ABCM Symposium Series in Mechatronics - Vol. 4 - pp.490-498
Copyright © 2010 by ABCM

An Open Control System for Manipulator Robots

Diego Caberlon Santini, diegos@ece.ufrgs.br
Walter Fetter Lages, w.fetter@ieee.org
Universidade Federal do Rio Grande do Sul
Departamento de Engenharia Elétrica
Av. Osvaldo Aranha, 103-90035-190 Porto Alegre, RS, Brasil

Abstract. *In recent years, the demand for increased capability and flexibility has lead to an increase in the need for controllers based on open architectures. The Open Robot Control Software (OROCOS) project was born as an effort to match this demand. It is a general-purpose, open-source, modular framework for robot and machine control. In this paper, we present an implementation of a robot controller using OROCOS. The system environment is the Linux operating system with the RTAI real-time patch. The controller is based on a distributed architecture where each processing node is associated to a joint of the robot. Hence, the system could be extended to other robots. The CANbus is used for real-time data transfer such as sensor measurements and actuator commands. In OROCOS, each joint is mapped to a component and can use all the features of the framework. This reduces the cost and work because it allows the reuse of elements of an existing system. The paper present results of a controller implementation which illustrates the benefits of open architecture systems.*

Keywords: *Open architecture, OROCOS, robotics, real-time control, manipulator*

1. Introduction

The market of manipulator robots is dominated by closed architecture systems. A closed system is hard to modify, and therefore total cost of ownership can be very high. This type of system may be desirable when the application is well defined and is not expected to change over time (Ford 1994). However, properties like, flexibility, reconfigurability, modularity and reusability have been demanded by industries and this has lead to a growing use of controllers based on open architectures.

A open architecture approach allows for integration of new hardware or software components. Hence, the system can be continually improved to follow the constant moving needs of modern industry. Another great benefit of open controllers is the use of "Common of-the-shelf" (COTS) components (Miller 1993), which reduces the development time and cost.

The Open Robot Control Software (OROCOS) project was born as an effort to create an open architecture robot control system. It is a general-purpose, open-source, and modular framework for robot and machine control (Bruyninckx 2001). The present work describes the use of OROCOS to build an open control system for a manipulator robot. The paper describes the hardware developed to interface with the robot and the new OROCOS components created to integrate the whole system.

This papers is organized as follows: Section 2 presents the robot where the control architecture was implemented. Section 3 gives details about the hardware of the architecture. Section 4 introduces the OROCOS project, some background on how to build a component and some of the built-in components available in OROCOS. Section 5 describes the new components created to integrate the hardware with OROCOS. Section 6 shows the system interconnection and experimental results from a trajectory-tracking experiment. Conclusion and future works are described in Section 7.

2. The Janus Robot

The Janus robot, shown in figure 1, is used in this paper. It is an anthropomorphic two-armed robot, with eight degrees of freedom in each arm, and a stereo vision system. The vision system shown in figure 2 consists of two links connected in series by two revolute joints. Two cameras are attached to the far-end of the chain. Each joint is driven by DC motors and contains an incremental quadrature encoder and a reference index inductive switch.

3. The Proposed Control Architecture

The control architecture proposed in this paper is based on distributed processing nodes, called AIC, for each joint of the robot. Each AIC drives a DC motor through a PWM converter and interfaces with an incremental quadrature encoder, an index inductive switch and an electromagnetic brake.

A CANbus is used for real-time data transfer such as sensor measurements and actuator commands, between each AIC and a host PC. CANbus is an open protocol (Xuemei and Liangzhong 2007) that has gained widespread popularity not only in the automotive industry but also in the industrial automation arena. CAN has also proven that it fits very well into the suite of field-buses or sensor/actuator buses because of its low price, multiple suppliers, highly robust performance

