Em Evidência o Potencial e Limitações dos Compiladores NQC e BRICKOS e seus Respectivos Sistemas Operacionais

Douglas Machado Tavares [1]
Viviane André Antunes [2]
Luiz Marcos Garcia Gonçalves [2]

Resumo: Descrevemos algumas ferramentas disponíveis para operar e programar a unidade de controle (RCX) desenvolvida pela LEGO. Mostramos também, os resultados de uma análise comparativa realizada entre os compiladores NQC e BRICKOS e seus respectivos sistemas operacionais no tocante a desempenho, eficiência e usabilidade. Estas ferramentas foram desenvolvidas por *experts* usando engenharia reversa, seguindo o modelo de desenvolvimento *Open Source*.

Abstract: We describe operation and program development tools available for the processing unit (RCX) developed by LEGO. We also show results of a comparative analysis performed between the compilers NQC and BRICKOS and their operating systems including performance, efficiency, and usability. These tools were developed through reverse engineering by experts, and follow the open source development standards.

1 Introdução

Muitas das pesquisas na área de Inteligência Artificial integram percepção, raciocínio e ação [20, 12, 6, 25, 9]. Dentro deste contexto a robótica também se apresenta como uma área que proporciona uma ampla integração de metodologias ou técnicas diversas como percepção (sensores, fusão sensorial, visão computacional, dentre outros), meta-heurísticas (Algoritmos Genéticos, Computação Evolutiva, Inteligência Social, Redes Neurais, Lógica Difusa, dentre outros), navegação, planejamento inteligente, teoria de controle, dentre outras. Os mini-robôs móveis são exemplos atuais de plataformas viáveis que permitem o desenvolvimento de aplicativos e experimentos, além do aprimoramento de conhecimentos e a verificação da aplicabilidade destas integrações, proporcionando flexibilidade e baixo custo.

Dentre alguns dos robôs móveis comerciais utilizados em pesquisas podemos citar a linha de robôs da *K-Team*, como: o *Khepera*, o *K-alice* e o *Koala* [14], a linha de robôs

[1]Instituto de Computação (IC), UNICAMP, Caixa Postal 6176
13083-970 - Campinas - SP - Brasil
{douglas.tavares@ic.unicamp.br}
[2]Departamento de Engenharia de Computação e Automação (DCA), UFRN
59.072-970 - Natal - RN - Brasil
{viviane@engcomp.ufrn.br, lmarcos@dca.ufrn.br}

da *Activ Media Robotics*, como: o *Pioneer* 2-*DXe*, o *AmigoBot* e o *Pioneer* 2-*AT* [1] e os andarilhos da *Lynxmotion* [19].

Recentemente, os *kits* da linha LEGO *MINDSTORMS* [15], começaram a ser utilizados como uma boa opção de mini-robôs móveis para pesquisa e ensino, destacando-se pelo baixo custo, flexibilidade e pela grande velocidade de prototipagem. Convém ressaltar que os *kits* LEGO foram concebidos inicialmente para diversão educativa ou educação prática, visando principalmente atingir um público composto de crianças e adolescentes. As ferramentas fornecidas pelo fabricante junto com os *kits* provêem um controle muito rudimentar do potencial que o *hardware* contém, sem que seja possível uma exploração total dos recursos do mesmo. Em suma, devido a sua complexidade e seu potencial muito alto para a faixa etária a qual foi destinado e também ao aparecimento ou desenvolvimento de várias ferramentas de *software* para controle em nível um pouco mais baixo (usando linguagem 'C' ao invés da linguagem visual provida pelo fabricante), os *kits* passaram a ser mais usado nos ambientes de pesquisa e no ensino superior. Estas ferramentas de *software* surgiram principalmente a partir de re-engenharias (ou engenharia reversa) do *hardware* feitas pela comunidade científica, usando ambientes e código livre (open-source [22, 27]).

Assim, com o aparecimento dessas ferramentas, os *kits* da LEGO possibilitaram transformar um brinquedo educativo simples de adolescentes em um mini-robô adequado à pesquisa. Estes *kits* são compostos por pequenos blocos de plástico (peças tradicionais LEGO) e por outras peças como: motores, sensores de toque, de luz, de temperatura, de rotação, câmera de vídeo e engrenagens. Os *kits* possuem ainda uma unidade de controle programável chamada de RCX, a partir da qual pode-se fazer a leitura dos dados sensoriais ou gerar corrente para mover os motores (de forma independente) e uma torre de transmissão, que permite comunicação entre o computador e o RCX. Como exemplo destes *kits* podem ser citados o *kit* 9790 (ver Figura 1).

Neste trabalho apresentamos conceitos e a arquitetura da unidade de controle RCX, procurando ressaltar seu potencial e seus problemas. e uma uma análise dos dois compiladores de programas para RCX, mais usados pela comunidade de pesquisa, o NQC e o BRICKOS (outros existentes são apenas citados), bem como os respectivos sistemas operacionais alvos. Ao final, apresentamos análises comparativas entre eles, levando em consideração vários aspectos, como: potencialidade, velocidade, tamanho em memória, complexidade de aprendizado, dentre outros. Esta análise se baseia em inúmeros testes e implementações que realizamos nos últimos dois anos (incluindo mais de 200 horas de execução de programas), além de compilação de experiências coletadas, a partir de usuários dos compiladores e seus sistemas operacionais, atuais estudantes de Engenharia de Computação e de Ciência de Computação, à nível de pós-graduação (doutorado e mestrado) e de graduação, em disciplinas ministradas, usando os *kits* da LEGO, levadas a efeito nas Universidades Estadual de Campinas (UNICAMP), Federal de Mato Grosso do Sul (UFMS) e Federal do Rio Grande

Figura 1. *Kit* 9790 da linha LEGO *MINDSTORMS.*

do Norte (UFRN).

