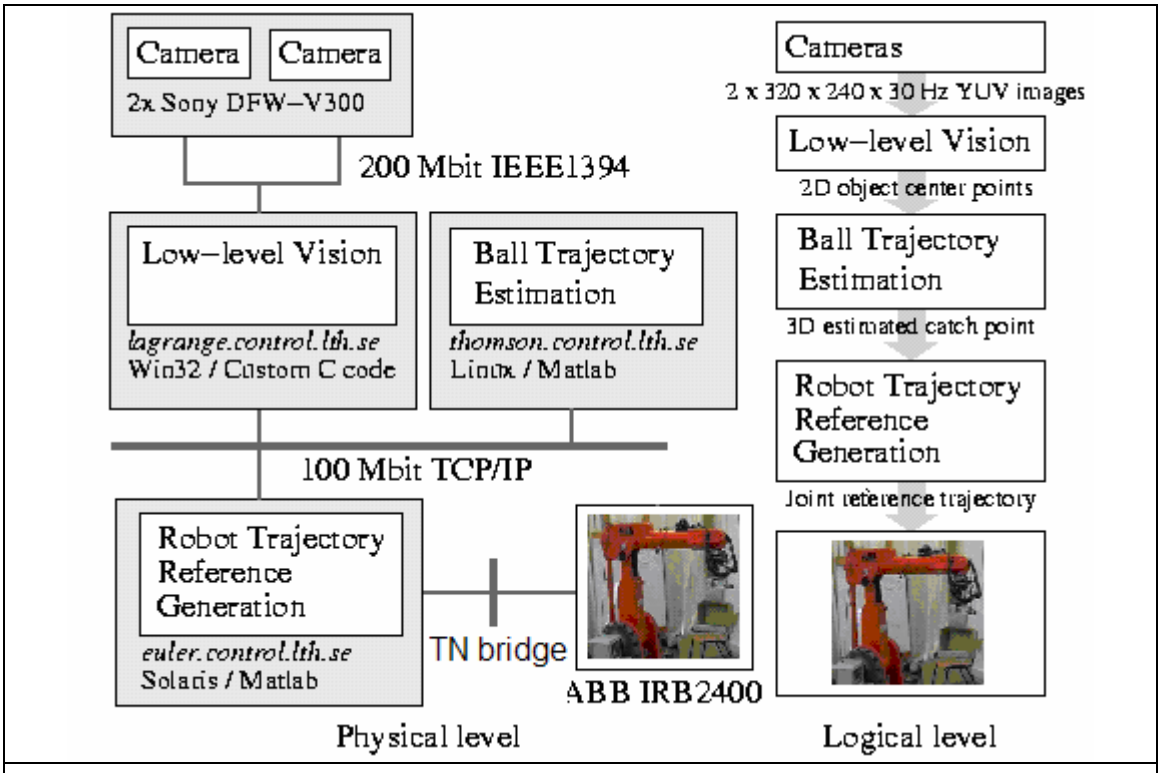

Figure 2: The vision subsystem runs on three different computers (excluding the TN bridge which is run on a separate Linux/RTAI machine). To the right, the information flow in the vision subsystem is shown.

2.1 Low-level vision

The low-level vision computer receives 2 separate 30 Hz streams of 320x240 pixel YUV images from two digital cameras. Green color spots of a specific size are extracted as 2D feature points. Feature points are sent at 30 Hz rate to the ball trajectory estimation computer using TCP/IP.

