Arquitetura Aberta para *Retrofitting* de Robôs

Walter Fetter Lages†, Renato V. Bayan Henriques†, Alexandre Queiroz Bracarense‡

†Universidade Federal do Rio Grande do Sul
Departamento de Engenharia Elétrica
Av. Osvaldo Aranha, 103
90035-190 Porto Alegre, RS

‡Universidade Federal de Minas Gerais
Departamento de Engenharia Mecânica
Belo Horizonte, MG

w.fetter@ieee.org, rventura@eletro.ufrgs.br, 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 It was verified that the mechanical parts of the robot were in good conditions, but the electronics parts were very outdated.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.

Resumo

Este artigo apresenta a experiência dos autores no *retrofitting* de um robô ASEA IRB6. Foi verificado que as peças mecânicas do robô estavam em boas condições, mas a eletrônica era bastante antiga. Uma nova arquitetura de controle baseada em um enfoque de sistema distribuído é proposta para substituir o controlador original do robô. O *hardware* e o *software* da arquitetura proposta são descritos e resultados experimentais obtidos com a nova arquitetura são apresentados.

I Introdução

Robôs industriais estão em uso já há um bom tempo e existem diversos trabalhos enfocando os benefícios e problemas da robotização dos mais diversos processos. No entanto, pequenas e médias empresas não tem acesso à esta tecnologia principalmente devido aos altos custos. No entanto, convém notar que os altos custos associados à robotização não são devidos ao robô apenas, mas também devido aos assessórios e à programação para utilização do Robô.

Assessórios para robôs, como placas de I/O e comunicação são custam muito mais do que placas similares direcionadas ao mercado de PCs, mesmo considerando-se os efeitos das diferenças nas escalas de produção. Uma das razões para estes altos custos é a falta de uma arquitetura aberta para os controladores de robôs. Cada fabricante de robô possui seus protocolos e interfaces proprietários, forçando os usuários a adquirir todo o sistema do mesmo fabricante. Adicionalmente, o modelo de *marketting* utilizado pelos principais fabricantes de robôs desencoraja a montagem de um sistema robotizado em etapas.

Outro ponto a acrescentar dificuldades para a adoção de robôs por pequenas empresas é a programação. Atualmente, a maioria dos robôs industriais

são programados através de linguagens de programação de robôs que assemelham-se mais à linguagem Assembly de algum microprocessador do que à uma linguagem de programação moderna. Obviamente, o treinamento de programadores para estas linguagens representa uma porção significativa dos custos de instalação de um robô. Alem disso, não é incomum que a linguagem de programação mude de série para série de robôs de um mesmo fabricante, de forma que é bastante comum que uma empresa tenha que utilizar diversas linguagens de programação de robô.

Existem controladores comerciais para robôs que podem controlar robôs de qualquer fabricante [6]. Estes controladores eliminam a necessidade de utilizar-se diversas linguagens de programação. No entanto, a arquitetura de tais controladores não é aberta, impossibilitando a utilização do robô em aplicações que necessitem estratégias de controle avançadas. A linguagem de programação de tais controladores, denominada `RobotScript` [2] possa ser utilizada para todos os robôs suportados pelo controlador, independentemente do fabricante. Porém, `RobotScript` é baseada em `VBScript`, uma linguagem proprietária fortemente acoplada ao sistema operacional Windows. Além do Windows não ter sido desenvolvido para controle em tempo real, ele é conhecido por utilizar protocolos proprietários que são alterados com relativa freqüência. Portanto, o Windows não parece ser apropriado para servir como base para uma arquitetura aberta para controle de robôs.

Uma iniciativa acadêmica para gerar um padrão aberto para controle de robôs é o projeto OROCOS (Open RObot COntrol Software) [4]. This Este projeto está em seus estágios iniciais e ainda não produziu um protótipo funcional. É também um projeto mais voltado para a arquitetura de *software* de todo o sistema robótico e não enfoca de forma adequada a arquitetura de *hardware* necessária.

Arquiteturas abertas de *hardware* e *software* para controle de robôs foram definidas pelo projeto PINO [7], mas como foram desenvolvidas especificamente para pequenos robôs com pernas, também não parecem adequadas para manipuladores industriais ou outras classes de robôs.

Por outro lado, grandes companhias estão começando a substituir seus robôs velhos por novos. Devido aos custos muito menores estes robôs descartados são uma alternativa viável para a robotização de pequenas e médias empresas. No entanto, com estes robôs utilizam tecnologia antiga, alguns do benefícios associados ao uso de um robô podem ser perdidos.