2 Motivação

Os *kits* com RCX vêm também sendo utilizados como ferramenta de Ensino em cursos do ensino superior (graduação, pós graduação, e também cursos técnicos). Podemos citar algumas disciplinas ministradas recentemente como: “Robótica: Sistema Sensorial e Motor” na UNICAMP e UFMS [10], “Percepção Robótica” na UFRN [11], “Sistemas Robóticos Autônomos” na UFRN [2]” e “Introdução à Engenharia de Controle e Automação” na UNICAMP [8].

Os alunos, usuários do RCX, se vêem necessitados de ferramentas de programação que lhes permitam explorar ao máximo o potencial do RCX e que lhes sejam familiares. Notamos isso principalmente nas disciplinas ministradas em cursos de Engenharia e de Ciência de Computação.

Podemos também citar inúmeros tópicos de pesquisas, que podem ser explorados com a utilização dessas ferramentas e do RCX, como por exemplo: autonomia robótica, navegação, inteligência computacional, aprendizado de máquina, sistemas multi-robôs, sistemas embarcados e uma série de outros.

Pelo exposto acima, torna-se necessário e cabível estudar o RCX bem como seus compiladores e seus sistemas operacionais, executar testes e comparações, analisar e apontar ferramentas que atendam aos requisitos acima. Neste trabalho serão abordadas as ferramentas

que utilizam a linguagem ‘C’ ou um dialeto baseado nesta linguagem. A razão desta escolha é pelo fato da linguagem ‘C’ ser bastante difundida dentre estes grupos de usuários.

3 RCX

O *Robotic Control eXplorer* (RCX), é um computador embarcado de propósito específico (uma unidade de controle), sua forma aproxima-se a de um paralelepípedo de dimensões $95mm$ x $63mm$ x $40mm$ (ver Figura 2). O RCX é um produto da linha LEGO *MINDSTORMS* [15], com finalidade de controlar os dispositivos elétricos (motores e sensores) de mini-robôs construídos com blocos de plástico da LEGO. A característica “embarcado” permite criar programas em um computador, compilá-lo e transferi-lo para o RCX, usando uma interface de comunicação (torre de transmissão). Assim, o robô pode executar o programa de forma autônoma, a partir do acionamento de um botão no RCX. Nesse caso, o robô opera sem a intervenção humana, característica essencial a um robô móvel autônomo.

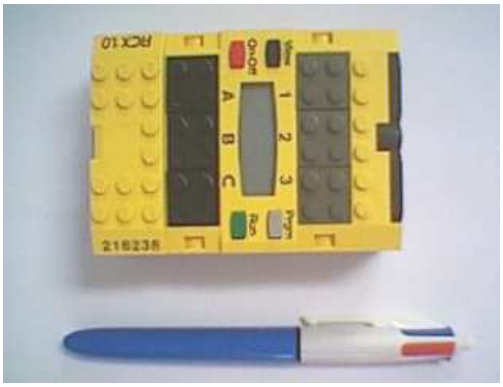

Figura 2. Unidade de Controle RCX.

Outro modo de operação também pode ser utilizado, este modo ocorre através de um protocolo de controle chamado de RemoteRCX [28]. Este protocolo permite que um programa principal possa operar no computador e enviar comandos a pequenos programas, chamados de interpretadores, os quais são executados nos robôs. Uma combinação de ambos pode ser interessante, por exemplo, no caso de se ter um mini-robô (um protótipo) como um guia de museu. Este robô pode ser requerido a executar uma tarefa comandada pelo computador. Por exemplo, se um usuário pede ao robô para se aproximar de um dado quadro, este comando é enviado pela interface de comunicação e o robô poderia então executa-lo. Em uma situação de risco (colisão com um obstáculo), um programa puramente embarcado pode

operar. Neste caso, apesar do robô estar executando comandos solicitados pela interface de comunicação, o programa embarcado teria maior prioridade de execução assumindo o controle. Após o desvio, o programa no computador retomaria o controle.

O RCX usa a arquitetura RISC (conjunto reduzido de instruções), possuindo 32 registradores de propósito geral, um microcontrolador *Hitachi*, mais especificamente o $H8/3292$, de $16MHz$ e palavra com 8 *bits* de tamanho. O microcontrolador é responsável por tarefas como o controle lógico, incluindo entrada e saída serial (serial I/O), conversão de analógico para digital e vice-versa, relógios internos dentre outros. A memória disponível no RCX é constituída por $16K$ de memória ROM e $32K$ de memória RAM, sendo esta última destinada a parte do sistema operacional e a programas do usuário. A quantidade de memória RAM total ($32K$) menos a quantidade ocupada pelo sistema operacional é toda a memória disponível, que pode ser manipulada pelos programas. A alimentação do RCX ocorre através de 6 pilhas do tipo AA (pequenas) de $1.5v$, ligadas em série ou através de uma entrada para conversores de energia.

Externamente, o RCX apresenta 3 saídas de tensões (A, B e C), onde podem ser conectados dispositivos eletrônicos (motores, buzinas, lâmpadas, dentre outros), 3 entradas (1, 2 e 3), onde podem ser conectados sensores (temperatura, toque, luz, dentre outros), uma tela de cristal líquido, onde serão exibidas pequenas mensagens e quatro botões (*On-Off*, *View*, *Prgm* e *Run*) (ver diagrama da Figura 3).