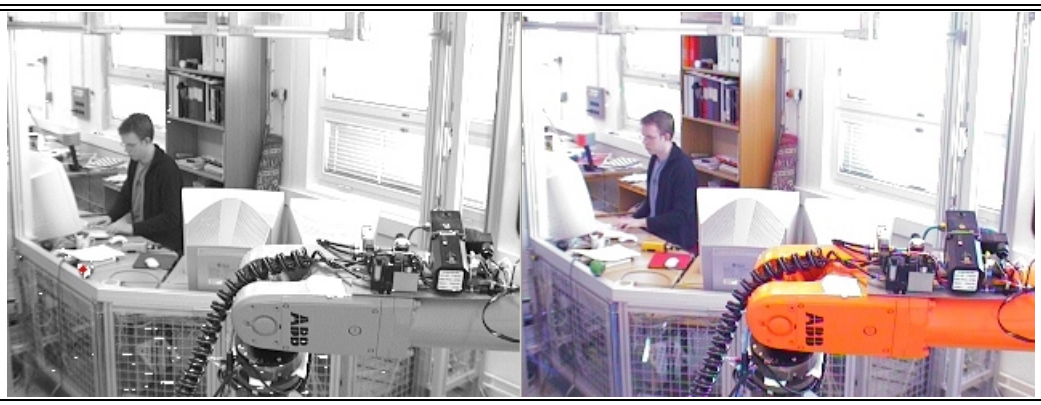

Figure 3: A typical camera image with pixels in the detection color interval (greenish) shown in white in the right image. All patches of more than 25 pixels are considered a feature and marked in red (in this case only the ball). The color thresholding technique is, as expected, quite noisy. The over-exposure is intentional as the images are taken towards the window.

The feature extraction subsystem is conceptually simple, and implemented in C on a PC computer to be able to process the images from two cameras at 30 Hz.

Features are extracted using an orthogonal color interval in the YUV color space (which is delivered from the cameras). For the experiment, the selected color interval is centered around green hues. This thresholding technique is usually quite sensitive to light and noise, and therefore a wide color interval has been chosen to ensure that the tracking object is included. The green interval is defined as

$$60 < Y < 150$$
$$U < 120$$
$$V < 135$$

for Y, U, and V values between 0 and 255.

A flood-fill algorithm is applied to the thresholded image, calculating the center points of all connected patches of "green" pixels. A patch must consist of a minimum number of pixels to be valid (most often 25 pixels in a 320x240 image). This removes most of the noise-like outliers.

It should be mentioned that the robot lab is located in a room with highly varying light conditions (from only artificial light to direct sun light). This is a very tough environment for any vision system, but the vision system appears robust. For extreme variations the aperture of the cameras need to be adjusted, but normally the automatic gain of the cameras handle it.

The center points of all patches in all cameras are sent to an Extended Kalman Filter (EKF), which has a dynamical model for free-falling objects. The EKF will determine if the image features belong to any of the tracked objects, and, if so, use them as measurements.

2.2 Ball trajectory estimation

The ball trajectory is continuously estimated in real-time using the accumulating number of available samples before the ball is caught. As new estimates become available, the catch trajectory is updated accordingly. An extended Kalman filter is applied to estimate the free-fall parabola of the ball. The estimate is used to predict a potential ball catch point and if the catch point is close enough to the robot gripper a robot catch trajectory is sent to the robot controller.

The potential catch point is located in a 2D-plane in the robot workspace. The plane is located such that mostly joints 2 and 3 in the robots are used to reach the catch point.