Felizmente, a mecânica de robôs industriais não tem mudado muito. As principais diferenças os robôs antigos para os novos estão nos acionamentos das juntas e no controlador, incluindo o *software*. Este fato possibilita a atualização de robôs antigos para a tecnologia atual através do retrofitting do controlador do robô.

Este artigo descreva a experiência dos autores no retrofitting de um robô ASEA IRB6, incluindo o desenvolvimento de uma arquitetura aberta para controle de robôs. A arquitetura proposta não tem a pretensão de tornar-se uma arquitetura padrão para controle de robôs, mas deve servir para identificar algumas das exigências para uma arquitetura aberta padrão para controle de robôs.

O restante deste artigo está organizado com segue: A sessão II descreve a mecânica do robô ASEA IRB6 e as etapas percorridas para adaptá-la para suportar a arquitetura de controle proposta na sessão IV. A sessão III descreve as atualizações no circuito elétrico do robô, enquanto a sessão V descreve o *software* proposto para executar no robô atualizado. Experimentos com a nova arquitetura de controle são apresentados na sessão VI. Conclusões e direções para pesquisa futura são discutidos na sessão VII.

II Mecânica do ASEA IRB6

O ASEA IRB6, mostrado na figura 1, é um robô manipulador com cinco graus de liberdade construído em 1977. Como é um robô antigo, antes do *retrofitting*, ele foi completamente desmontado para identificação e verificação das peças. Algumas peças danificadas pelo uso prolongado foram substituídas. A figura 2 mostra algumas etapas da desmontagem do robô.

Um módulo atuador é mostrado na figura 3. Cada atuador é composto por um motor D.C, um *resolver*, um tacômetro e um interruptor para sincronismo. Os eixos 2 e 3, que estão sujeitos à gravidade possuem também freios eletro-mecânicos.

Detalhes do acoplamento entre atuador e junta são mostrados nas figuras 4 e 5. O acoplamento da junta 4, não mostrado, é semelhante ao acoplamen-

Figura 1: Robô ASEA IRB6.

Figura 2: Desmontagem do robô: Motores (2) e (3) removidos.

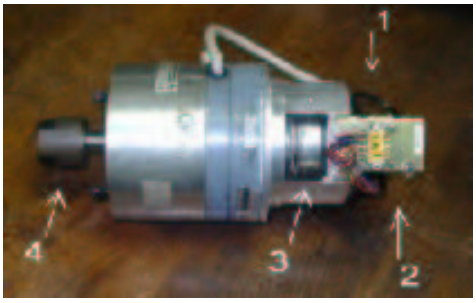

Figura 3: Módulo atuador: Resolver (1), conector (2), tacômetro (3) e saída de torque (4).

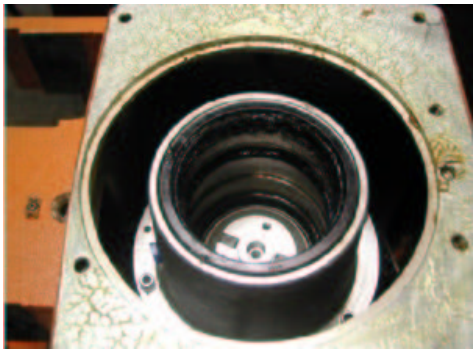

Figura 4: Transmissão da junta 1, o *harmonic drive* está sem o disco elíptico rígido.

to da junta 5.

Após a desmontagem, limpeza, substituição das peças danificadas e lubrificação, o robô foi remontado. Em geral pode ser observado que embora fosse um robô antigo as peças mecânicas estavam em boas condições.

III Atualização Elétrica

Ao contrário dos componentes mecânicos, os componentes elétricos do robô não estavam em boas condições. O controlador original do robô, visto na figura 6, não estava operando. Como de costume, no manual com o diagrama elétrico do robô a parte eletrônica é vista como uma *caixa-preta*, cujo circuito é proprietário.

Como a tecnologia do controlador era antiga e baseada principalmente em eletrônica analógica, substituiu-se o controlador por um novo baseada em uma arquitetura desenvolvida na UFRGS e des-

Figura 5: Fuso (5) e alavanca (3) de atuação da junta 2, alavanca de atuação da junta 3 (4) e acoplamento para o atuador da junta 5.

Figura 6: Controlador original do ASEA IRB6.

