Robot Retrofitting: A Perspective to Small and Medium Size Enterprises

Walter Fetter Lages
Federal University of Rio Grande do Sul
Electrical Engineering Department
Av. Osvaldo Aranha, 103
90035-190 Porto Alegre, RS, Brazil
w.fetter@ieee.org
Alexandre Queiroz Bracarense
Federal University of Minas Gerais
Mechanical Engineering Department
Belo Horizonte, MG, Brazil
bracarense@ufmg.br

Abstract

This paper presents the authors experience in the retrofitting of an old ASEA IRB6 robot. It was verified that the mechanical parts of the robot were in good conditions, but the electronics parts were very outdated. A new controller architecture based on distributed system approach is proposed to replace the original robot controller. The hardware and software of the proposed architecture are described and experimental results obtained with the new controller are presented.

I Introduction

Industrial Robots are in use for a long time. Also, there are many works addressing the benefits and problems faced while robotizing a given process. However, small and medium size enterprises do not have access to this technology mainly due to the high costs. The high costs associated with robotics are not due to the robot itself only, but also due to robot accessories and programming.

Robot accessories such as I/O, communications and other interface cards are much more expensive than similar cards targeted to the PC market, even when scale-of-production effects are factored out. One of the reasons for such over-pricing is the absence of a standard architecture for robot controllers. Each robot manufacturer uses its own proprietary interface and protocols, hence robot users are forced to buy all the system from the same manufacturer. Furthermore, the marketing model used by major robot manufacturers discourages the incremental building of a robotic system. The purchase of additional modules much later after the installation of the robot is not an easy proposition.

Another point to pose difficulties to small companies is the programming of robots. Currently most industrial robot are programmed by dedicated robot programming languages that resembles more the Assembly language of a microprocessor than a modern programming language. Needless to say that the training of an employee to use such a language represents a significant portion of the cost to install a robot. Furthermore, it is not uncommon that the programming language changes even from series to series of robots from the same manufacturer, let alone from manufacturer to manufacturer. That means that it is not uncommon for a company to have to use one programming language for each of its robots.

There are commercial robot controllers that can control robots from any manufacturer [7]. Such a controller eliminates the need to learn different robot programming languages. However, its architecture is not open. Also, its programming language `RobotScript` [3] can be used for all robots supported by the controller. Nonetheless `RobotScript` is based on `VBScript`, a proprietary language tightly coupled to the Windows operating system. Besides Windows has not been designed with real-time control in mind, it is known for proprietary protocols that change from time to time. Hence it does not appear to be appropriate to serve as a basis for a standard open architecture for robot control.

An academic initiative for generating an open standard for robot control is the OROCOS (Open RObot COntrol Software) project [5]. This project is in its early stages and does not produced a working prototype yet. Also, it is more oriented towards the software architecture of the whole robotic system and does not properly address the supporting hardware architecture.

Open hardware and software architectures form robot control were defined by the PINO project [8], but since they were designed with small legged mobile robots in mind, they do not seems adequate for industrial manipulator robots.

On the other hand, large companies are replacing their old robots by brand-new robots. Due to much lower costs, these disposed robots would be a viable alternative for robotizing small and medium size enterprises. Nonetheless, since they are old technology robots, some benefits of using a robot can be lost.

Fortunately, the mechanics of industrial robots has not changed too much. The main differences from old to newer robots reside in the actuator power drives and controller, including the software. This fact enables us to upgrade old robots to current technology by retrofitting the robot controller.

This paper describes the experience of authors in retrofitting an ASEA IRB6 robot, including the development of an open architecture for robot control. The proposed architecture is not specifically intent to became a standard architecture for robot control, but should serve as a testbed to identify some of the requirements for a standard open architecture for robot control.

The reminder of this paper is organized as follows: Section II describes the ASEA IRB6 robot mechanics and the steps performed in order to adapt it to support the control architecture proposed in section IV. Section III describes the upgrades to the electrical circuitry of the robot, while section V describes the software proposed to run the upgraded robot. Experiments with the new control architecture are presented in section VI. Conclusions and directions for future research are discussed in section VII.