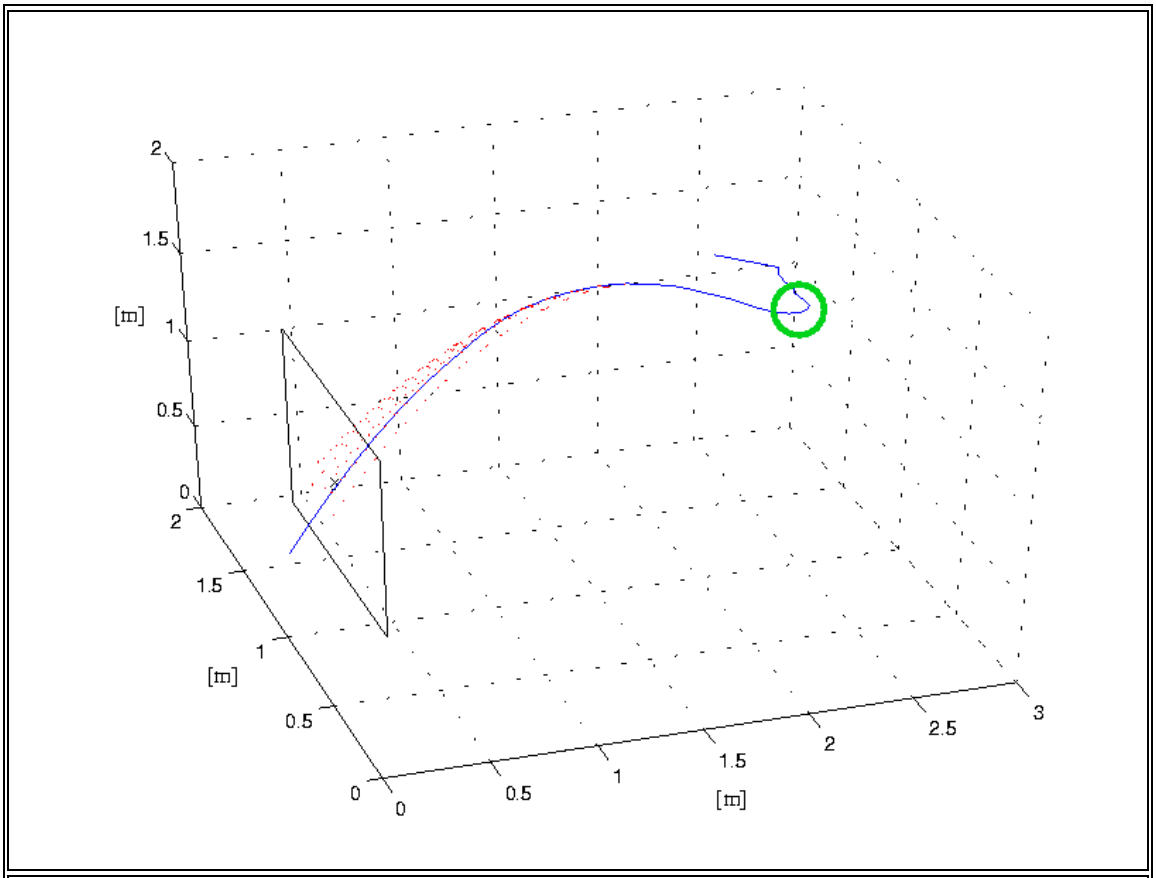

Figure 4: The trajectory of throw 1 marked in blue, with prediction trajectories sent to the robot in red. Black square marks the working area of the robot, and green circle marks the

start of the throw (the wobbly part is when the ball is in the thrower's hand).

2.3 Trajectory generation

The robot catch trajectory is generated based on the latest predicted potential catch point (converted to robot joint space), as well as maximum robot joint accelerations and velocities. As the catch point is updated at 30 Hz, the trajectory is also re-calculated at this frequency. Typically the potential catch point converges against the real catch point as the ball approaches the cameras and the robot. In the current setup the first prediction is usually available about 0.5s before the catch. The robot controller is fed with sub-trajectories from the latest estimated catch trajectory. Sub-trajectories (currently consisting of 6 via-points) is sent to the controller at the rate of the vision system (30 Hz).

