Estudo de Caso Usando o *Framework* Robot Operating System (ROS)

Iohan G. Vargas[1], Dalton Matsuo Tavares[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Goiás (UFG)
Avenida Dr. Lamartine Pinto de Avelar, 1120, Setor Universitário, Catalão – GO – Brasil

iohanufg@gmail.com, dalton.tavares@catalao.ufg.br

Abstract. *This paper briefly describes the ROS framework and its functionalities, presenting its strengths and demonstrating its usability in a practical application, whose goal is to read the touch sensor of the LEGO® Mindstorms® Education NXT 9797 kit. By means of a rich toolset offered by the ROS framework, it is possible to efficiently and promisingly circumvent the limitations of the native LEGO software, using a computer as an extension, which will perform additional processing and therefore, allow the creation of more complex prototypes.*

Resumo. *Este artigo descreve de forma sucinta o framework ROS e suas funcionalidades, apresentando pontos positivos e demonstrando sua usabilidade em uma aplicação prática cujo objetivo é a leitura do sensor de toque do KIT LEGO® Mindstorms® Education NXT 9797. Por meio de um rico conjunto de ferramentas oferecido pelo framework ROS, é possível contornar de forma eficiente e promissora as limitações do software nativo do KIT LEGO®, utilizando como extensão de processamento um computador, o qual deve realizar processamento adicional e consequentemente, permitir a criação de protótipos mais complexos.*

1. Introdução

Robot Operating System (ROS) é uma plataforma de software de código aberto projetado para suportar uma nova geração de robôs pessoais. A comunidade ROS desenvolveu o *framework* para uma ampla gama de plataformas, disponibilizando-o para diferentes sistemas operacionais (SOs), como Windows, Linux e Mac OS. O *framework* ROS é um rico conjunto de ferramentas que gerencia de forma eficiente a mediação entre Sistema Operacional (SO) e demais aplicações, fornecendo bibliotecas reutilizáveis e ferramentas que são projetadas para trabalhar de forma independente, com o intuito de ajudar desenvolvedores a criarem aplicações no campo da robótica. Pode-se afirmar que o ROS representa uma coleção muito útil de softwares voltados exatamente para ajudar na percepção, controle e modelagem dos dispositivos de hardware que serão lidos, fornecendo serviços que são esperados de um SO, incluindo a abstração do hardware em baixo nível, o que permite a leitura e o controle do mesmo, passagem de mensagens entre processos e gerenciamento de pacotes [Quigley et al. 2009].

O ROS possui uma ampla documentação, em sua maioria em língua inglesa[1]. Na página da comunidade ROS, existe um extenso material de apoio, oferecendo suporte a

1 Disponível em http://www.ros.org/wiki/ Acesso em: 17/01/2013.

compartilhamento de códigos fonte e experiências realizadas com o *framework* ROS, além de artigos científicos que envolvem sua utilização em diversas áreas do conhecimento. A comunidade ROS possibilita ao usuário fazer parte da mesma, mediante um cadastro pessoal no site. Dessa forma, o usuário pode deixar eventuais perguntas e/ou dúvidas que poderão ajudá-lo a solucionar algum problema detectado[2].

Este software é uma ferramenta de código aberto (*open source*), com todo seu código do núcleo licenciado pela licença BSD (*Berkeley Software Distribution License*). Paralelamente o sistema também é projetado para ser capaz de incluir componentes escritos sob várias licenças públicas da GNU (GNU *general public license* ou GPL[3]) [Cousins 2010b]. Possui uma estrutura distribuída de processos, a qual apoia a reutilização de código em robótica. Suas bibliotecas possuem funcionalidade para diversas linguagens de programação que serão mencionadas na seção 2. Este *framework*, originado da *Stanford University* e *Research Institute*, é considerado apropriado para grandes processos de desenvolvimento e é atualmente apoiado pela *Willow Garage*.

No contexto desse artigo, o software ROS será usado com o kit LEGO® Mindstorms® Education NXT 9797, com o intuito de compor um estudo de caso visando a leitura de sensores. Para oferecer melhores esclarecimentos para a compreensão do estudo de caso proposto, a seção 2 irá apresentar uma breve descrição do *framework* ROS e de suas funcionalidades, a seção 3 irá apresentar o estudo de caso pretendido e a seção 4 as conclusões e propostas de trabalhos futuros.

2. Robot Operating System (ROS)

O *framework* ROS foi projetado para atender a um conjunto específico de desafios encontrados durante o desenvolvimento em grande escala de robôs de serviço. O ROS é muito mais do que apenas um serviço oferecido a robôs móveis e de manipulação [Alexander et al. 2012].

Uma aplicação ROS é um grafo de nodos, os quais representam um processo no *framework* ROS. Os nodos podem residir em uma mesma máquina ou em máquinas diferentes. A comunicação entre nodos é feita usando dois protocolos, os tópicos (*topics)*, protocolo assíncrono que implementa um fluxo de dados e troca de mensagens, possibilitando que múltiplos *publishers/subscribes* concorram a um único tópico, onde a ordem das mensagens é irrelevante e os serviços (*services*), protocolo síncrono que é definido como uma chamada de procedimento remoto, a qual espera por uma mensagem de resposta, sendo possível ter vários clientes simultâneos. Ambos utilizam como diretório mestre, o ROS *Master*[4].