II ASEA IRB6 Mechanics

The ASEA IRB6, shown in figure 1, is a manipulator robot with five degrees of freedom built in 1977. Since it is an old robot, before the retrofitting with a new controller, it was completely disassembled in order to identify and verify the parts. Same parts damaged due to prolongated use were remanufactured. Figure 2 shows the first steps in the robot disassembly.

An actuator sub-assembly is shown in figure 3. Each actuator is composed by a D.C. motor, a resolver, a tachometer and a sync-switch. Axis 2 and 3, which are subject to gravitational forces, include electromechanical brakes.

Details of the actuator-joint couplings are shown in figures 4 and 5. Coupling of joint 4, not shown, is similar to coupling

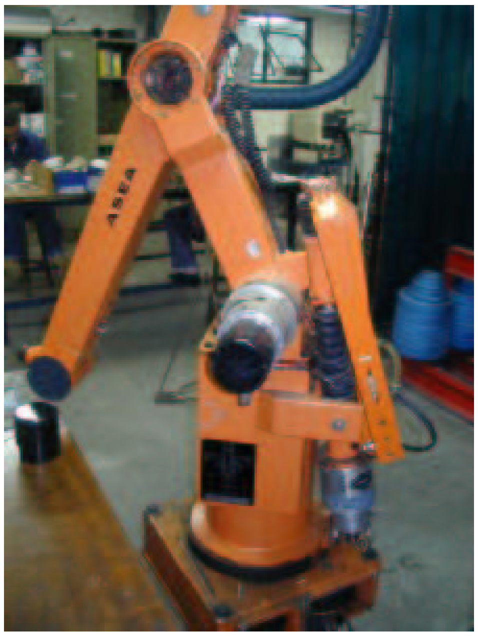

Fig. 1: ASEA IRB6 robot.

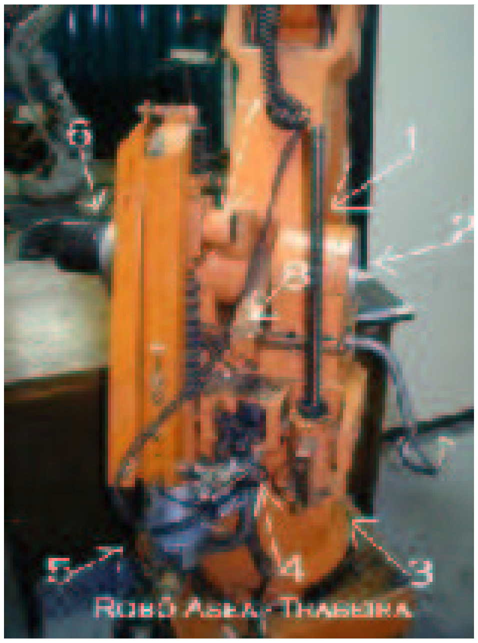

Fig. 2: Robot disassembly: Motors removed (2) and (3).

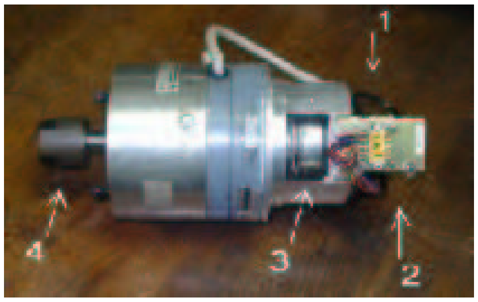

Fig. 3: Actuator sub-assembly: Resolver (1), connector (2), tachometer (3) and torque output point (4).

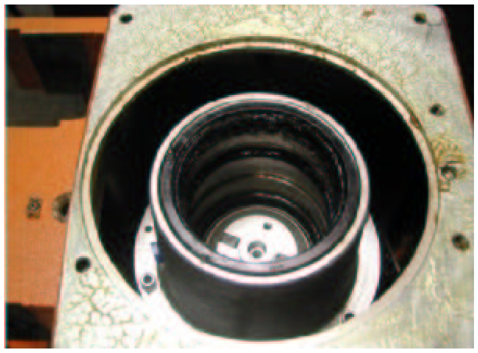

Fig. 4: Transmission for joint 1, the harmonic drive is shown without its rigid elliptic disk.

of joint 5.