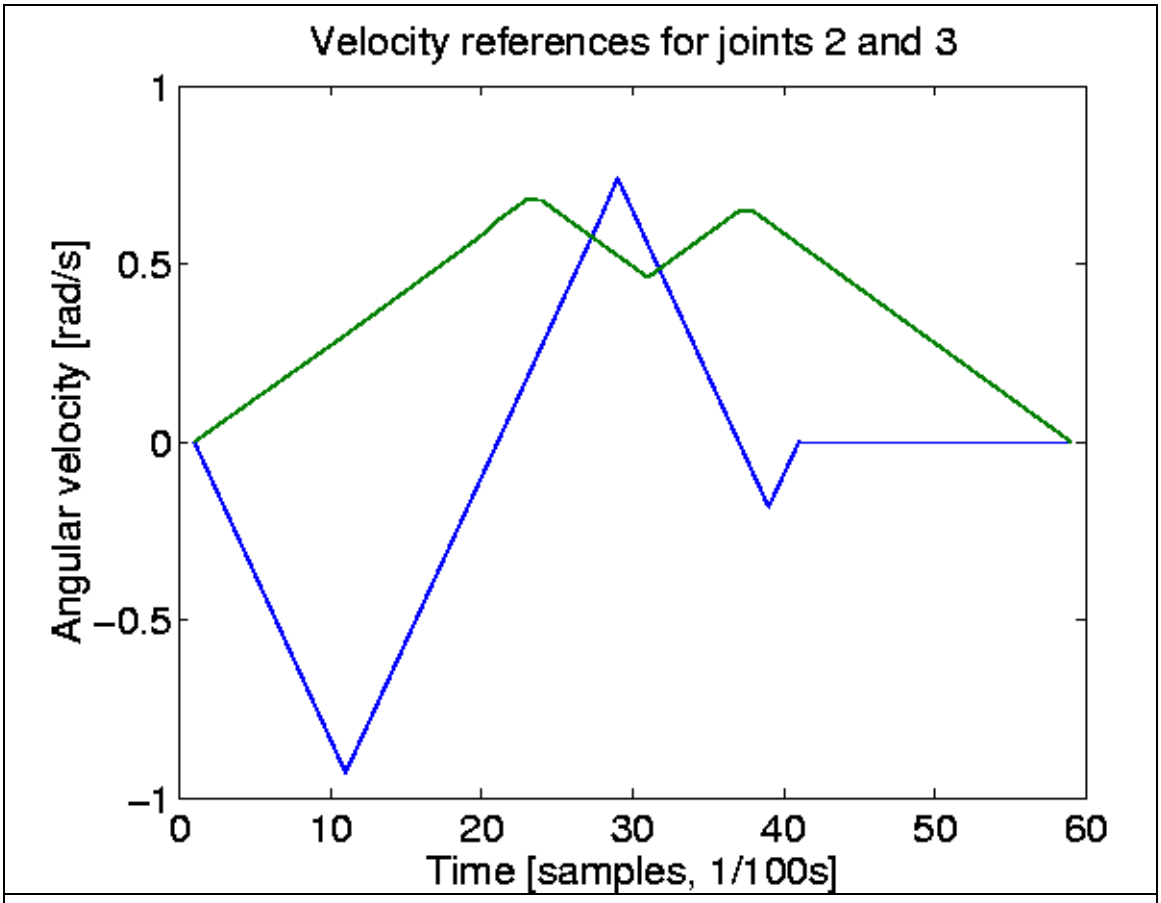

Figure 5: Velocity references for joints 2 and 3 for two different throws. Blue line (starting with negative slope) corresponds to joint 2, and green corresponds to joint 3.

Figure 4 shows a typical estimated trajectory from the tracking systems. The estimated trajectory is shown in blue, and the predicted trajectories from

different points in time are shown in red. As can be seen, the variance of the hit point estimation is quite small. For this throw, the first coordinate is sent to the robot about 14 frames before the ball hits the robot. This corresponds to about 0.47 seconds, which is most often enough for the robot to move to a hit point within 0.5 meters. The whole ball trajectory is about 0.7 seconds long for this throw.

3 Testbed robot controller

The ABB Irb2000 robot is controlled through an open controller (which is not ABB original) developed at Dept. of Automatic control. The control system consists of cascaded PID controllers with velocity feed-forward typically running at 8 (or 4 in some tests) kHz.

The testbed robot controller is built from four computer nodes. A distributed system is formed where three of the nodes communicate through switched Ethernet. From a control perspective a closed loop is formed by measuring current joint angles (resolver), calculating reference joint torques (controller) and driving joint motors (actuator).

The resolver and actuator computer nodes consist of Etrax computers running Linux. The controller node is implemented on two PPC MVME cards running Linux and a proprietary RTOS called Stork (developed at Dept. of Automatic Control). The Stork PPC card acts as slave against the Linux PPC card. Stork PPC and Linux PPC communicates through shared memory over the VME bus.