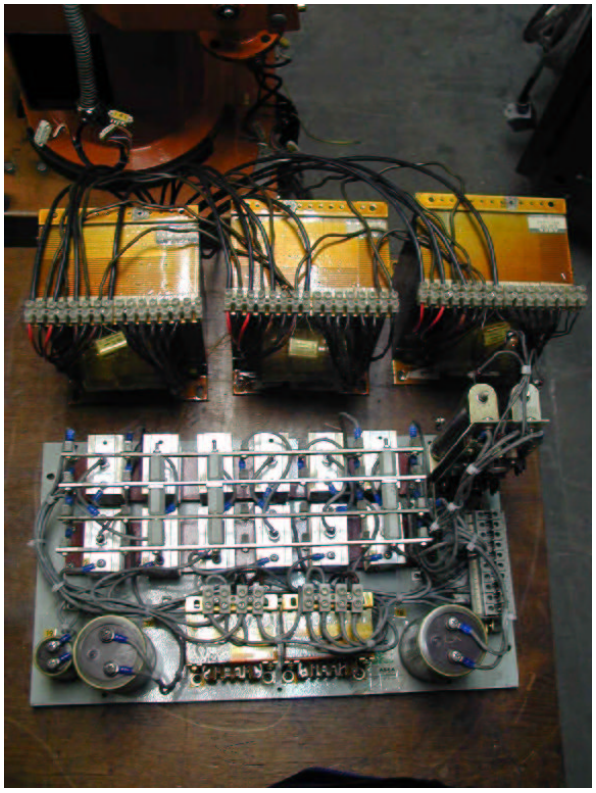

Figura 7: Fonte de alimentação modificada.

crita na sessão IV.

Originalmente os motores eram acionados através de *drivers* baseados em amplificadores lineares, necessitando de uma fonte de alimentação simétrica de $\pm 48V$. No entanto, na nova arquitetura, os motores são acionados por amplificadores chaveados utilizando PWM, necessitando apenas de uma fonte de alimentação simples de $+24V$. Reconfigurando-se as conecções $\triangle/Y$ e a utilização dos *taps* dos transformadores da fonte foi possível adequar-se a fonte às novas necessidades utilizando-se os mesmos componentes da fonte original. A Figure 7 mostra a fonte após as modificações.

Adicionalmente, como a nova arquitetura de controle é baseada em tecnologia digital, os *resolvers* e tacômetros foram substituídos por *encoders* incrementais com resolução de 2048ppr. Os interruptores para sincronismo foram mantidos de modo a ter-se um ponto de referência para os *encoders* incrementais.

Figura 8: Manipulador Janus.

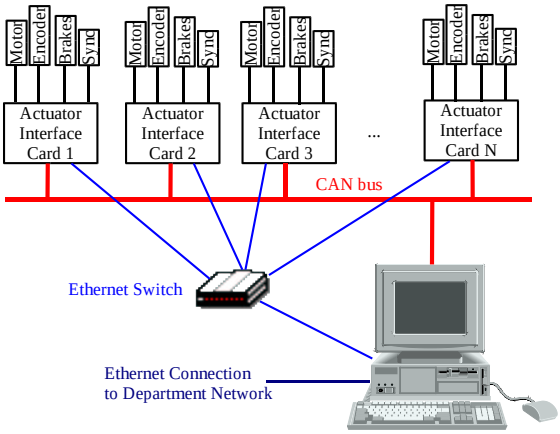

Figura 9: Arquitetura de controle.

IV Arquitetura de Controle

A arquitetura de controle utilizada no *retrofitting* do ASEA IRB6 não foi desenvolvida especificamente para este robô. Na realidade, ela foi desenvolvida para o robô Janus [5]. O Janus é um manipulador antropomórfico (figure 8) com dois braços e uma cabeça de visão stereo com movimentos de *pan-and-tilt*. Cada braço possui oito graus de liberdade com juntas acionadas por motores D.C. As juntas possuem ainda *encoders* incrementais, sensores indutivos para a posição de referência e freios eletromecânicos.