After disassembly, cleaning, replacement of damaged parts and lubrication the robot was reassembled. In general it could be remarked that although it was an old robot its mechanical parts were in good conditions.

III Electrical Upgrading

Contrarywise to the mechanical components, the electrical components of the robot were not in good conditions. The original robot controller shown, in figure 6, was not operating at all. As usual a manual with electrical schematics of the robot was available, but the electronics parts are proprietary and seen as a "black-box".

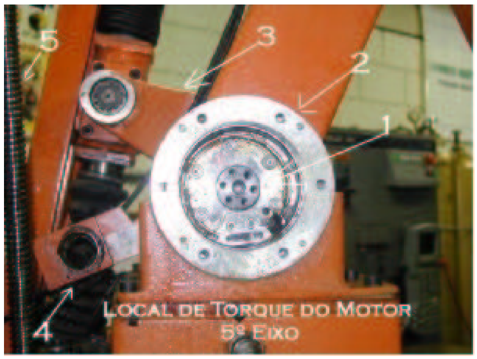

Fig. 5: Screw (5) and lever (3) for actuating joint 2, lever for actuating joint 3 (4) and coupling for joint 5 actuator (1).

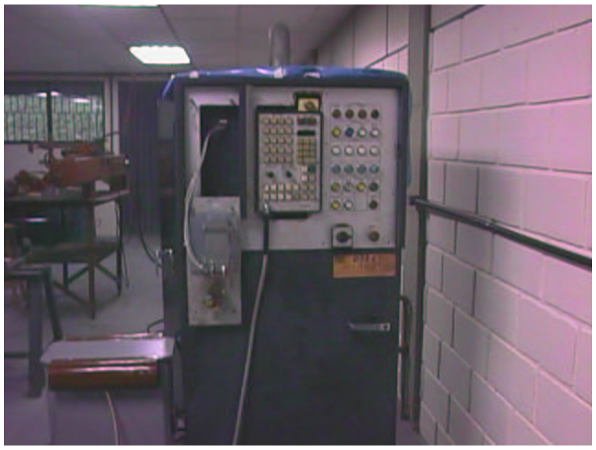

Fig. 6: Original ASEA IRB6 controller.

Since the controller technology was old, mainly based on analog electronics, it was replaced by a new one based on an architecture developed at UFRGS an described in section IV. From the original electrical components little more than the cabling and power supply was retained.

Originally the motors were powered by drivers based on analog linear amplifiers,

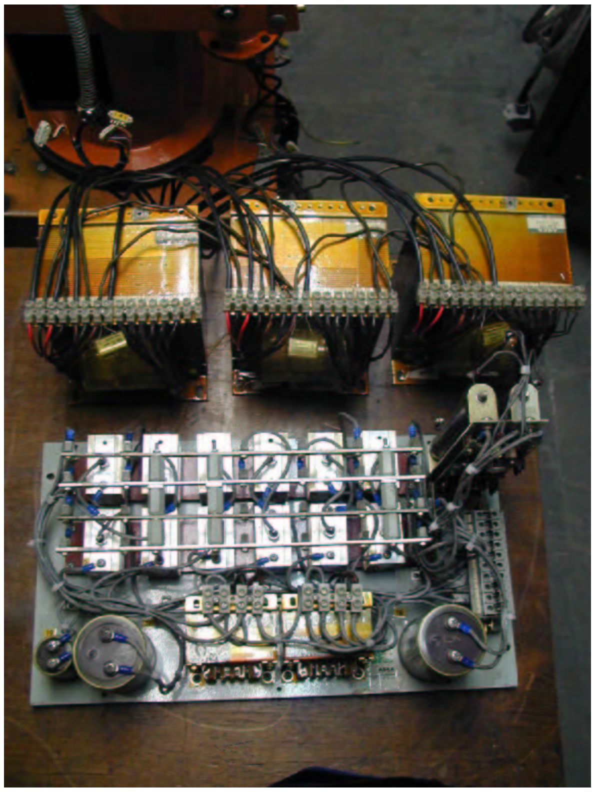

Fig. 7: Rearranged power supply.