Sua organização como *framework* pode ser resumida em um serviço ponto-a-ponto (*peer-to-peer*), baseado em ferramentas (*tools-based)*, aplicável a múltiplas linguagens (*multi-lingual*), leve (*thin*), livre e de código aberto. Contudo, segundo Quigley et al. (2009), nenhum outro *framework* existente possui esse conjunto de critérios de projeto reunidos em uma só ferramenta. Esses critérios podem ser explicados de forma objetiva, como segue [Quigley et al. 2009]:

1. Ponto-a-ponto: um sistema construído usando ROS consiste em uma série de

2 Disponível em http://answers.ros.org/questions/ Acesso em: 17/01/2013.
3 Disponível em http://www.nacaolivre.com.br/open-source/licenca-gpl/ Acesso em: 20/01/2013.
4 Disponível em http://barraq.github.com/fOSSa2012/slides.html#slide-13 Acesso em: 19/01/2013.

processos, potencialmente distribuídos em um número diferente de hospedeiros, ligados em tempo de execução do sistema em uma topologia ponto-a-ponto.

2. Multi linguagem: ao escrever algum código, muitas pessoas têm preferências por uma determinada linguagem de programação. Estas podem opções técnicas ou pessoais. Por essa razão, o ROS foi projetado com um idioma neutro, suportando atualmente C, C ++, Python, LISP. Para fornecer o suporte a diversas linguagens de programação, o ROS utiliza uma IDL (*Interface Definition Language*) , que é uma linguagem neutra de definição de interface para descrever as mensagens enviadas entre os módulos.
3. Baseado em ferramenta: com o intuito de gerenciar a complexidade do ROS, existe um projeto de *microkernel*, no qual um grande número de pequenas ferramentas são usadas para criar e executar vários componentes, ao invés de construir um sistema monolítico.
4. Leve: Tendo em vista a reutilização de código, os desenvolvedores do ROS foram encorajados a se fundamentarem em bibliotecas independentes. O seu sistema de compilação executa construções modulares dentro da árvore de códigos-fonte, usando o CMake[5], o que torna trivial seguir essa ideologia “leve”.
5. *Free and Open-Source:* como mencionado na seção 1, o código fonte completo do ROS está disponível publicamente, permitindo o desenvolvimento de projetos comerciais e não comercial.

3. Estudo de Caso: Leitura de Sensor usando o ROS

Para o desenvolvimento do presente estudo de caso, foi usado o kit LEGO® Mindstorms® Education NXT 9797. É importante registrar que a última versão do ROS (*fuerte*) não oferece suporte para o pacote LEGO® Mindstorms® NXT. Assim, as versões que podem ser usadas com esse kit de desenvolvimento são versões anteriores, como o ROS Electric ou o ROS Diamondback. A versão usada foi o ROS Electric, por ser uma versão estável e adequada a este estudo de caso. Com as ferramentas disponíveis, abre-se um amplo leque de possibilidades para o controle do kit de desenvolvimento. O foco do presente artigo é a demonstração da leitura de sensores do kit LEGO®, em especial o sensor de toque.

3.1. O kit LEGO® Mindstorms® NXT

O kit LEGO® Mindstorms® NXT 2.0 é um kit robótico modular produzido pela LEGO, é constituído por um conjunto de peças de encaixe, acrescido de sensores e motores que são controlados por um processador programável ou bloco inteligente, o qual permite a leitura de sensores e acionamento de motores [Quirino e Gonçalves 2010]. Esse kit é interessante no contexto experimental por ser voltado à educação tecnológica[6].

O bloco inteligente permite a conectividade via módulo *Bluetooth* ou porta USB 2.0. Disponibiliza três portas de saída e quatro portas de entrada digitais, possui um *display* que mostra as funcionalidades do bloco e um alto-falante. Além do bloco inteligente, o kit LEGO® Mindstorms® Education NXT 9797 possui bateria recarregável de lítio; três servo-motores; dois sensores de toque, um sensor ultra-sônico, um sensor de

5 Cmake é um sistema de automação de compilação de sistemas. Disponível em http://leandro.setefaces.org/2011/tutorial-cmake-p-i/ Acesso em: 20/01/2013.

6 Disponível em http://mindstorms.lego.com/en-us/whatisnxt/default.aspx Acesso em: 17/01/2013.

cor e de intensidade luminosa [Silva e Porto 2011]. Seu hardware apresenta alta durabilidade e facilidade de alteração considerando as peças de encaixes, sendo caracterizado por um custo relativamente baixo em comparação a outros kits (ex. Aldebaran NAO[7], bioloid[8] etc).

A interação com o kit LEGO® Mindstorms® NXT é realizada por meio de uma conexão entre o SO Ubuntu 11.10 e o bloco inteligente, via *Bluetooth* ou cabo USB. Em geral, a interface usual para o ROS é um terminal no qual são executados programas em linhas de comando.