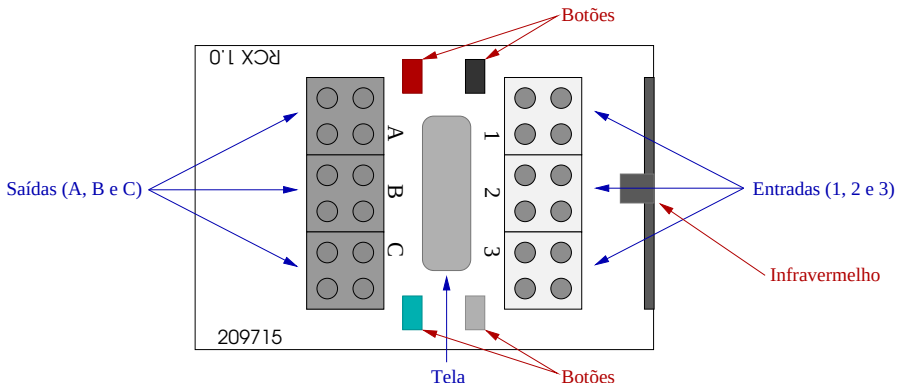

Figura 3. Diagrama Externo do RCX.

4 Sistemas Operacionais

O sistema operacional (SO) é uma das camadas de *software* mais importantes em sistemas computacionais. Ele não apenas gerência recursos como CPU, memória e periféricos, mas também estende a funcionalidade do *hardware*, para suportar *softwares* aplicativos. Ou seja, na prática, o SO é a interface entre os programas de usuário e o *hardware*, que é controlado pela parte do SO executado em baixo nível. O SO do RCX é dividido em duas partes. Uma destas fica armazenada na memória ROM, sempre presente, chamada de micro-programa e outra armazenada na memória RAM chamada de *firmware*.

A primeira parte, o micro-programa, é responsável por controlar as saídas, fazer leitura de sensores, controlar os relógios internos, controlar um pequeno *display* (tela de cristal líquido), controlar quatro botões para interação direta com o usuário, fazer transferência de dados, controlar o infravermelho para comunicação com um computador ou com outro RCX, controlar um alto falante, dentre outras. Tais funções são implementadas em um nível mais baixo. Essa parte do SO apresenta algumas semelhanças com o programa armazenado no *BIOS* de um computador pessoal (PC).

Já a segunda parte, o *firmware*, "pode" prover uma interface mais fácil com a primeira parte, permitir gerenciamento de memória (como paginação), implementar o controle de processos, implementar semáforos, implementar escalonador de processos, fornecer suporte ao fluxo do controle (como controle condicional e laços), permitir uso de operações básicas (uso da ULA), dentre outras funcionalidades. O *firmware* deve ser transferido para o RCX na primeira vez em que ele é ligado, após a troca de pilhas e quando a memória RAM for inicializada ("zerada").

Alguns programadores e grupos de usuários criaram novos *firmwares*, como o LEJOS [17], o BRICKOS [7] (nova versão do LEGOS), dentre outros, e conseqüentemente várias linguagens e compiladores (variantes de 'C', 'C++', Java, Pascal, Esterel, dentre outras). Estes *softwares* foram criados a fim de atender algumas das necessidades não supridas pelas ferramentas fornecidas pelo fabricante, através do uso de engenharia reversa e aproveitando-se do fato do *firmware* residir na memória RAM.

4.1 *Firmware* da LEGO

O *firmware* da LEGO, ou *firmware* padrão, é um SO simples, o qual basicamente é constituído por uma máquina virtual. Uma das vantagens da máquina virtual é a abstração da arquitetura, com isso, caso a arquitetura seja modificada, o *firmware* ainda poderá manter compatibilidade com programas escritos antes das modificações, perante nova implementação da máquina virtual. Outra vantagem é que a máquina virtual do RCX apresenta alta estabilidade.

Uma das principais desvantagens da máquina virtual é a redução da velocidade de execução (apesar da utilização de diversas técnicas de otimização) e a ocupação de uma área de memória a ela destinada, que poderia ser utilizada por outros programas. A tarefa básica da máquina virtual consiste em interpretar os *bytecodes* (códigos de programas), isto é, os *bytecodes* são reduzidos a uma série de instruções e dados apropriados para a máquina alvo, no caso o microcontrolador *Hitachi* $H8/3292$ do RCX. Os *bytecodes* são constituídos pelo *opcode* (código da operação) seguido por zero ou mais operandos. O *opcode* indica a ação a ser tomada. Se mais informações forem requeridas, estarão codificadas em um ou mais operandos os quais seguem imediatamente o *opcode*.

Atualmente, existem duas versões do *firmware* da LEGO para o RCX chamadas de RCX1 e RCX2. Na versão RCX2 foram feitas diversas melhorias na máquina virtual. Tais melhorias permitem funcionalidades extras aos programas, isso sem perder a compatibilidade com programas escritos anteriormente para o RCX1. Dentre os compiladores que geram *bytecode* para a máquina virtual do *firmware* da LEGO destacam-se o Robolab, o RIS e o NQC.

4.2 *Firmware* BRICKOS

O BRICKOS [7] (nova versão do LEGOS [16]) é um projeto – código fonte aberto. Ele é um SO preemptivo, padrão *POSIX*, multi-tarefa escrito para o RCX. Implementa gerenciamento de memória (como paginação), controle de processos, comunicação inter-processos, semáforos, escalonador de processos com prioridade, suporte ao fluxo do controle e controle total sobre o *display*, dentre outras funcionalidades. Além disso, o BRICKOS permite amplo uso da memória, sendo que o limite para o número de variáveis, de funções, de processos e de sub-rotinas é o próprio tamanho da memória ($32K$).

Atualmente existem compiladores para 'C'/'C++' (compilador BRICKOS) e para Pascal (em fase de teste), que geram programas executáveis para o SO BRICKOS. Notam-se que esses compiladores, na realidade, suportam um 'dialeto' de 'C'/'C++' e de Pascal, isto é, eles suportam um conjunto restrito de funções. Além dos compiladores, há também um simulador (emulegOS) e um protocolo de troca de mensagens chamado de LNP.

4.2.1 emulegOS O emulegOS é uma ferramenta desenvolvida para simular parcialmente o *hardware* do RCX, executando o SO BRICKOS. Seu principal objetivo é propiciar um ambiente mais confortável para testar e eliminar erros dos programas em desenvolvimento. O emulegOS é um *software* inicialmente escrito para o SO *Linux*, o qual interpreta códigos 'C'/'C++', escritos para o BRICKOS, emulando o comportamento destes códigos, em uma janela (ver Figura 4), tal qual ocorreria em um sistema real.