Embora a arquitetura tenha sido projetada para o robô Janus, nenhuma adaptação foi necessária para utiliza-la para o ASEA IRB6, indicando que a arquitetura é flexível bastante para ser utilizada com diferentes tipos de robôs. Como pode ser visto na figura 9 trata-se de uma arquitetura de processamento distribuído baseada em módulos denominados *Actuator Interface Card* (AIC). Cada junta possui uma AIC que aciona diretamente o atuador da junta e acessa os sensores associados. Os módulos AIC comunicam-se através de conexões CAN e Ethernet. Aparentemente o uso destes dois canais de comunicação é redundância, mas eles são utilizados para fins distintos. O barramento CAN é utilizado para dados de tempo real diretamente relacionados com o controle do robô, como leituras dos sensores e comandos para os atuadores, enquanto a conexão Ethernet é utilizada para dados de supervisão não relacionados diretamente com o controle do robô. Métodos para controle em tempo real através de conexões Ethernet estão sendo desenvolvidos, mas não estão maduros suficientes para serem utilizados como base para um controlador para robôs industriais.

A interface com o usuário executa em um computador PC compatível que comunica-se com as AICs através do barramento CAN e de uma conexão Ethernet. Este computador possui também uma segunda conexão Ethernet, atuando como *gateway* para a rede do Departamento e para a Internet.

O papel da AIC no controle do robô não é definido pela arquitetura além do ponto de que ela deve ser capaz de comandar o atuador da junta e ler os sensores associados à junta. A idéia é que cada AIC é um nodo em um sistema de controle distribuído e pode atuar como controlador de junta, implementando algoritmos de controle simples como PID ou atuar como um processador de entrada e saída, com o algoritmo de controle implementado no computador PC ou outro nodo conectado ao barramento CAN. Para suportar esta flexibilidade, o *software* executado por dada AIC pode ser carregado através da conexão Ethernet.

O computador executa uma variante do sistema operacional Linux com uma extensão para suporte a sistemas de tempo real denominada RTAI (Real Time Application Interface) [1]. Entre as vantagens do RTAI está o fato de que é possível desenvolver programas *hard real-time* no espaço do usuário (utilizando a API LXRT). Outras variantes de Linux

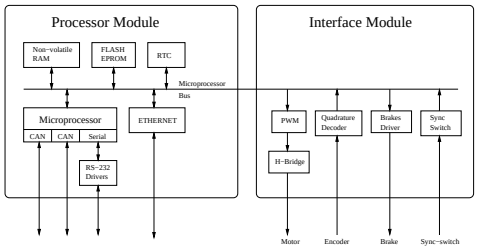

Figura 10: Diagrama de blocos da placa AIC.

para tempo real exigem que as tarefas de tempo real sejam desenvolvidas com módulos do *kernel*, restringindo as ferramentas que podem ser usadas e exigindo mais privilégios para o usuário. Para sistemas com muitos usuários não familiares com o Linux, isto é uma séria desvantagem.

Um diagrama de blocos da AIC é mostrado na figura 10. Existem dois módulos. O módulo processador possui um microprocessador, um relógio de tempo real e interfaces CAN, Ethernet e RS-232. O módulo de interface possui um soquete SIMM72 para o módulo processador, um PWM, um decodificador em quadratura e interfaces para o interruptor de sincronismo e os freios.

Como o módulo processador é um módulo SIMM72, ele pode ser facilmente substituído, sem a necessidade de reprojeto do módulo de interface, caso um processador mais potente venha a ser necessário. Para acomodar as necessidades de diferentes tipo de motores, a freqüência do PWM pode ser configurada por *software*. A saída do PWM está conectada a uma ponte H MOSFET que pode acionar diretamente a maioria dos motores D.C. utilizados em robôs industriais. O decodificador de quadratura decodifica o sinal de um *encoder* incrementa de dois canais, recuperando o sentido de movimento e quadruplicando a resolução do *encoder*. Este decodificador está diretamente acoplado à um contadores de 16 bits que armazena a contagem de pulsos entre leituras.

No *retrofitting* do ASEA IRB6 cada AIC foi montada em um gabinete individual, como mostrado na figura 11. No entanto, para aplicações onde um maior número de AICs seja necessário, como no manipulador Janus, elas podem ser montadas em um sub-bastidor padrão Eurocard.