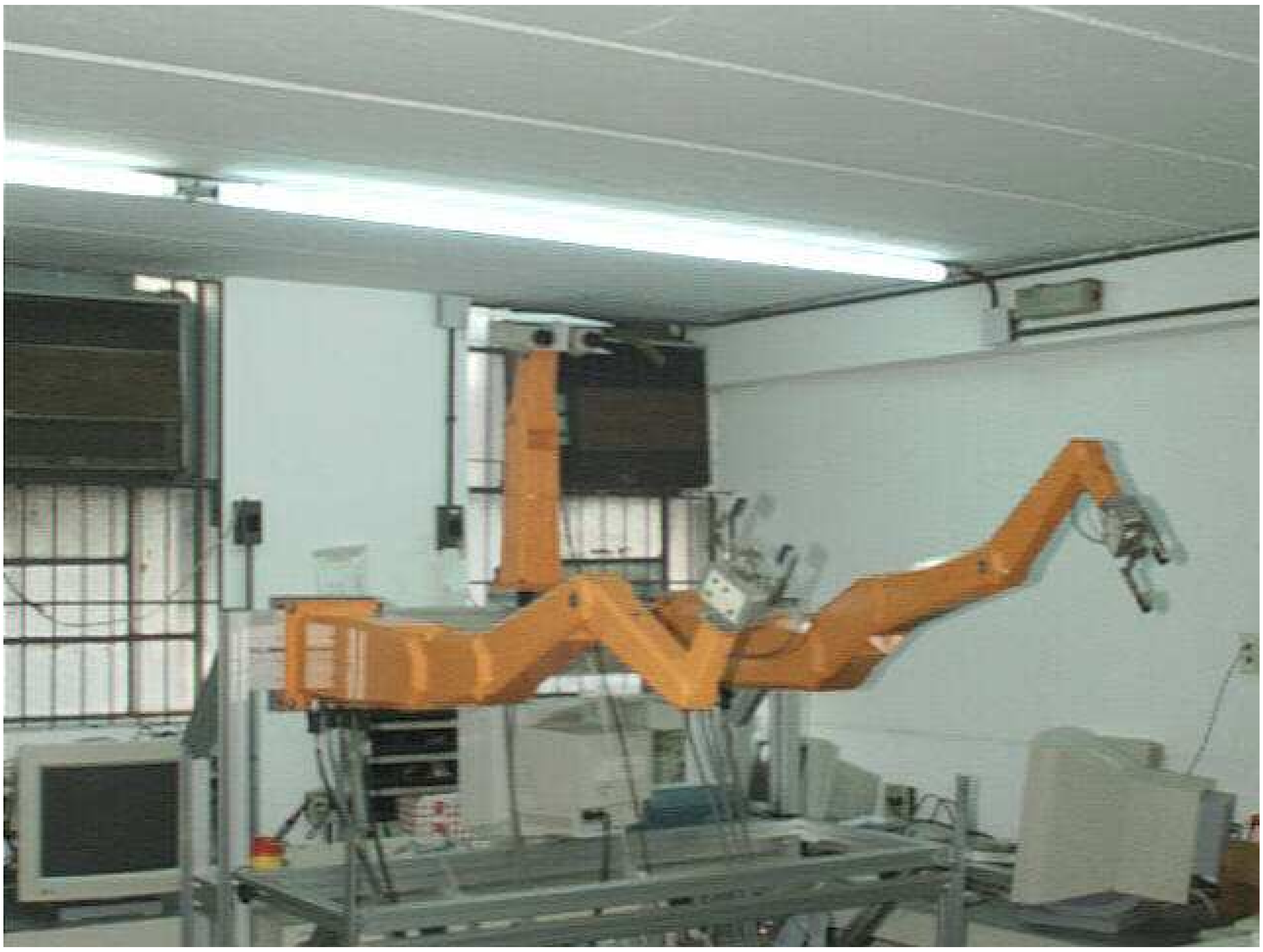

Figure 1. Janus robot.

and already widespread acceptance. Priority arbitration, error detection and re-transmission are all handled by the CAN controller hardware. Thus, the network may have a mechanism to ensure that the data transmission and reception is uninterrupted.

A personal computer with an AMD Athlon 64 4000+ processor and 1GB of RAM, running on Linux-2.6.22 with the RTAI-3.6 real-time patch is used as host. The controller based on the OROCOS framework executes in the host PC. A scheme of the connection between the host PC and the AICs nodes is shown in figure 3. More details about the hardware and software can be found in (Santini and Lages 2008).

4. OROCOS

The OROCOS project is a general-purpose and open robot control software package and follows the Open Source development model that has been proven to work for many other software packages (Bruyninckx 2001). The OROCOS rely on a ”divide and conquer“ approach and, as such, it relies on 4 supporting C++ libraries, as show in figure 4 :

The Orocos Real-Time Toolkit (RTT): provides the infrastructure and the functionalities to build robotics applications in C++, with emphasis in real-time, on-line interactive and component based applications (*The OROCOS project* 2009).

The Orocos Components Library (OCL): provides some components models for general purpose.

The Orocos Kinematics and Dynamics Library (KDL): allows for the calculation of kinematic chains in real-time.