Algumas características principais:

Figura 4. Simulador para RCX (emulegOS).

- a interface visual é bem simples, permitindo que o usuário configure e interaja com os sensores, com o programa em funcionamento, para simular eventos externos, além de mostrar os estados atuais dos motores (virtualmente) conectados às saídas A, B, e C;
- existência de uma camada (API) para emular as rotinas do SO BRICKOS. Desta forma, a maioria do código é executado numa máquina virtual, simulando o BRICKOS no *Linux*, incluindo a sustentação multi-tarefa e troca de mensagens;
- "emulação do mundo real", o emulegOS simula algumas características mecânicas de um robô, como um sensor de rotação girar, quando um motor funcionar ou um motor ligar 'x' segundos após um sensor de toque ser tocado, dentre outras;
- ganho de tempo e facilidade na eliminação de erros.

4.2.2 LNP O *LegOS Networking Protocol* (LNP) [18] é um protocolo de troca de mensagens presente no *kernel* do SO BRICKOS. Ele permite que dois ou mais RCX troquem mensagens de texto empacotadas. Os pacotes são transmitidos através de uma codificação, usando luz infravermelha. O protocolo LNP possui dois tipos de pacotes: um chamado de *LNP-integrity* e outro chamado de *LNP-addressing*.

O pacote *LNP-integrity*, garante que os dados foram transferidos sem alteração de seus valores, mantendo a integridade da mensagem. Já o pacote *LNP-addressing*, simplesmente adiciona endereçamento ao primeiro, de forma que o pacote passa a ser direcionado.

O LNP também foi implementado para computadores pessoais. A implementação desse protocolo é chamado de LNPd e consta basicamente de um *daemon* (um tipo de

software) e uma biblioteca, que possui a mesma interface da biblioteca lnp do compilador BRICKOS, ou seja, com os mesmas funções, porém ela pode ser utilizada por um compilador como por exemplo o 'gcc' no SO *Linux*, possibilitando assim, que *softwares* escritos na linguagem 'C', executados em um computador, troquem pacotes com um ou mais RCX. O *daemon*, também de nome LNPd, controla a torre de infravermelho conectada à porta PS2 ou serial do computador, tratando colisões e gerenciando a fila de pacotes.

5 Compiladores

O nome compilador, criado nos anos 50, significa um programa, cuja função principal é o processo de tradução de uma linguagem fonte para uma linguagem objeto. Pode-se dizer que o compilador traduz um programa descrito em uma linguagem de alto nível, para um programa equivalente em código de máquina de um processador (baixo nível). Os *cross-compilers* são compiladores, que são executados em uma determinada arquitetura, mas que geram códigos para serem executados em uma outra arquitetura alvo. Os compiladores existentes para o RCX são na verdade todos *cross-compilers*, mas para simplificação serão denominados simplesmente de compiladores. A seguir serão discutidos alguns desses compiladores.

5.1 O Compilador Robolab

O Robolab [26] é um compilador integrado a um ambiente de programação visual fornecido pela LEGO e desenvolvido para a plataforma *Windows*. Ele gera código para ser interpretado pela máquina virtual do *firmware* da LEGO.

Utilizar o Robolab é relativamente fácil, nele a programação é baseada em ícones ou componentes (ver Figura 5), ou seja, a programação é feita dispondo esses componentes em uma seqüência lógica e conectando-os através de fios (ver Figura 6).

O Robolab foi inspirado no *LabVIEW* (um produto da *National Instrument*), ele é bem didático e possibilita a usuários sem conhecimento profundo de programação desenvolverem programas para controlar o RCX com facilidade e rapidez. Porém, essa simplicidade acaba limitando todo o potencial oferecido pelo RCX que pode ser explorado. As linguagens gráficas são freqüentemente mais fáceis de aprender (nenhum erro de sintaxe), mas são geralmente mais tediosas de usar do que uma linguagem textual. Ainda, as metáforas gráficas de codificação dessas linguagens podem limitar significativamente os tipos de programas, que podem ser escritos, bem como a utilização completa de todo o potencial do *hardware*.

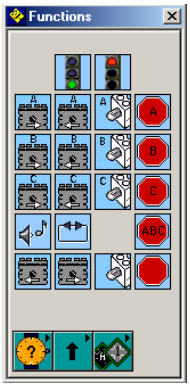

Figura 5. Alguns Componentes Disponíveis no Robolab.

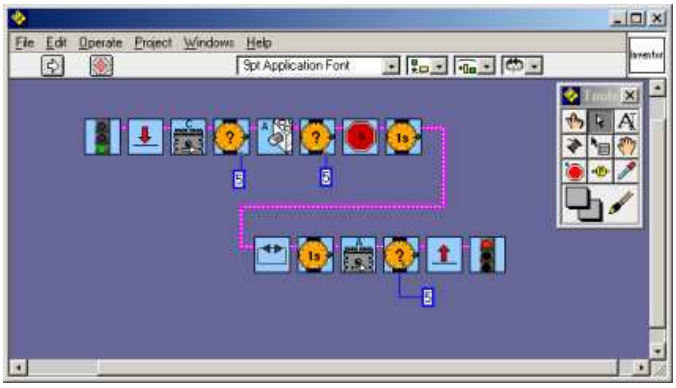

Figura 6. Um Programa Desenvolvido no Robolab.

5.2 O Compilador NQC

O "Not Quite C" (NQC) [5, 4, 3, 23] é um compilador para uma linguagem simples com sintaxe semelhante à linguagem 'C'. Pode-se dizer que sua linguagem é um dialeto do 'C'. O compilador NQC gera códigos para as máquinas virtuais presentes nos *firmwares* da LEGO para RCX, *Scout* e *CyberMaster*, produtos da linha LEGO *MINDSTORMS*.