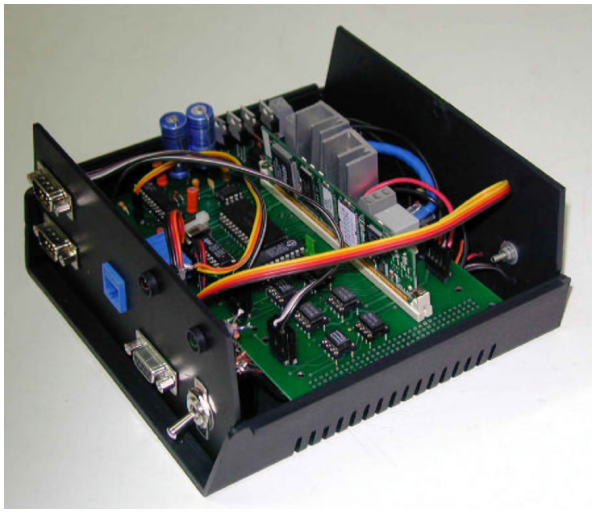

Figura 11: AIC no seu gabinete.

V *Software*

O módulo TINI [3], utilizado como processador das AICs, provê um ambiente de execução que inclui um sistema operacional multitarefa com suporta à gerenciamento de memória e entrada e saída, sistema de arquivos, uma pilha TCP/IP e uma máquina virtual Java. Um *shell* de sistema operacional semelhante ao UNIX, baseado em Java, também está disponível. O sistema operacional inclui servidores TELNET, FTP e de console serial, além de um cliente DHCP.

O *software* executando na AIC é baseado em JAVA. Um pacote Java (`br/ufrgs/eletro/AIC`) foi criado para modelar os dispositivos conectados a AIC: PWM, motor, *encoder*, interruptor de sincronismo e freio. Existem também classes para modelar a AIC como um todo e a interface com o *host*. Obviamente, cada classe possui métodos públicos para suportar as operações possíveis, por exemplo: `apply()` e `release()` para a classe `Brake` e `on()`, `off()` e `set(double voltage)` para a classe `Motor`. Embora todos os métodos públicos sejam acessíveis através de programas em Java, os métodos críticos foram implementados em Assembly como métodos nativos. A classe `Host` abstrai o canal de comunicação. As classes derivadas `HostCAN`, `HostTCP` e `HostUDP` encapsulam todos os detalhes da comunicação com o *host*.

O pacote AIC é utilizado para implementar o *da-*

emon de controle, que implementa um controlador de junta ou atua como um processador de entrada/saída, como discutido na sessão IV. Este *daemon* é carregado na AIC por FTP e é chamado pelos *scripts* de inicialização. Como a AIC possui memória não-volátil, imediatamente após energizada o *daemon* estará pronto para comunicar-se com o *host*.

Atualmente, o *retrofitting* do ASEA IRB6 está utilizando um *daemon* simples que comporta-se como um processador de entrada e saída. Ele é implementado como um programa Java com diversos ıthreads. O *thread* inicial trata os argumentos passados na linha de comando e dispara dois outros *threads*, um para enviar a leitura dos *encoders* a cada $10ms$ e outro para receber e processar comandos enviados pelo *host*, como tensão a ser aplicada nos motores.

No lado do *host* existe uma estrutura semelhante, com classes para modelar o motor, *encoder*, interruptor de sincronismo e freio, conectados à uma AIC hipotética e uma classe para modelar a AIC como um todo. Esta classe é ainda derivada para as classes `AIC_CAN`, `AIC_TCP` e `AIC_UDP` que encapsulam os detalhes de comunicação com as AICs. Portanto, o programador do *host* é isolado dos detalhes de comunicação. O programa do *host* pode ser desenvolvido como se os dispositivos conectados às AICs fossem locais. Um detalhe importante é que o lado do *host* é suportado por uma biblioteca de classes escritas em C++. Ou seja, apesar das classes e métodos serem similares aos existentes nas AICs, no *host* eles são implementados em C++.

Existem diversas razões para a implementação da biblioteca de suporte no *host* em C++ ao invés de Java. Inicialmente, C++ parece ser uma linguagem melhor do que Java para o desenvolvimento de técnicas avançadas de controle. A teoria de controle moderno e, em particular, o controle de robôs é fortemente baseado em álgebra matricial. A capacidade de sobrecarga de operadores de C++ possibilita o desenvolvimento de bibliotecas de manipulação de matrizes bastante conveniente. Por outro lado, Java não permite a sobrecarga de operadores, forçando um distanciamento maior entre a notação matemática das operações matriciais e a notação utilizada na programação.

O computador executa uma variante do sistema operacional Linux com uma extensão para suporte a sistemas de tempo real denominada RTAI (Real Time Application Interface) [1]. Este sistema não suporta a execução de aplicações Java em tempo real. Adicionalmente o padrão de Java para tempo real ainda não está suficientemente maduro, enquanto padrões para tempo real em C/C++, como as extensões ao POSIX, existem há um bom tempo e são suportadas pelo RTAI e diversos outros sistemas operacionais para tempo real.