The Orocos Bayesian Filtering Library (BFL): an independent framework for inference in Dynamic Bayesian Networks.

In this paper, the RTT and OCL libraries are used in their version 1.8.

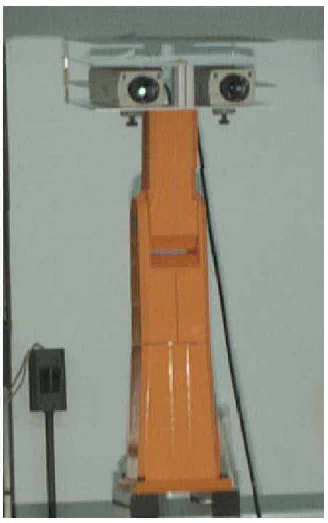
Figure 2. Vision system of the Janus robot.

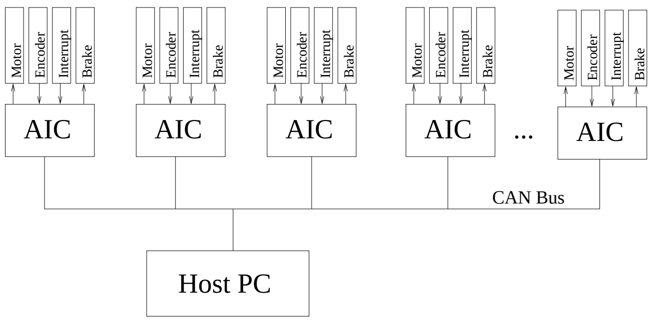

Figure 3. Distributed control system.

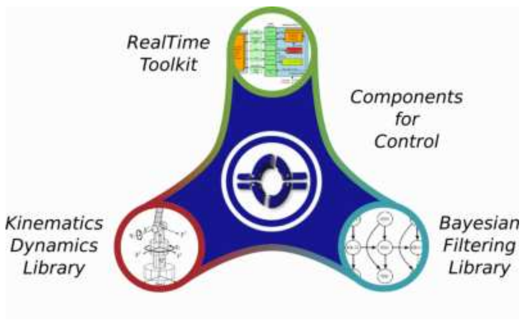

Figure 4. OROCOS libraries.

4.1 Real-Time Toolkit

The RTT is a middleware between the applications components and the operating system. It manages the communication, the execution flow and the configuration of components, as show in figure 5. In RTT, the interface of a component consists of attributes, properties, commands, methods, events and data flow ports. The interface is shown in details in figure 6. This library serves as a base to build an application upon.

A component model is a description of the component. It defines the interface, behavior, and implementation of the component. It is built using primitives from RTT. A component is a modular and replaceable part of the system that encapsulates the implementation and exposes its interfaces. Components are instantiated from component models.

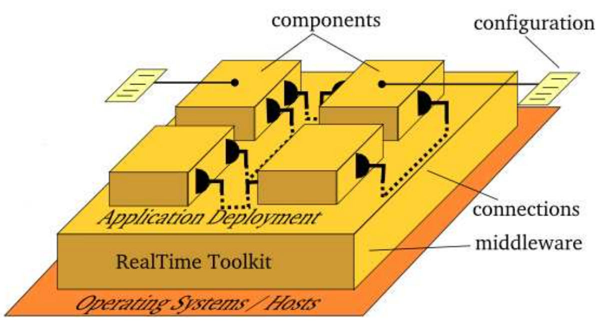

Figure 5. OROCOS as middleware(Soetens 2009).

Attributes and properties are used to configure the component when it is instantiated from a component model, although only properties can be written to and updated from a configuration file in XML format. In this way, it is possible to store persistent states, i.e., values that are important to keep between program executions, like the final joint position can be the next initial position. Reading and writing properties and attributes is done in real-time but is not thread-safe. Hence if a component reads a value while another component is updating such a value exactly at the same time, this could cause