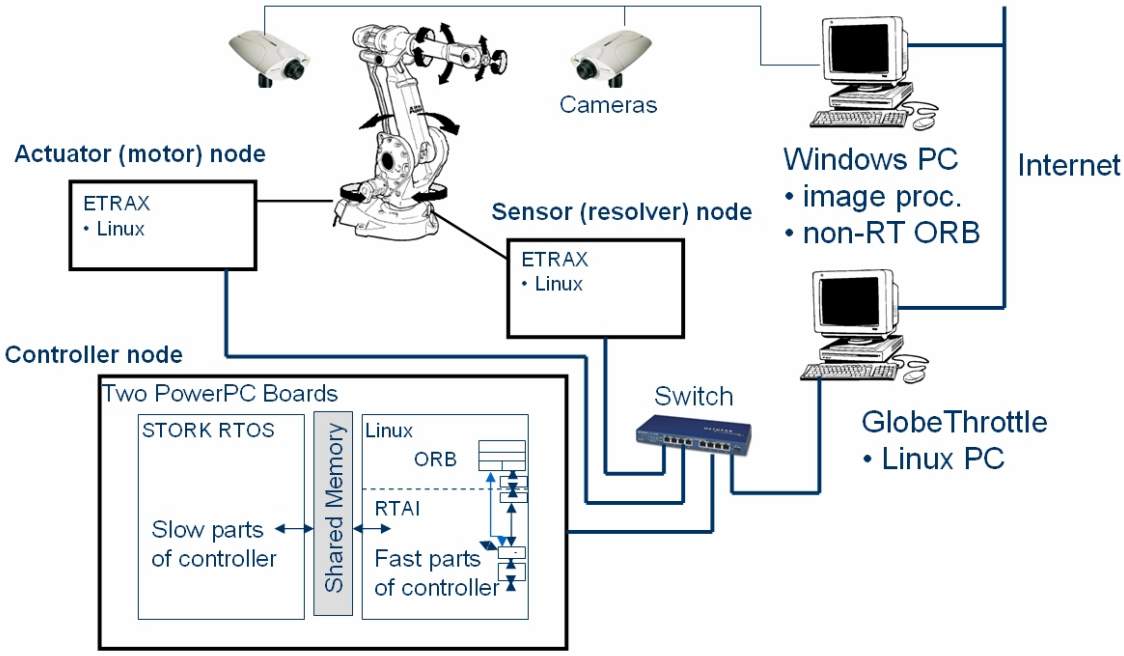

Figure 6: The robot controller together with the vision subsystem.

From an extrinsic perspective the robot controller expects to receive trajectories consisting of a vector of time-stamps, joint positions and joint velocities for joints 1-6. It is possible to divide a trajectory into several subtrajectories which are sent gradually to the controller. Thus, the end-point of the trajectory does not need to be known at the time the trajectory is started. This fact is utilized in the testbed experiment.

In the current setup the testbed experiment runs OCI-RT-ORBs on all nodes except the Etrax computers. For the moment it is not possible to run the TN driver on the Etrax:es due to Linux/RTAI problems, and therefore not the TN OCI transport. The old RT-system is therefore still used on the Etrax:es. On the other hand, all computer nodes connected through switched Ethernet have been running ORBs communicating through IIOP, so there seem to be no inherent problem with using Corba technology on Etrax:es.

4 RCT hard real-time requirements

The base frequency of the controller is adjustable up to 10 kHz, but typical sampling rates are 4 or 8 kHz. The controller implementation assumes the maximum latency from resolver to actuator to be one sample. This implies a maximum latency of 250 and 125 microseconds. The TN transport consumes a total of 100 microseconds in communication latency (resolver to controller and

controller to actuator) leaving 150 and 25 microseconds respectively for allowable computational delay. On the faster MVME-2400 boards, 25 microseconds is also the CPU time needed for performing the joint servo control. Hence, the likely overhead of full CORBA would not be acceptable, but the hybrid communication approach used permits a sufficiently fast processing.

5 CORBA viewpoint

Based on the experience and studies during the RCT implementation, this section contains a short discussion on key concepts in CORBA, RT-CORBA and OCI, from the viewpoint of approaches to HRT-CORBA support.

4.1 HRT in core CORBA

The philosophy behind CORBA is to provide location transparency for objects located across a network. For small embedded systems location transparency in itself is a problematic concept. For these systems the communication pattern is a big part of the system design, maybe bigger than the software design itself. The design is often based on a holistic system analysis. In essence, current CORBA is good at scaling up for enterprise type of systems, but what about scaling down to small distributed embedded systems.

It can be argued that HRT CORBA should be based on core CORBA or minimum-CORBA, and not RT-CORBA. The argumentation for RT-CORBA goes like the following from the Real-Time CORBA Specification v1.1 (p1-5): *"Interoperability may not be as important for a Real-time CORBA system as for a CORBA system because Real-Time dictates a measure of system-wide design control to deliver predictability and therefore also some control over which ORB to deploy."* The system-wide design approach is understandable, and even necessary, if distributed RT computing is based on the same models and assumptions as non-distributed computing is. For instance, thread scheduling based on priorities is somewhat problematic but useful within a single CPU, but simply using the same approach for distributed RT systems creates according to our RCT experience two types of problems:

1. Priorities are global, preventing proper modularisation. That is, the timing of each object depends on the system globally.
2. The management of priorities (in IDL and implementation) adds complexity and memory needs, decreasing portability to small system.