O compilador NQC é desenvolvido e mantido sem qualquer suporte da LEGO. Tendo Dave Baum como idealizador e principal mantenedor. O NQC é um *software* livre licenciado pela *Mozilla Public License* (MPL) [21]. Possui versões oficiais até o momento, para os sistemas operacionais *Windows* (95/98/ME e NT 4,0) e *Mac* X, bem como versões não oficiais para *Linux*, *Solaris*, *FreeBSD*, *WinCE* e agendas *PDA*. Devido ao fato da sintaxe do NQC ser muito similar à linguagem de programação 'C', a tarefa de programar torna-se fácil para programadores com pequena experiência em 'C'. Como o NQC gera código para os *firmwares* da LEGO, significa que é possível armazenar no RCX programas, que foram gerados tanto pelo Robolab quanto pelo NQC.

A vantagem do NQC gerar código para os *firmwares* da LEGO é que ele se beneficia da estabilidade oferecida por esses *firmwares*. No entanto, os programas gerados pelo NQC estarão sujeitos a restrições impostas pelos *firmwares* da LEGO. Por exemplo, se o *firmware* não fornece suporte a ponto flutuante, o NQC também não poderá fornecê-lo. Outras alternativas de compiladores sem tais limitações são as que geram código para outros *firmwares*, como os: BRICKOS, LEJOS e o pbForth.

A interface do NQC permite compilação apenas através de linha de comando. Mas

existem programas que fornecem ambientes integrados para construir, compilar, carregar (baixar) programas e receber dados, utilizando o NQC, sendo que o mais conhecido deles é o RCXCC.

5.2.1 RCXCC O RCX *Comand Center* (RCXCC) é um ambiente de programação, desenvolvido para a plataforma *Windows* (95/98 e NT) [24]. A janela principal do RCXCC (ver Figura 7) é um *front-end* para o NQC. O editor possui a ferramenta *color-code*, que destaca palavras reservadas da linguagem, além de possibilitar a edição de vários arquivos de uma vez. Desse editor, pode-se invocar o compilador, bem como invocar o programa que transmite o código para execução ao RCX.

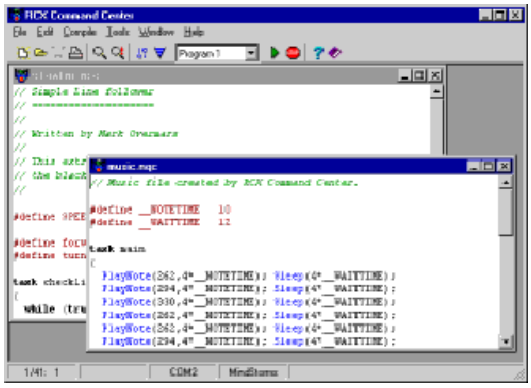

Figura 7. Janela Principal do RCXCC.

Dentre as propriedades mencionadas, o RCXCC possui ainda uma janela que permite monitorar o que está ocorrendo com recursos do RCX, como memória e sensores em tempo real. Nesta janela é possível controlar o robô diretamente ou através de *joystick*. Janelas de diagnóstico e de composição de músicas para o RCX também estão disponíveis (ver Figura 8). A versão mais recente possui uma documentação *on-line* completa tanto do RCXCC como da linguagem do NQC.

5.3 O Compilador BRICKOS

O BRICKOS [7] é um compilador para linguagem 'C'/'C++', que contém uma parte das bibliotecas padrão do 'C' implementadas e suporta programação orientada a objetos, utilizando 'C++'. Com isso, basicamente qualquer programa em 'C'/'C++', que seja pouco menor que $32K$ e com algumas restrições de sintaxe devido ao *hardware* dedicado podem ser

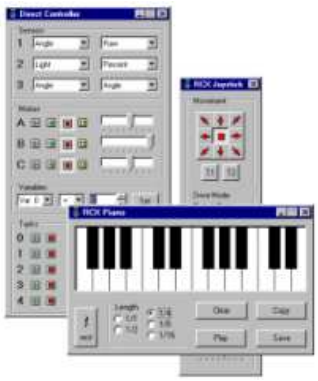

Figura 8. Outras Ferramentas do RCXCC.

escritos, utilizando tais bibliotecas e compilados pelo BRICKOS. Desta forma, o BRICKOS torna-se uma boa opção para programadores de 'C'/'C++' que desejam ingressar na programação do RCX sem a necessidade de aprendizado de uma nova linguagem.

O compilador BRICKOS gera códigos nativos para serem executados no *firmware* BRICKOS. Assim como o compilador NQC, o compilador BRICKOS e o *firmware* BRICKOS são desenvolvidos sem qualquer suporte da LEGO. Eles são *softwares* livres licenciados pela *GNU General License Public* (GPL) [13]. No momento, possui versão oficial somente para *Linux*, sendo possível recompilá-lo para executar sob o *Windows*.

Como o alvo dos programas compilados pelo compilador BRICKOS é o *firmware* BRICKOS, tais programas se beneficiarão de todas as facilidades oferecidas pelo *firmware* BRICKOS, como: ponto flutuante emulado, semáforos, utilização de *array* (vetores) com capacidade limitada somente pela memória, número de variáveis globais e locais limitado somente pela memória, dentre outras. Essas características tanto do *firmware* quanto do compilador BRICKOS fazem deste pacote de *software*, um dos mais poderosos, dentre os pacotes alternativos destinado ao RCX.

6 Testes e Comparações

Para efetivar a análise comparativa entre os compiladores NQC e BRICKOS e seus respectivos SO, realizamos alguns testes de instalação, verificando potencialidades das linguagens e o desempenho dos programas por eles gerados.