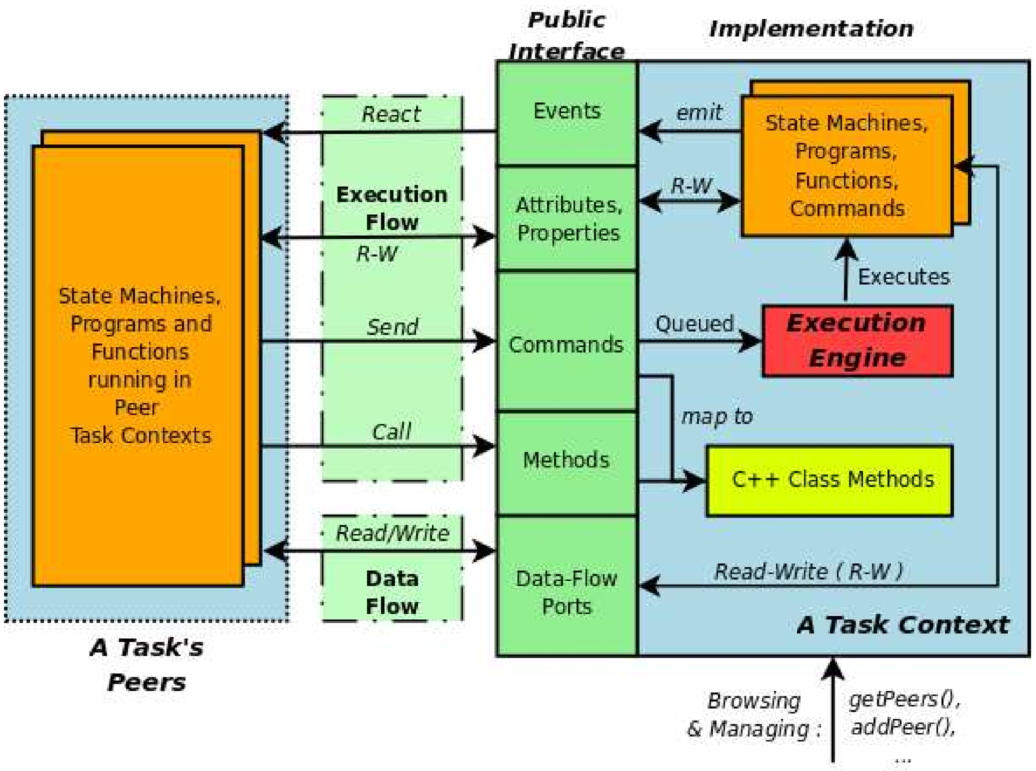

Figure 6. Interface of a component(Soetens 2009).

a mutal exclusion problem, resulting in an unknown value on reading. Because of this, reading and writing properties from for a running component are limited to the task own activity (Soetens 2009).

A command is a function that is executed asynchronously with respect to the caller, running in the thread of the command owner. Multiple commands, for the same component, are queued by RTT. Contrarywise, methods are intended to be called synchronously by the caller, and execute like a function in the thread of the caller. Calling methods is done in real-time but is not thread-safe and should, for a running component, be guarded with a Mutex if it's functionality requires so (Soetens 2009).

An event is a signal that is emitted to subscribers of that signal of the component. This allows for a reaction of other components to a change in the system. One or more functions can be called when an event is triggered, and they can be executed asynchronously or synchronously. Publishing and reacting to an event is done in real-time (Soetens 2009).

The data flow ports are the way to exchange information between components. A data flow can be read-only, write-only or read-write, through buffered or unbuffered ports. A read-only port can only be read in its component and cannot be write. Reading and writing data ports is always done in real-time and is thread-safe (Soetens 2009). The table 1 resumes the real-time characteristics for component interface.

Table 1. Real-time characteristics for component interface.

Interface	Real-time	Thread-safe	Synchronous
Event	yes	n/a	yes/no
Attributes/Properties	yes	no	yes
Commands	yes	n/a	no
Methods	yes	no	yes
Data Ports	yes	yes	yes

One component can only access the other component's interface when it is connected as a "peer". This connection can be uni-directional or bi-directional and allows the reaction of events, the sending of commands and the call of methods from another component. However, the data flow ports should be connected to each other in a explicit way and do not need to be connected as "peer". This is useful to build an interface for AIC component as described in section 5.

4.2 The OROCOS Components Library

The OCL is a set of components models contributed by users to the OROCOS project. This section explains the components that are used in this work: `TaskBrowser`, `DeploymentComponent`, `ReportingComponent` and `nAxesControllerPos`.

4.2.1 `Taskbrowser`

The `TaskBrowser` is a component model for user interaction with other components. When `TaskBrowser` component is connected to another component, it dynamically creates data ports and connects them to the other component. In this way, it can read/write in data ports, send commands to and call methods from the other component.

4.2.2 `DeploymentComponent`

The `DeploymentComponent` is a component model for loading and configuring other components through an XML file or an OROCOS script. In general when an OROCOS application is created and two components are instantiated from `DeploymentComponent` and `TaskBrowser` models. Then, the `TaskBrowser` component uses the XML file to command the `DeploymentComponent` component to do the system configuration. The `DeploymentComponent` component does the basic tasks: creates the components of the system; makes the interconnection between the components; configures the properties of the components from a specific XML file for every component; starts the components.