implying a $\pm 47V$ symmetrical power supply. However in the new proposed architecture, the motors are powered by PWM amplifiers and H bridges, requiring a single $+24V$ power supply. Nonetheless, as the original power supply transformers had many taps, and playing with primary and secondary $\triangle/Y$ connections, the new power supply could be built by just rearranging the same components used by the old one. Figure 7 shows the reassembled power supply.

Also, as the new control architecture is based on digital technology, resolvers and tachometers were replaced by an incremental encoder with 2048ppr. The sync-switches were retained to serve as reference point for the incremental encoders.

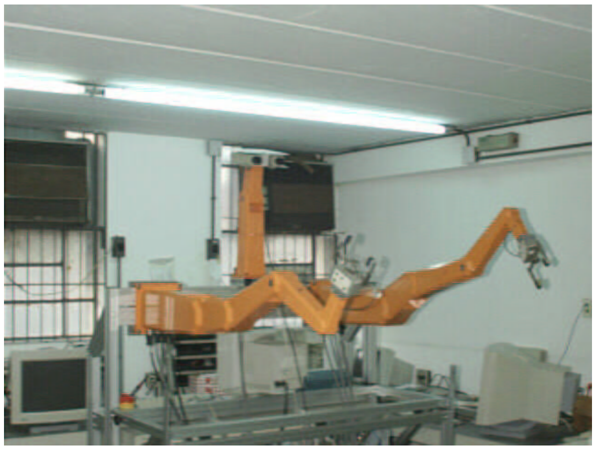

Fig. 8: Janus manipulator.

IV Control Architecture

The control architecture used in the retrofitting of ASEA IRB6 as not developed specifically for this robot. Actually, it was developed for the Janus robot [6]. Janus is an anthropomorphic manipulator (figure 8) with two arms and a pan-and-tilt stereo vision head. Each arm has 8 degrees of freedom. The joints are actuated by D.C. motors. Each joint has also an incremental encoder, a reference position inductive sensor and electro-mechanical brakes.

Although the architecture was designed for the Janus robot, no adaptation was needed to use it to control the ASEA IRB6. That is an indication that the proposed architecture is flexible enough to be used with many types of robots. A general view of the control architecture is shown in figure 9. It is a distributed processing architecture based on the Actuator Interface Card (AIC). Each joint has an AIC which

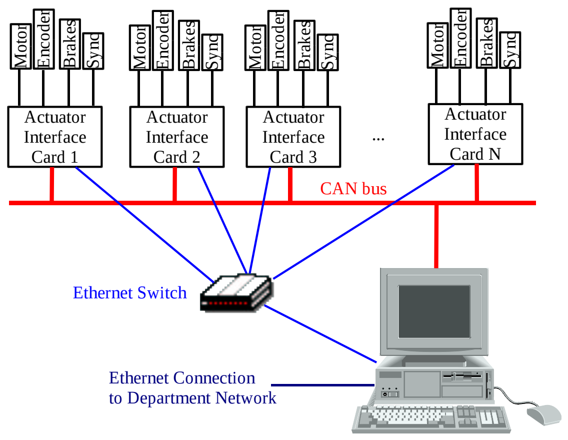
Motor
Encoder
Brakes
Sync
Actuator Interface Card 1
Actuator Interface Card 2
Actuator Interface Card 3
...
Actuator Interface Card N
CAN bus
Ethernet Switch
Ethernet Connection to Department Network

Fig. 9: Control Architecture.

directly drives the joint actuator and interfaces with associated sensors. They communicate through CANbus and Ethernet connections. It appears redundancy to have two connections, but they are used for diverse types of communication. CANbus is used for real-time data directly related to the robot operation such as sensor reading and actuator commands, while the Ethernet connection is used for supervisory data without direct relation with the robot operation. Real-time control over Ethernet connection is being developed [1], but it is not mature enough to be the basis of an industrial robot controller.