In a wider perspective, both priority and resource management as defined in RT-CORBA can be questioned for HRT purposes. A time-triggered approach (in design and/or communication) appears to be more appropriate.

There is to our knowledge no conclusion on this issue, but assuming the extra RT-CORBA features were omitted, there are some core CORBA details that deserves some attention from an embedded systems point of view:

- Some interesting concepts exists in the Corba specification, for example transient IORs holding location details for making direct connection to a server, bypassing any ORB daemon process, which could potentially be borrowed for hard-RT purposes and provide a very useful concept for incorporating hard-RT into the specification in a graceful, minimal change kind of way. Unfortunately (at least for transient IORs) the ORB utilization of transient IORs is implementation specific, thereby not providing a stable platform to base hard-RT implementation upon.
- Corba 3.0 specification section 13.6.10.1: "Other transport protocols can be explicitly specified when they become available."
- Corba 3.0 specification section 13.6.10.6: "New protocols can be added to corbaloc as required. Each new protocol must implement the <future_prot_addr> component of the URL and define a described in Section 13.6.10.1, "corbaloc URL" on page 13-24."
- The specification of human readable object references (corbaloc) allows explicit definition of new transport protocols as they become available. Still allowing for new protocols seems to be an implementation issue.

There is also the concept of Interceptors [pureCORB, p459]: *"Interceptors provide hooks into the ORB or interception points within the request/reply sequence,…" "They are a means of structuring an ORB's interactions with extra-ORB services."* and on page 460 it continues *"IOR Interceptors are concerned with adding service specific information related to an object or server into tagged components of the profile in the IOR when the IOR is created."*

RT-CORBA utilizes core CORBA to incorporate RT according to the following. Real-Time CORBA Specification v1.1, p1-5: *"Instead of specifying an RT-IOP, this specification* ***uses the 'standard extension' mechanisms provided by IIOP.*** *These mechanisms are GIOP ServiceContexts, IOR Profiles, and IOR Tagged Components."* Therefore, RT-CORBA gets bigger than core CORBA which is already (in many cases) too big for embedded systems. From this point of view, minimum-CORBA appears to be a more appropriate basis for HRT-CORBA. To freely explore possibilities, however, further experiments towards HRT-CORBA are within the RCT based on (but not using all of) RT-CORBA and the Open Communication Interface (OCI).

4.2 RT-CORBA and OCI

RT-CORBA offers selection and configuration of transport protocols. Not to be limited to the use of a few standardised protocols, there is support for implementation specific communication, in terms of CORBA services and the OCI. Some examples are:

- RT-CORBA Specification v1.1, p1-10: "New policy types are defined to configure the following server-side RT CORBA features:
  - "server-side thread propagation..."
  - "priority model..."
  - **"protocol selection"**
  - **"protocol configuration"**
- RT-CORBA Specification v1.1, p1-11: "Real-Time CORBA defines a number of policies that may be applied on the client-side of CORBA applications."
  - "creation of priority-banded sets of connections between clients and servers."
  - **"the creation of a non-multiplexed connection to a server."**
  - **"client-side protocol selection and configuration."**
- RT-CORBA Specification v1.1, p2-30: "ProtocolProperties should be defined for any other protocols usable with an RT-CORBA implementation, but unless they are standardized in an OMG specification their name and contents will be implementation specific. ProtocolProperties for other protocols may be standardized in the future,..."
- RT-CORBA Specification v1.1, p2-30: "The properties are provided to allow the configuration of protocol specific configurable parameters. Specific protocols have their own protocol configuration interface that inherits from the RTCORBA::ProtocolProperties interface."
- RT-CORBA defines the TCPProtocolProperties interface.

From a CORBA application implementation viewpoint RT-CORBA provides the needed set of definitions to accomplish bounded response times for a distributed application, and even hard-RT systems if the communication, the RTOS scheduling, and the ORB are appropriate for hard RT, and if the complete system is properly engineered concerning priorities and the like.