4.2.3 `ReportingComponent`

The `ReportingComponent` is a component model for monitoring and capturing data flow (from data-flow ports) between OROCOS components. The data can be logged into a file, or can be printed in a console. The configuration of what is logged and how data is captured is performed by a XML file. The `ReportingComponent` can be inserted into the system by the `DeploymentComponent` component just like any other component.

4.2.4 `nAxesGeneratorPos`

This component generates paths between the current positions and new desired positions for multiple axes. It uses KDL to compute the time interpolations. The paths of all axes are synchronized, meaning that all axes movements are scaled in time to the longest axis-motion. The interpolation uses a trapezoidal velocity profile using a maximum acceleration and a maximum velocity, which are configured in a XML file as properties of this component, together with the number of axis.

This component has a `moveTo(vector positions, double time)` command, which generates a new motion-profile starting from the current position to the desired position with a minimum time. Besides, a `resetPosition()` method resets the desired position to the current measured position and the desired velocity to zero, stopping the robot motion.

The `nAxesGeneratorPos` has two write-only data ports where the desired velocity and position are received and one read-only data port where the measured position is made available.

5. Components Models for Interface Janus to OROCOS

This section presents the components models that were build to interface the Janus to OROCOS: AIC, Controller and Bridge.

5.1 AIC Component Model

The AIC component model is an abstraction of the AIC hardware, show as block diagram in figure 7, details in (Santini and Lages 2008), thus represents every device in AIC card. Besides, it has to be able to represent any joint in the robot, independent of its details. For this, the AIC component model was created to grant flexibility and has a set of properties that enable the configuration of the component for each joint of the robot. These properties are:

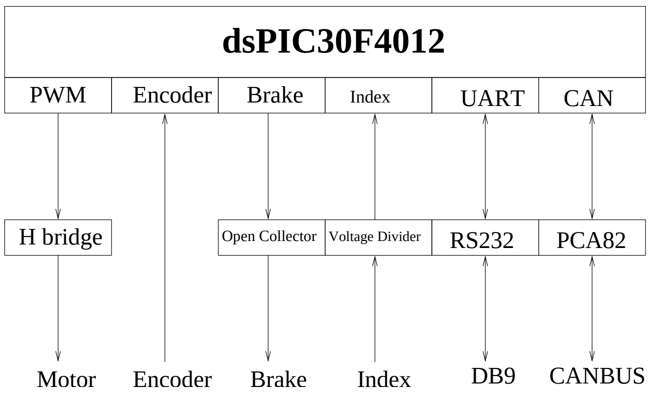

Figure 7. AIC block diagram.

JointNumber: The identification of the joint in the robot. Every AIC card has an ID number. This number identify the joint.

Brake: Informs weather the joint has a brake. Some joints of Janus have an electromagnetic brake to hold the robot position while it is in powered-off state. If the value of his property is true, the AIC component releases the brake on its initialization and applies on its finalization.

MotorSign: If true, inverts the sign of the voltage that is applied in the joint. This is useful because it is desirable that the motor turn in accordance with right hand rule and the Denavit Hartenberg Rules and the hardware is not always wired following this convention.

EncoderSign: If true, inverts the sign of readings of the encoder. Similar to MotorSign.

GearRatio: The gear ratio between the motor and the robot axis. This is necessary because the encoder measures the displacement of motor axis and the gear ratio is different for every joint.

InitialPosition: Initial position of the joint. Incremental quadrature encoder does not hold the absolute position and when the AIC component is initialized, it assumes that the joint will be in this position. This can be extended to store the last position of joint in the finalization of AIC component.

Each joint of Janus has an incremental encoder. This allows the measurement of the displacement of each joint and the position by integrating the displacement along the time. These data are writen into two write-only data ports. Some joints have an index inductive switch to detect the physical limit for the joint. This data is writen in a write-only data port. Furthermore, a write-only data port is used to make the voltage value that is applied to the motor available to other components. Note that all other components should only read this port.

The action of AIC component is made through the methods interface. The methods interface is used because it is synchronous with the caller, unlike the commands interface, and this is appropriate for a PID control strategy. The methods correspond to functions performed by the devices embedded in AIC card, like `MotorSet(double voltage)` to actuate the motor and `EncoderRead(void)` to update the position, displacement and index data ports.