The user interface runs on a PC compatible computer communicating with AICs through CANbus. This computer has also a second Ethernet connection, thus acting as a gateway to the department network and to the Internet.

Note that the role of AIC in the control of the robot is not defined by this architecture beyond the point that it should be able to command the joint actuator and read the joint sensors. Behind this architecture is the idea that each AIC is a node in a distributed control system. It can either act as a joint controller, implementing simple control algorithms such as PID or as an I/O processor, with the control algorithm implemented on the main computer or any other node connected to the CANbus. To support this kind of flexibility, the software executed by each AIC can be uploaded through the Ethernet connection.

The main computer runs a real time variant of the Linux operating system called RTAI (Real Time Application Interface) [2]. Among the advantages of RTAI is the fact that it is possible to write hard real time programs under user space (using the LXRT API). Other variants of real time Linux require hard real time tasks to be written as kernel modules, restricting the tools that can be used and requiring more privileges from the user. For systems with many users unfamiliar with Linux, that is a serious drawback.

A block diagram of AIC is shown in figure 10. It is composed of two modules. The processor module obviously has the processor, a real-time clock, CAN, Ethernet and RS-232 interfaces. Currently it is implemented by a Dallas Semiconductor TINI module [4]. The interface module was specially designed for AIC. It has a SIMM72 socket for the processor module, a PWM, a quadrature decoder, and interfaces for sync-switch and brakes.

Since the processor module is a separate SIMM72 module, it can be easily changed, without the need to redesign the interface module, should a more powerful processor be required in the future. In order to accommodate the requirements of many types of motors, the frequency of the PWM can

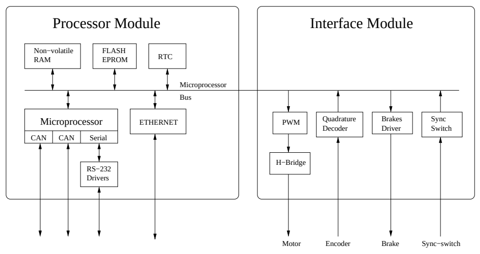

Fig. 10: Actuator Interface Card (AIC) block diagram.

be adjusted by software. The PWM output is connected to a MOSFET H-bridge, which can directly drive almost any D.C. motor used in industrial robots. The quadrature decoder receives the signal from a two-channel incremental encoder and decodes it, recovering the direction of motion and multiplying by four the encoder resolution. It is directly coupled to a 16-bit counter that stores pulses counts between processor readings.

In the retrofitting of ASEA IRB6, each AIC was assembled in a individual case, as shown in figure 11. However, for application where a higher number or AICs are needed, such as the Janus manipulator, they can be mounted in a standard Eurocard 19" rack.

V Software

The TINI module, used as AIC's processor, provides a runtime environment that includes a multitasking operating system supporting memory and I/O management, file system, a TCP/IP stack and a Java virtual machine. An UNIX-like operating system shell, based on Java, is also available. The operating system includes TELNET, FTP and serial console servers and a DHCP client.

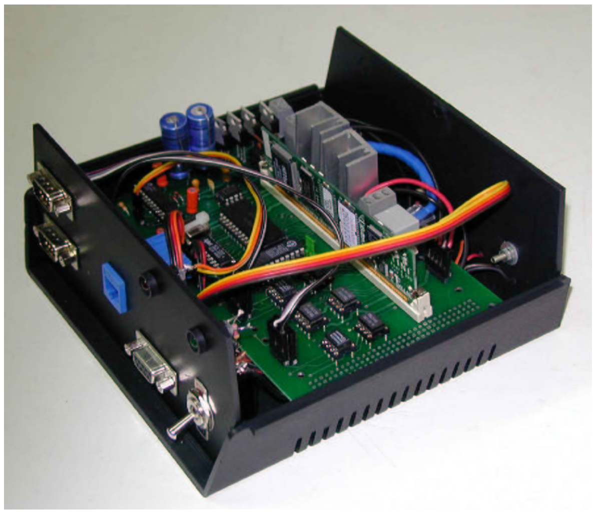

Fig. 11: Actuator Interface Card in its case.