When it comes to the system and implementation specific techniques for real-time transports, however, RT-CORBA does not help concerning the implementation of such communication or its connection to the ORB. Here OCI comes in by providing an API for pluggable transports as stated in the OCI specification, p3-7: *"The Open Communications Interface (OCI) defines common interfaces for pluggable transports. It supports connection-oriented, reliable "byte-stream" transports."*

Furthermore, according to the OCI specification, p3-7: *"Non-reliable or non-connection-oriented protocols can also be used if the transport plug-in itself takes care of reliability and connection management."*

The OCI specification provides access to both the message layer and the stream layer. With the Scilab ORB used in the RCT we used the stream-based OCI. This makes it difficult to replace GIOP messages with messages more suitable for RT communication, but the need for that was not clear until the ThrottleNet implementation was tested.

From a RTCorba perspective the TN transport should probably be defined with an ProtocolProperty, but since RT-CORBA does not deal with transports it is necessary to combine it with the OCI specification.

6 CORBA OCI TN transport implementation

To accomplish ORB-aware hard-RT communication, the approach as described in the hard-RT implementation is to let the ORB run on the user level, whereas the hard-RT part of the application (including its TN communication) resides in kernel space. This so called hybrid approach to HRT-CORBA, see Figure 7, combines the features of user-space with the predictability of kernel-space. The TN tunneling of TCP/IP gives the benefit that core CORBA using IIOP works unchanged. In application where user-space execution, e.g. on some other OS, gives sufficient RT properties, there is also the option to make use of TN via OCI. This was also implemented, and also serving as an example of TN-OCI integration for TNIOP, a part of the test application is listed in Figure 8.