5.2 PID Component Model

The PID component model is a implementation of a Independent PID controller in joint space, including control signal saturation. Each joint of the robot is treated as a single joint servomechanism. It is well-known that this control method is not the most effective for high performance manipulators, but it is widely used in industrial applications (Fu, Gonzales and Lee 1987) due to its simple implementation.

The PID controller involves three separate gains, shown in Equation (1): the proportional K_p, the integral K_i and derivative K_d. The $\bar{u}$ is the saturation, which means the maximum admissible value of the input. The error value $e(t)$ is the difference between the desired value and the measured output.

$$\begin{cases} u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \dot{e}(t) & , |u(t)| \leq \bar{u} \\ u(t) = \bar{u} & , u(t) > \bar{u} \\ u(t) = -\bar{u} & , u(t) < -\bar{u} \end{cases} \tag{1}$$

The discrete form of PID algorithm is as show in Equation (2) (Hemerly 1996). This Equation has a better response that the direct discretization on Equation (1) to systems with saturation constraints.

$$\begin{cases} u[k] = u[k-1] + k_p(e[k] - e[k-1]) + k_i e[k] + k_d(e[k] - 2e[k-1] + e[k-2]) & , |u[k]| \leq \bar{u} \\ u[k] = \bar{u} & , u[k] > \bar{u} \\ u[k] = -\bar{u} & , u[k] < -\bar{u} \end{cases} \tag{2}$$

The PID component implements the Equation (2) without loss of generality by including all its parameters as properties of PID controller. This way, every joint can have its own control law using the same component model. This grants more extendibility for the whole system. Another property is defined to tie the controller to an specific AIC.

In order to exchange data between the AIC and PID components, the PID must to have the data ports similar to those existing in AIC, but with a read-only permission. Besides, a read-only data port is used to receive the desired position from nAxesGeneratorPos.

The PID component creates a periodic thread that receive the desired position from nAxesGeneratorPos and the measured position from AIC. Then, it calculates the output value using Equation (2). The voltage is applied to the motor through the methods in AIC.

5.3 Bridge Component Model

In order to write the measured positions to nAxesGeneratorPos is necessary to put the AIC position in a vector format. The bridge component models has the purpose of convert data formats between robot-oriented built-in OROCOS components and the joint-oriented custom components such as PID and AIC. It has two read-only data ports of double to read the AICs positions and one write-only data port to write the vector with this positions. When a AIC component writes on its position data port, a interrupt occurs immediately and the vector of measured positions is updated too.

6. Experimental Results

The experimental system, shown in Figure 8, is used to control the vision system of the Janus robot. The system configuration is done with the DeploymentComponent component (Deployer) via a script and XML files. For each joint, an AIC component is created with its respective properties. Then the PID components are created and connected to their correspondent AIC, which grants them access to AIC's methods. Furthermore, the nAxesGeneratorPos component, ReportingComponent component (Reporter), the bridge component are inserted into the system.