Para os testes de instalações do NQC utilizamos os SO *Windows* (95 e 98) e *Linux*. No *Windows*, instalamos a versão nqc-2.1r1, juntamente com o RCXCC, versão rcxcc-3.1, os quais mostram-se relativamente fáceis, bastando descompactar e utilizar. A versão nqc-2.2.r2 para *Linux*, também mostrou-se muito fácil de instalar, bastando utilizar comandos como: `./configure e ./make` e modificar permissões de escrita na porta de conexão da torre de infravermelho.

Quanto aos testes de instalação do BRICKOS, fizemos apenas no SO *Linux*, nas versões brickos-0.2.6.10, legos-0.2.6 e legOS-0.2.5. Exigindo mais tempo para instalar o *cross-compiler* do microcontrolador *Hitachi* H8/3292, essencial ao BRICKOS.

Além dos testes de instalação, escrevemos um total de 40 programas para ambos os compiladores, com o objetivo de descobrir as potencialidades de cada uma das linguagens, tais como: comandos para controle de motores, leitura dos sensores, gerência de processos e semáforos. Testamos ainda o LNP exaustivamente, somando um total de mais de 200 horas de testes.

Para finalizar os testes, idealizamos quatro objetivos, com diferentes níveis de dificuldade, a serem cumpridos por um robô com o RCX "embarcado". Cada Objetivo foi implementado em ambos os compiladores com a finalidade de comparar o desempenho dos programas gerados. Os objetivos propostos são que um robô:

1. pare assim que detectar qualquer obstáculo;
2. navegue sobre uma mesa, detectando e desviando-se das bordas da mesma;
3. tenha dois comportamentos básicos: desviar de obstáculos e seguir duas linhas no chão;
4. resgate objetos de isopor, de uma cor previamente determinada, localizados em uma pequena arena retangular.

Nos Objetivos 1 e 2, os quatro programas gerados, para cumprí-los, tiveram aproximadamente o mesmo tamanho e seus respectivos códigos tiveram o mesmo número de linhas. Constataram, ainda, que os mesmos foram de fácil implementação e atingiram os objetivos da forma esperada.

Quanto ao Objetivo 3 (ver Figura 9), devido ao BRICKOS utilizar semáforos e tratar multi-tarefa de forma muito elegante, a implementação ficou extremamente fácil e trivial. O contrário ocorreu com o NQC.

O Objetivo 4 foi pensado a partir de um exercício proposto para as disciplinas: "Robótica: Sistema Sensorial e Motor", na UNICAMP e UFMS [10], e "Percepção Robótica", na UFRN

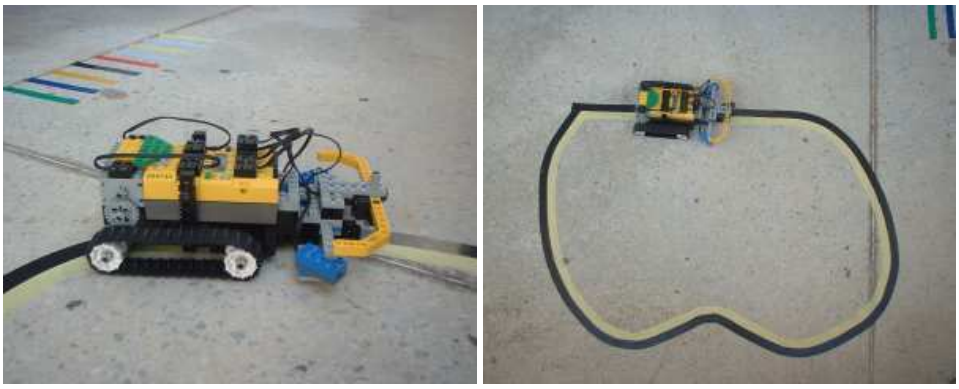

Figura 9. Robô Seguidor (Objetivo 3).

[11]. Tal exercício é uma competição entre dois robôs, onde vence o robô que resgatar mais objetos de isopor, de uma cor previamente determinada, localizados em uma pequena arena retangular. De posse dos códigos fonte, escrito pelos alunos das referidas disciplinas, foram escritos dois novos códigos: um para o NQC e outro para BRICKOS, e construídos dois robôs idênticos. Ficou constado que os códigos gerados pelo BRICKOS para esse Objetivo ficaram menores do que os gerados pelo NQC, como esperado, devido ao NQC gerar código para uma máquina virtual, o que não ocorre com o BRICKOS. Também observou-se que o programa escrito para o BRICKOS privilegiou-se da capacidade do mesmo em suportar maior número de variáveis, bem como vetores de maiores dimensões. Após realizadas as competições os dois robôs obtiveram números de vitórias parecidas (12 para o robô com código gerado pelo BRICKOS e 8 para o robô com código gerado pelo NQC). No ano de 2003, essa competição foi incorporada ao *Latin American IEEE Student Robottics Competition* e chamada de *Lego Rescue*.

6.1 Comparando os *firmwares*

Algumas comparações entre os *firmwares* da LEGO e BRICKOS são apresentadas na Tabela 1, que elucida características e limitações dos mesmos.