Software running on AIC is based on Java technology. A Java package (`br/ufrgs/eletro/AIC`) was created to model the devices connected to an AIC: PWM, motor, encoder, sync-switch and brakes. There are also classes to model the whole AIC and the host interface. Of course, each class has public methods supporting the possible operations, for example: `apply()` and `release()` for the `Brake` class and `on()`, `off()` and `set(double voltage)` for the `Motor` class. Although all public methods are accessible in Java, time critical methods were implemented in Assembly as native methods. The `Host` class abstracts the communication channel. Its derived classes `HostCAN`, `HostTCP` and `HostUDP` encapsulates all details of communication with the host computer.

The AIC Java package is used to build the AIC daemon that can implement a joint controller or simply act as an I/O processor, as discussed in section IV. This daemon can be uploaded to AIC by using FTP and can be called by the initialization scripts. Since AIC has non-volatile memory, upon power-on the daemon will be ready to communicate with host computer.

Currently, the retrofitting of ASEA IRB6 is using a simple daemon that behaves like an I/O processor. It is implemented as a multi-threaded Java program. The initial thread handles user arguments and releases two other threads, one to send the reading of encoder each 10ms and other to receive and process commands, such as motor voltage to be applied, sent by host computer.

On the host computer side there is a similar structure. There are also classes modeling motor, encoder, sync-switch and brakes connected to an hypothetical AIC, a class to model the AIC itself, which is derived to `AIC_CAN` `AIC_TCP` and `AIC_UDP` classes that encapsulates the details of communication with the AICs. Therefore, the host programmer is isolated from the communication details. The host program can be developed as the AIC devices were local. An important point is that the host side is supported by a class library written in C++. That means that although the class and methods are similar to the ones existing on AICs, on the host side they are implemented in C++.

There are several reasons for the implementation of the host supporting library in C++ instead of Java. Initially, C++ appears to be a better language than Java for development of advanced control algorithms. Modern control theory and robot control in particular are largely based on matrix algebra. The operator overloading capabilities of C++ enables the development of very practical matrix manipulation libraries. Similar packages can not be build for Java since it does not permit operator overloading.

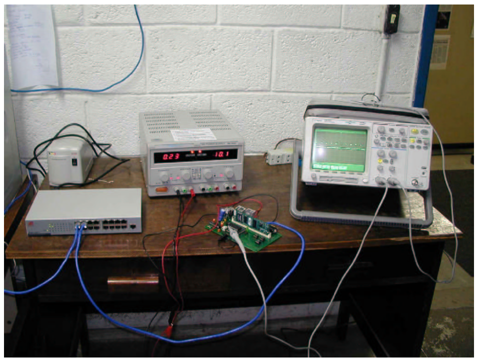
Fig. 12: AIC under test.

The host computer executes a real-time variant of the Linux operating system called RTAI [2]. This system does not support the execution of Java applications in real-time. Furthermore, standard real-time Java specifications are mature enough, while standards for real-time C/C++, such as POSIX, exist for a long time and are supported by RTAI and many others real-time operating systems.

VI Experiments

The architecture proposed in this paper was extensively tested in our labs. Figure 12 shows an AIC under test. Note the PWM signal on the oscilloscope screen.

After the individual test of each AIC they are mounted on the robot controller enclosure which was adapted to receive the new controller architecture. The new enclosure lay-out can be seen in figures 13 and 14.

A sequence of robot motions can be seen in figures 15- 20.