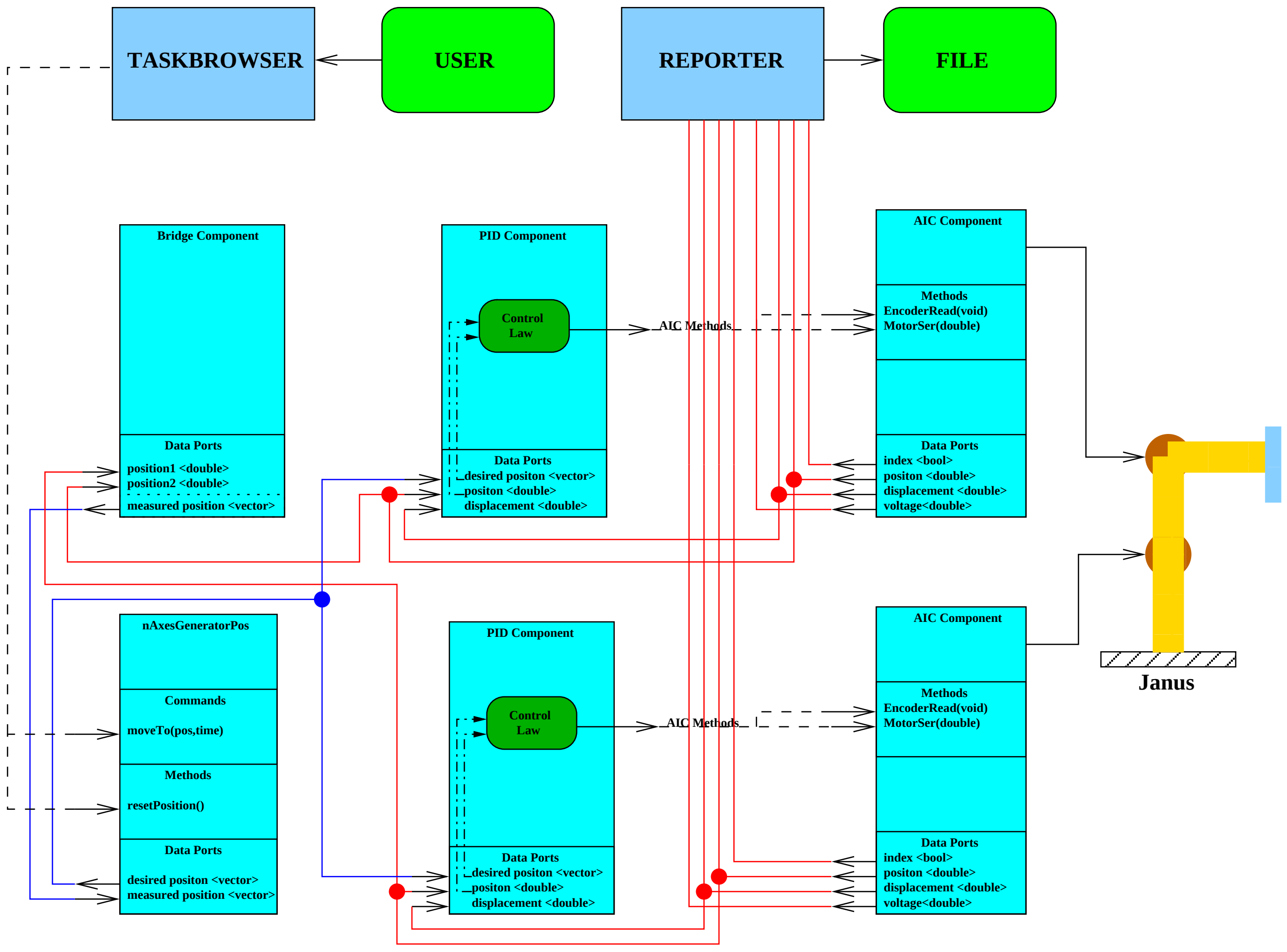

Figure 8. System.

All the data ports connections are done through a configuration file in XML format. The Reporter is connected to AICs to receive all the AICs status. It will be responsible for store this informations in log files. The TaskBroswer component is used as user interface, it can be connected and used to send commands to nAxesGeneratorPos component by the user. When a command is send, the nAxesGeneratorPos component will write new desired positions, that will be the reference for PIDs components.

The performance of the system is evaluated by a trajectory-tracking experiment. From the initial position, a new position is commanded to the joints of the robot, and the generated trajectory and the measured trajectory are compared in figures 9 and 10.

The performance of system is evaluated through the following criterions: integral absolute error criterion (IAE), integral time absolute error (ITAE), integral squared error (ISE) and integral time squared error (ITSE). The results are shown in table 2. In all cases, the results of the criterions is very small if compared to the excursion of every joint.

Table 2. System performance.

Criterion	Joint 1	Joint 2
IAE	0.0213	2.3584×10^{-5}
ITAE	0.0271	0.0149
ISE	0.0012	5.5622×10^{-10}
ITSE	0.0014	4.8340×10^{-4}

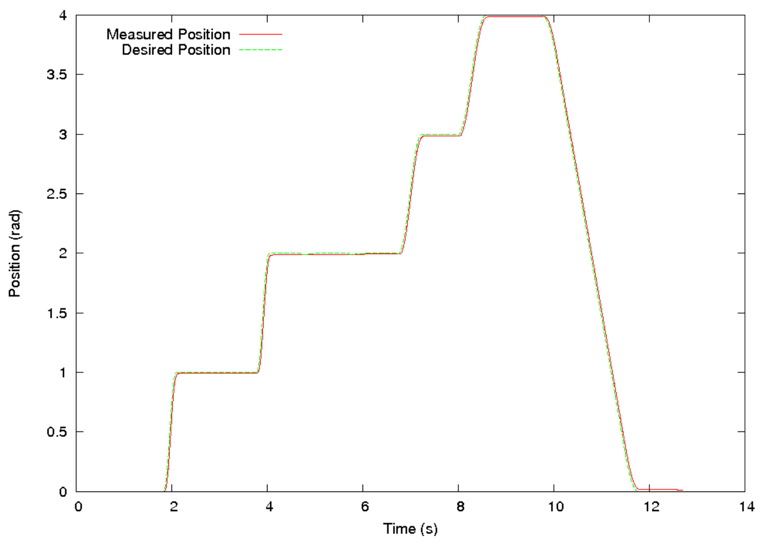

Figure 9. Trajectory-tracking of joint 1.