Da Tabela 1, podem ser observadas inúmeras vantagens do BRICKOS sobre o RCX1 e RCX2, tais como: emulação de ponto flutuante, maior velocidade de execução (programa executado em código nativo), possibilidade de maior número de sub-rotinas, de variáveis globais, de tarefas, de *loops* (laços) aninhados, de programas. As duas principais vantagens que podem ser destacadas, são: o suporte do BRICKOS aos semáforos e paginação de

Tabela 1. Comparando os *Firmwares.*

	RCX1	RCX2	BRICKOS
Programas	interpretado	interpretado	nativo
Paginação de memória	não	não	sim
Emulação de ponto flutuante	não	não	sim
Multi-tarefa	sim	sim	sim
N^o de tarefas	10	10	a memória é o limite
N^o de sub-rotinas	8	8	a memória é o limite
N^o de variáveis globais	32	32	a memória é o limite
N^o de variáveis locais/tarefa	0	16	a memória é o limite
N^o de *loop* aninhados	4	4	a memória é o limite
N^o de programas armazenados	5	5	8
Variáveis compartilhadas	não	sim	sim
Monitoramento de eventos	não	sim	sim
Suporte nativo a semáforos	não	não	sim
Licença	proprietária	proprietária	GPL

memória, sem a qual não seria possível aumentar a quantidade de variáveis, de tarefas, dentre outras. Em implementações que exijam bastante memória para armazenar tabelas de estado, como, por exemplo, em algoritmos de aprendizado por reforço, o BRICKOS se torna a opção viável e plausível, já que o mesmo implementa paginação de memória. Algumas deficiências do RCX1, já superadas no RCX2, são o compartilhamento de memória entre processos e o monitoramento de eventos. O fato do RCX1 e RCX2 interpretarem o código fonte pode significar alguma vantagem sobre o BRICKOS, ou seja, o mesmo programa pode executar em várias arquiteturas diferentes. Isso pode ser útil por exemplo, no caso em que o *hardware* seja modificado pelo fabricante. A limitação de memória no RCX1 e RCX2 deve-se ao fato de terem sido desenvolvidos para um público alvo nada exigente ou que não precisa de tantos recursos de programação.

Com base em testes, em experimentos realizados e em análises obtidas no decorrer das disciplinas citadas na Seção 6, constatou-se que os *firmwares* da LEGO apresentaram alta estabilidade, enquanto o *firmware* BRICKOS ficou inoperante por dezesseis vezes. Isso pode ser significante na escolha da plataforma se estabilidade for uma condição básica. Verificamos que tanto o RCX2 quanto o BRICKOS conseguem imprimir o valor de variáveis no *display*, o que é essencial na depuração de programas.

6.2 Comparando os Compiladores

Comparações entre os compiladores são apresentadas na Tabela 2 e na Tabela 3.

Tabela 2. Comparando os Compiladores.

	NQC	BRICKOS
Orientação a Objetos	não	sim
Bibliotecas padrão 'C'	não	sim
Velocidade de compilação	boa	boa
Arquitetura alvo	4	1
Licença	Mozilla	GPL

A Tabela 2 mostra uma comparação entre os compiladores NQC e BRICKOS, apontando algumas características e limitações importantes. Nesta, podemos ver que uma das vantagens do BRICKOS é a implementação de algumas das bibliotecas padrões da linguagem 'C' (`stdio, conio, math, etc.`), o que facilita a programação e o suporte à orientação a objetos. O BRICKOS não gera código executável para outras plataformas que não o RCX com o SO BRICKOS. Isso não vem ao caso, uma vez que estão sendo discutidas ferramentas para o RCX. Quanto às licenças, GPL provê mais liberdade de mudanças, o que pode ser uma vantagem.

Quanto a compilação, no BRICKOS é um pouco mais lenta, o que não acarreta em problemas, uma vez que isso é feito *off-line* (antes do robô entrar em execução). Podemos considerar ambas compatíveis. Em compensação, a transferência de programas do computador para o RCX é mais rápida com o BRICKOS.

Tabela 3. Pesquisa.

	NQC	BRICKOS
Instalação	★★★★	★★★
Usabilidade	★★★	★★★★
Disponibilidade Documentação	★★★★	★★
Aprendizado	★★★★	★★★★★
Estabilidade	★★★★	★★

A Tabela 3 compara os compiladores com base em informações coletadas a partir de amostragem feita com os alunos das disciplinas citadas na Seção 6. Os resultados mostrados (estrelas) foram obtidos a partir da média ponderada de respostas aos questionários por mais

de 50 alunos: categorizadas como fácil, normal e difícil.

Os ítens analisados foram:

- instalação: grau de dificuldade encontrado pelo usuário para instalação dos compiladores;
- usabilidade: esse ítem se refere a facilidade de uso dos compiladores, bem como outros *software* de apoio, como: emuladores, editores, ambiente de programação dentre outros;
- disponibilidade de documentação: número e facilidade de documentações para as linguagens e para os compiladores;
- aprendizado: nível de facilidade para o aprendizado das linguagens suportadas pelos compiladores;
- estabilidade: estabilidade dos *softwares* gerados pelos compiladores.

7 Conclusões

Constamos que o NQC foi o preferido pelos alunos das disciplinas citadas na Seção 6, quanto aos critérios instalação, usabilidade, disponibilidade de documentação e estabilidade. Fato justificado pelo grau de dificuldade dos exercícios propostos, que nessas disciplinas não exigia alto desempenho tanto do *hardware* quanto do *software*. Por outro lado, o aprendizado do BRICKOS foi o quesito melhor apontado, devido à sua proximidade da linguagem 'C'/'C++'.

A partir das análises realizadas neste trabalho, não podemos generalizar, dizendo que o BRICKOS seja o melhor em todos os aspectos que o NQC. Porém, analisando as tabelas comparativas, a pesquisa de campo e os testes realizados, podemos, "*hoje*", destacar o BRICKOS como opção mais apropriada para trabalhar com o RCX em pesquisas. Seu uso por programadores experientes pode compensar sua falta de estabilidade. Além disso, falta de estabilidade é um preço que se paga pelas características essenciais à pesquisa (paginação, mais memória, dentre outras).

Finalmente, podemos destacar o LNP como sendo o fator diferencial na escolha do BRICKOS, o qual permitiu várias implementações, destacando: o protocolo de controle RemoteRCX [28], controladores para os robôs usando este protocolo de controle e utilização de robôs como *avatares* em um ambiente real para usuários conectados via *Internet* num ambiente virtual [30, 29].

Referências

[1] Activ - ActivMedia Robotics Corporate, November 2002. Available at `http://www.activrobots.com`.