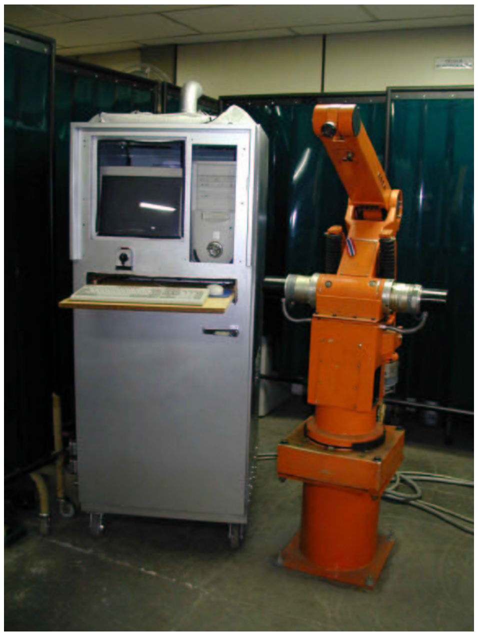

Fig. 13: Controller enclosure front view.

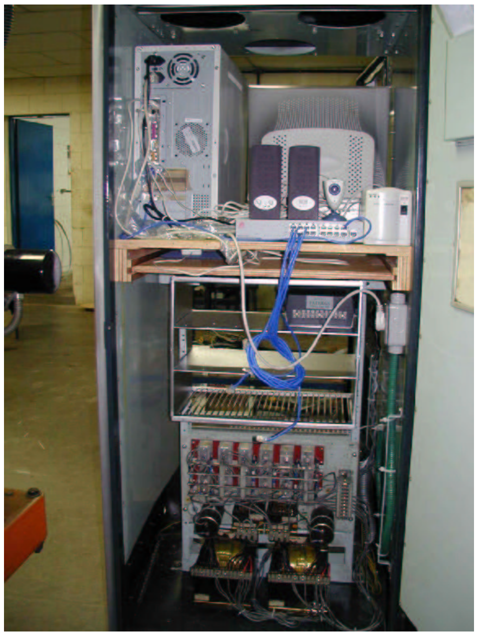

Fig. 14: Controller enclosure rear view.

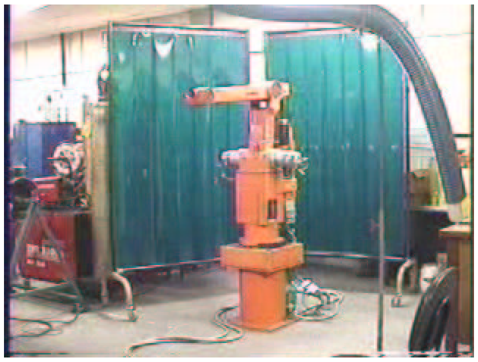

Fig. 15: Robot motion sequence.

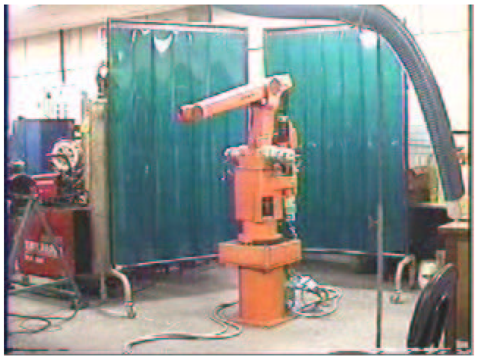

Fig. 16: Robot motion sequence.

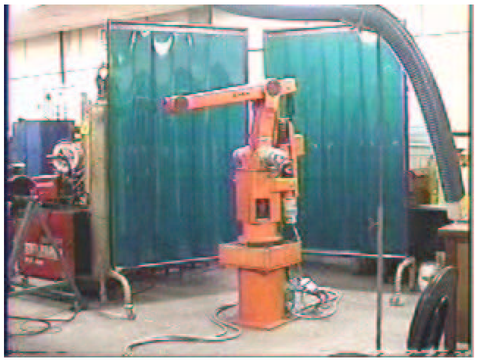

Fig. 17: Robot motion sequence.

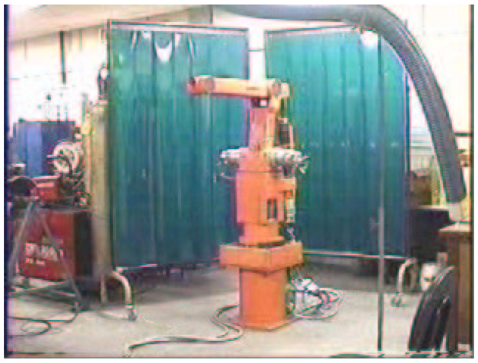
Fig. 18: Robot motion sequence.