The generated and the measured trajectories are very close in both joints and this demonstrates that the performance of the system is satisfactory.

7. Conclusions and Future Work

An open architecture controller for the Janus robot was presented in this paper. By using the OROCOS and the component-based paradigm, it was possible to achieve:

- a reduced project time, due to the reuse of components, like reporter, avoiding the need to build these features. The RTT layer allows for working in real-time and to dealing with communications among threads without knowing all the the details of the implementation.
- an easily extension to other manipulator robots due to the facility of add more AIC components to control more joints.

The system was designed as a base for more complex work. In future developments, other approaches for control, like strategies using the dynamic model of the robot, will be implemented.

8. REFERENCES

Bruyninckx, H.: 2001, Open robot control software: the orocos project, *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, Vol. 3, Piscataway, NJ, USA: IEEE Press, Seoul, Coréia, pp. 2523–2528.

Ford, W.: 1994, What is an open architecture robot controller?, *Proceedings of the 1994 IEEE International Symposium on Intelligent Control*, Piscataway, NJ, USA: IEEE Press, Columbus, USA, pp. 27–32.

Fu, K. S., Gonzales, R. C. and Lee, C. S. G.: 1987, *Robotics Control, Sensing, Vision and Intelligence*, Industrial Engineering Series, McGraw-Hill, New York.

Hemerly, E. M.: 1996, *Controle por Computador de Sistemas Dinâmicos*, Edgard-Blücher.

Miller, D.: 1993, Standards and guidelines for intelligent robotic architectures, *Proceedings of AIM Space Programs and Technologies Conference and Exhibit*.

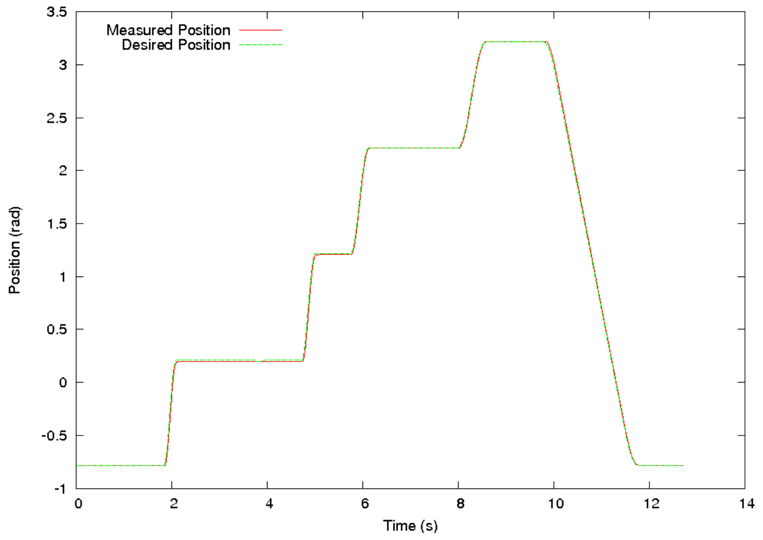

Figure 10. Trajectory-tracking of joint 2.

Santini, D. C. and Lages, W. F.: 2008, A distributed robot control architecture using RTAI, *Proccedings of the Tenth Real-Time Linux Workshop*, Centro Universitario del Norte, Universidad de Guadalajara, Colotlán, Mexico, pp. 1–5. Available at <`http://www.osadl.org/fileadmin/events/rtlws-2008/p19.pdf`>.

Soetens, P.: 2009, The orocos component builder's manual, *Technical report*, Flanders' Mechatronics Technology Centre. Available at <`http://www.orocos.org/stable/documentation/rtt/current/doc-xml/orocos-components-manual.p`

The OROCOS project: 2009. <`http://www.orocos.org/`> [Online; accessed 27-Abr-2009].

Xuemei, L. and Liangzhong, J.: 2007, Study on control system architecture of modular robot, *IEEE International Conference on Robotics and Biomimetics*, Piscataway, NJ, USA: IEEE Press, Sanya, China, pp. 508–512.