[2] Pablo Javier Alsina. Disciplina 'Sistemas Robóticos Autônomos', Engenharia de Computação - UFRN, 2002. Disponível em `http://www.dca.ufrn.br/~pablo/sisrob.html`.

[3] Dave Baum. *NQC Programmer's Guide*. Version 2.2 r1.

[4] Dave Baum. *NQC User Manual*. Version 2.3 r1.

[5] Dave Baum. NQC - 'Not Quite C' is a simple language with a 'C'-like syntax that can be used to program Lego's RCX programmable brick (from the Mindstorms set), November 2002. Available at `http://www.baumfamily.org/nqc`.

[6] Reinaldo A. C. Bianchi and Anna Helena Reali-Costa. O Sistema de Visão Computacional do time FUTEPOLI de Futebol de Robôs. In *Congresso Brasileiro de Automática*, 13, pages 2156–2162, Florianópolis, 2000. UFSC / Sociedade Brasileira de Automática. Anais.

[7] BrickOS - An open source embedded operating system and provides a 'C' and 'C++' programming environment for the Lego Mindstorms Robotics Kits, November 2002. Available at `http://brickos.sourceforge.net`.

[8] João Vilhete Viegas D'Abreu. Disciplina 'Introdução à Engenharia de Controle e Automação', Engenharia de Controle e Automação (Mecatrônica) - UNICAMP, 2001.

[9] Gedson Faria and Roseli Aparecida Francelin Romero. Explorando o Potencial de Algoritmos de Aprendizado com Reforço em Robôs Móveis. In *IV Congresso Brasileiro de Redes Neurais*, pages 237–242, São José dos Campos - SP, 1999. ITA.

[10] Luiz Marcos Garcia Gonçalves. Disciplina 'Robótica: Sistemas Sensorial e Motor', Ciência da Computação - UNICAMP, 2001. Disponível em `http://www.ic.unicamp.br/~lmarcos/courses/mo810`.

[11] Luiz Marcos Garcia Gonçalves. Disciplina 'Percepção Robótica', Engenharia de Computação - UFRN, 2002. Disponível em `http://www.dca.ufrn.br/~lmarcos/courses/robotica`.

[12] Luiz Marcos Garcia Gonçalves and Fernando W. Silva. Control Mechanisms and Local Perception to Support Autonomous Behavior in Virtual Animated Agents. *Computer Graphics Journal*, March 2002.

[13] GPL - GNU GENERAL PUBLIC LICENSE, March 2003. Available at `http://www.gnu.org/copyleft/gpl.html`.

[14] K-Team S.A., November 2002. Available at `http://www.k-team.com`.

[15] LEGO Mindstorms, November 2002. Available at `http://mindstorms.lego.com`.

[16] LegOS - A open source embedded operating system and provides a 'C' and 'C++' programming environment for the Lego Mindstorms Robotics Kits, November 2002. Available at `http://legos.sourceforge.net`.

[17] LejOS - Java for the RCX, November 2002. Available at `http://lejos.sourceforge.net`.

[18] LNP - LegOS Network Protocol, November 2002. Available at `http://legos.sourceforge.net/HOWTO/x405.html`.

[19] Lynxmotion, Inc, November 2002. Available at `http://www.lynxmotion.com`.

[20] Maja J. Matarić. Reinforcement Learning in the Multi-Robot Domain. *Autonomous Robots*, 4(1):73–83, March 1997.

[21] MPL - Mozilla and Netscape Public Licenses, March 2003. Available at `http://www.mozilla.org/MPL`.

[22] Open-Source - Initiative (OSI) is a non-profit corporation dedicated to managing and promoting the Open Source Definition for the good of the community, November 2002. Available at `http://www.opensource.org`.

[23] Mark Overmars. *Programming Lego Robots using NQC*. Department of Computer Science Utrecht University, TB Utrecht the Netherlands, October 1999. Version 3.03.

[24] RcxCC - A front-end for NQC, November 2002. Available at `http://www.cs.uu.nl/people/markov/lego/rcxcc`.

[25] Carlos Henrique Costa Ribeiro, Anna Helena Reali-Costa, and Roseli Aparecida Francelin Romero. Robôs Móveis Inteligentes: Princípios e Técnicas. In *Jornada de Atualização em Inteligencia Artificial*. Congresso da Sociedade Brasileira de Computação, 2001.

[26] ROBOLAB - Software for programming and controlling the RCX is an icon-based, November 2002. Available at `http://www.lego.com/eng/education/mindstorms`.

[27] SourceForge - The world's largest Open Source software development web site, providing free hosting to tens of thousands of projects, November 2002. Available at `http://sourceforge.net`.

[28] Douglas Machado Tavares. Ferramentas Computacionais para Robôs Móveis Autônomos. Master's thesis, Universidade Estadual de Campinas - UNICAMP, 2004.

[29] Douglas Machado Tavares, Aquiles Medeiros Burlamaqui, Viviane André Antunes, George Christian Basilio Thó, Anfranserai Morais Dias, Tatiana Aires Tavares, Guido Lemos Filho, and Luiz Marcos Garcia. A Real Time Platform for Playing with Robots and Avatars in Mixed Reality Spaces. In *IV Workshop de Tempo Real*, Natal-RN, Brasil, March 2003. 21th SBRC2003.

[30] Douglas Machado Tavares, Aquiles Medeiros Burlamaqui, Anfranserai Morais Dias, Meika Iwata Monteiro, Viviane André Antunes, George Christian Basilio Thó, Tatiana Aires Tavares, Carlos Magno de Lima, Luiz Marcos Garcia Gonçalves, Pablo Javier Alsina, Adelardo A. Dantas de Medeiros, and Guido Lemos Filho. Hyperpresence - An Application Environment for Control of Multi-User Agents in Mixed Reality Space. In *Conference held at Orland*, Florida, USA, March 2003. 36th Annual Simulation Symposium.