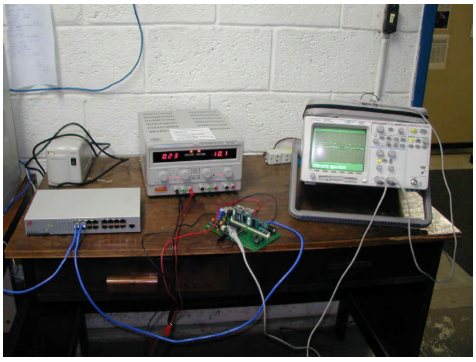

Figura 12: AIC sob teste.

VI Experimentos

A arquitetura proposta neste artigo tem sido extensivamente testada nos robôs ASEA IRB6 e Janus. A figura 12 mostra uma AIC sob teste. Note o sinal PWM na tela do osciloscópio.

Após o teste individual de cada AIC, elas foram montadas no gabinete do controlador do robô, que foi adaptado para receber a nova arquitetura de controle. O novo *lay-out* do gabinete do controlador pode ser visto nas figuras 13 e 14.

Uma seqüência de movimentos do robô pode ser vista nas figuras 15- 20.

VII Conclusão

Este trabalho descreve a experiência dos autores no *retrofitting* de um robô ASEA IRB6. Embora seja um robô bastante antigo foi verificado que as peças mecânicas estavam em boas condições, ao contrário da parte eletrônica, que foi substituída por uma nova arquitetura baseada em um sistema distribuído.

A arquitetura utilizada para substituir o controlador original do robô é bastante flexível. Ela foi concebida para utilização em um robô e não foram

Figura 13: Vista frontal do gabinete do controlador.

Figura 14: Vista traseira do gabinete do controlador.

Figura 15: Seqüência de movimentos do robô.

Figura 16: Seqüência de movimentos do robô.

Figura 17: Seqüência de movimentos do robô.

Figura 18: Seqüência de movimentos do robô.

Figura 19: Seqüência de movimentos do robô.

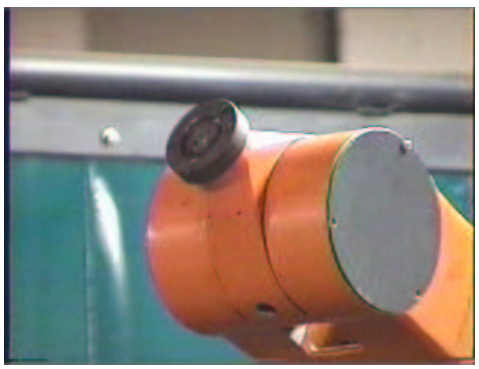

Figura 20: Seqüência de movimentos do robô.

necessárias modificações para utiliza-la no *retrofitting* do ASEA IRB6. Desenvolvimentos futuros para a arquitetura proposta incluem o desenvolvimento de uma implementação em FPGA e o desenvolvimento de novos módulos de interface para tratar atuadores com motores A.C. Atualmente está em desenvolvimento um módulo de interface para integrar sensores de força e torque na arquitetura.

Do ponto de vista de *software+*, o sistema proposto é baseado em Java e C++, que são linguagens de programação de uso geral e portanto o custo de treinamento de programadores deve ser bastante reduzido com relação ao de linguagens específicas para programação de robôs. Tipicamente, apenas a interface em C++ será visível para o usuário através de bibliotecas de classes que abstraem a arquitetura distribuída do sistema. Desenvolvimentos futuros incluem a integração do sistema com bibliotecas que tratem o controle de robô em um nível mais alto. Uma direção interessante é a integração da biblioteca C++ para controle do robô com *softwares* para programação *off-line* como o Workspace.

Agradecimentos

Este trabalho foi parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) através do Programa de Desenvolvimento Científico e Tecnológico em Informática (PROADI), processo Nº 01/05670, pela Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), através do processo Nº TEC 2471/98.

Referências

[1] 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.

[2] J. Lapham. RobotScript the introduction of a universal robot programming language. *Industrial Robot*, 26:17, 1999.

http://www.rwt.com/main/articles/ robots-cript.html.

[3] D. Loomis. *The TINI Specification and Developers Guide.* Addison-Wesley, Boston, MA, Jun 2001. http://www.ibutton.org/TINI/tinispec.pdf.

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

[5] 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.

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

[7] 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.