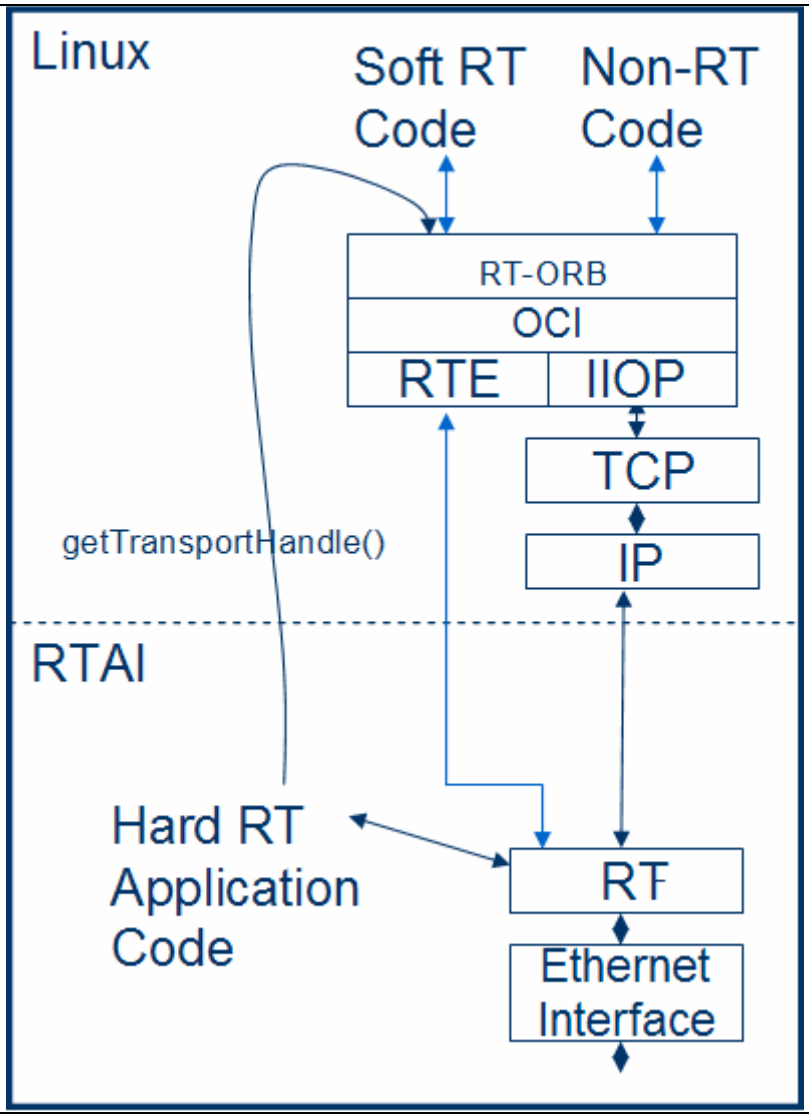

Figure 7 The hybrid approach of the hard-RT RCT implementation, with hard-RT parts of the application runningin Linux kernel space based on the RTAI.

```
// Get RTORB
CORBA::ORB_var orb = CORBA::ORB_init( argc, argv );
CORBA::Object_var obj = orb->resolve_initial_references( "RTORB" );
RTCORBA::RTORB_var rtorb = RTCORBA::RTORB::_narrow( obj.in() );

// Register appropriate transports with OCI
CORBA::Object_var confact_obj =
  rtorb -> resolve_initial_references( "OCIConFactoryRegistry" );
OCI::ConFactoryRegistry_var ConFactReg =
  OCI::ConFactoryRegistry::_narrow( confact_obj.in() );
// Build TNIOP transport
OCI_TNIOP::TNIOP_ConFactory* TNIOP_confactory =
  new OCI_TNIOP::TNIOP_ConFactory();
ConFactReg -> add_factory( TNIOP_confactory );
```

```
// Load Stringified Object Reference (from file for test purposes)
FILE *fIOR;
char ServerIOR[1024];
if ((fIOR = fopen("Servo.IOR", "r")) == NULL){
  fprintf(stderr, "Can not open input file Servo.IOR.\n");
  return -1;
}
fgets(ServerIOR, 1024, fIOR);
fclose(fIOR);

// String to object.
CORBA::Object_var object = orb->string_to_object(ServerIOR);
if(CALL_IS_NIL(obj.in()))
  {
   cerr << "Nil data reference" << endl;
   throw 0;
  }

//Narrow to Servo_var
RCT::Servo_var srv0 =RCT::Servo::_narrow(object.in());

//Apply client protocol policy
RTCORBA::ProtocolList protolist;
RTCORBA::Protocol proto;

proto.protocol_type=TAG_ID_TNIOP;//TAG_ID_IIOP if we want IIOP
proto.orb_protocol_properties = RTCORBA::ProtocolProperties::_nil();
proto.transport_protocol_properties = RTCORBA::ProtocolProperties::_nil();
protolist+=proto;

RTCORBA::ClientProtocolPolicy_ptr ClProPol=
  rtorb->create_client_protocol_policy(protolist);

CORBA::PolicyList pl;
pl+=ClProPol;

CORBA::Object_var object2 =
  srv0->_set_policy_overrides( pl, CORBA::SET_OVERRIDE);

RCT::Servo_var TNIOP_srv =
  RCT::Servo::_narrow( object2.in() );

CORBA::PolicyList plist;

TNIOP_srv->validate_connection(plist);

RCT::Servo_var srv = TNIOP_srv;


// Obtaining target joint pose q1..q6 from vision system, omitted here..

// Set Servo reference
srv->setTargetRef(q1,q2,q3,q4,q5,q6);

// etc. etc.
```

Figure 8 Using the TN transport in soft RT application code (user space).

The implementations, except for the ICa ORB itself, are available on request to any of the authors of this report. As documented in the RCT test report, the communication performance when using TNIOP (i.e. TN via the ORB on user space), similar to the listing in Figure 8, is not sufficient for the robot joint control (4 or 8 kHz). Instead, the hybrid approach, as depicted in Figure 7, provides the desired performance.

7 References

[CORB] Common Object Request Broker Architecture (CORBA/IIOP) Specification 3.01, Object Management Group, Needham, MA, U.S.A., 2002, http://www.omg.org

[RTCORB] OMG: Real-Time CORBA 1.0 Specification, http://www.omg.org

[minCORB] OMG: Minimum CORBA Specification, version 1.0, http://www.omg.org/docs/formal/02-08-01.pdf

[pureCORB] Fintan Bolton: "Pure CORBA", Sam's Publishing, 2002.

[InCORB] Mowbray, T. J. and W. A. Ruh: "*Inside CORBA*", Addison Wesley, 1997.