Fig. 19: Robot motion sequence.

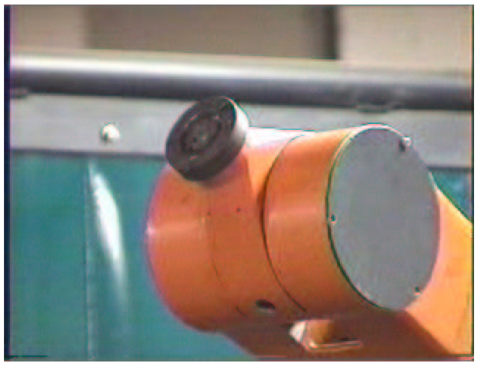
Fig. 20: Robot motion sequence.

VII Conclusion

This work described the experience of authors in the retrofitting an old ASEA IRB6 robot. Although it was an very old robot it was verified that the mechanics parts were in good conditions as opposed to the electronics, which was replaced by a new architecture based on distributed systems approach.

The architecture used to replace the original robot controller is very flexible since it was conceived for another robot and no modifications were needed to use it to retrofit the ASEA IRB6. Future improvements to the architecture include the development of a single chip to implement the interface module using FPGA technology and the development of a newer interface module to handle actuators powered by A.C. motors. Currently it is under development an interface module to integrate 6-axis force and torque sensors in the same architecture.

From the software point of view, the proposed system is based on Java and C++, which are current programming languages and therefore the cost for training robot programmers should be very reduced. Typically, only the C++ interface will be visible to the user through class libraries that masks out the details of communications with AICs. Future directions include the integration of the system with libraries that handle robot control at a higher level. A promising approach in this direction is the integration of C++ library for robot control with off-line programming packages such as Workspace.

Acknowledgments

The first authors was partially supported by the Research Support Foundation of the State of Rio Grande do Sul (FAPERGS), Brazil, through the Support Program for Scientific and Technological Development in Information Technology (PROADI) under grant number 01/05670.

Second author was partially supported by the Research Support Foundation of the State of Minas Gerais (FAPEMIG), Brazil, under grant number TEC 2471/98.

This work was also supported by RECOPE/MANET (Manufacturing Automation Network).

References

[1] G. H. Alt, R. da Silva Guerra, and W. F. Lages. An assessment of real-time robot control over IP networks. In *Proccedings of the Fourth Real-Time Linux Workshop*, pages 1–17, Boston, USA, 2002. University of Boston, Computer Science Department. http://www.realtimelinuxfoundation.org /events/rtlws-2002/paper.html #PAPER_g09_Alt.

[2] P. Cloutier, P. Mantegazza, S. Papacharalambous, I. Soanes, S. Hughes, and K. Yaghmour. DIAPM-RTAI position paper, nov 2000. In *Real-Time Operating Systems Workshop*, pages 1–28, Milano, Italy, 2000. Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano. http://www.rtai.org.

[3] J. Lapham. RobotScript the introduction of a universal robot programming language. *Industrial Robot*, 26:17, 1999. http://www.rwt.com/main/articles/ robotscript.html.

[4] D. Lommis. *The TINI Specification and Developpers Guide.* Addison-Wesley, Boston, MA, 2001.

[5] OROCOS. Open robot control software, 2002. http://www.orocos.org.

[6] R. Reginatto and W. F. Lages. Saturation compensation in the control of janus robot manipulator. In *Anais do XIV Congresso Brasileiro de Automática*, pages 74–79, Natal, RN, Brazil, 2002. Sociedade Brasileira de Automática.

[7] URC. Universal robot controller, 2003. http://www.rwt.com/main/urc.html.

[8] F. Yamasaki, K. Endo, M. Asada, and H. Kitano. Energy-efficient walking for a low-cost humanoid robot, PINO. *AI Magazine*, 23(1):60–61, 2002.