Muito embora o ROS não seja considerado "amigável ao usuário", as limitações do ambiente de desenvolvimento nativo fornecido com o kit LEGO® motivam o seu uso. Dentre as limitações observadas, pode-se citar uma linguagem de programação limitada, a qual não permite, por exemplo, a criação de um servidor baseado em *sockets* TCP no bloco inteligente e não dá suporte a tele-operação de robôs. Também existem limitações considerando o hardware nativo como a baixa capacidade de processamento (Atmel 32 bits ARM7) e pouca quantidade de memória (256 Kbytes FLASH, 64 Kbytes de RAM). Dessa forma, justifica-se o uso do *framework* ROS como forma de superar as limitações observadas, sendo possível criar protótipos mais complexos, mediante o uso de um computador como extensão para realizar processamento adicional.

3.2. Preparação do Espaço de Trabalho

Para instalação do ROS Electric, o site oficial do ROS oferece o procedimento completo para o SO Linux Ubuntu[9]. A versão usada para a confecção deste artigo foi a 11.10 em plataforma de 32 bits. O suporte estável para o ROS é fornecido para Linux Ubuntu, embora existam outros SOs com suporte experimental. Há quatro configurações padrão para a instalação do ROS. A mais apropriada para o estudo de caso pretendido, é a instalação da versão `Desktop-Full Install`, que é a recomendada pelo site do ROS. Para essa instalação é necessário estar conectado a Internet.

Após a instalação bem sucedida do ROS Electric é necessário modificar as configurações do interpretador de comandos usado, nesse caso, o `bash`, de modo que este adicione o ambiente de desenvolvimento do ROS a cada sessão iniciada. Além dos pacotes básicos do ROS Electric, recomenda-se a instalação de algumas ferramentas independentes, as quais poderão ser usadas posteriormente no sistema ROS. São elas o `rosinstall`, usada para adquirir facilmente pacotes e o `rosdep` que permite instalar de forma prática dependências à instalação original do ROS. Em seguida, é necessário criar uma *workspace* pessoal, onde os pacotes desenvolvidos serão criados. Esse procedimento é realizado mediante a execução do comando `rosws init ~/ros_workspace /opt/ros/electric`.

O espaço de trabalho padrão do ROS é configurado em `/opt/ros/electric/setup.bash`. Dessa forma, é preciso mudar esse diretório para a pasta criada anteriormente. Para tanto, basta editar o arquivo `.bashrc` e acrescentar o caminho real da nova pasta de trabalho (ex. `source`

7 Disponível em http://www.aldebaran-robotics.com/en/ Acesso em: 28/01/2013

8 Disponível em http://www.trossenrobotics.com/bioloid-and-humanoid-robot-kits.aspx Acesso em: 28/01/2013.

9 Disponível em: http://www.ros.org/wiki/electric/Installation/Ubuntu Acesso em: 17/01/2013.

/home/user/ros_workspace/setup.bash). Além disso, é necessário atualizar a referência do arquivo setup.bash (ex. source ~/ros_workspace setup.bash). Em seguida é criada uma pasta específica para uso na criação de pacotes ROS. Essa pasta é colocada dentro do diretório ros_workspace (ex. mkdir ~/ros_workspace/sandbox). O caminho do pacote deve ser inserido no *path* do ROS (ROS_PACKAGE_PATH), com o comando source /home/user/ros_workspace/setup.sh. Dessa forma, tem-se o sistema ROS instalado e a pasta pessoal de trabalho criada.

3.3. Adição dos Pacotes Necessários para a operação do kit LEGO®

Os procedimentos de instalação dos pacotes necessários para a operação do kit LEGO® Mindstorms® NXT estão disponíveis no site do ROS[10]. O software ROS para LEGO é organizado em pacotes e pilhas, os pacotes contém funcionalidades referentes ao processos do ROS, as pilhas coletam conjuntos de pacotes e juntos oferecem funcionalidade para a comunicação entre nodos, contendo a infraestrutura básica do ROS [Cousins 2010a]. Antes de instalar os pacotes NXT, é necessário fazer algumas configurações.

- A criação de um grupo lego e adição do usuário usado ao grupo;
- Criar um arquivo de regras udev para o grupo lego que acabou de ser criado. Isso pode ser feito de forma simples mediante o comando apresentado na Listagem 1, o qual define regras para o arquivo lego.rules. Este arquivo será movido da pasta /tmp e será identificado em /etc/udev/rules.d, para permitir o acesso via módulo USB ao bloco inteligente. Posteriormente deve-se reiniciar o udev.
- Inserir novas fontes para o gerenciador de pacotes, de modo que este possa instalar pacotes do ROS (Listagem 2).
- Instalação do pacote ros-electric-nxtall usando o gerenciador de pacotes do Ubuntu.

```
echo "BUS==\"usb\", ATTRS{idVendor}==\"0694\", GROUP=\"lego\", MODE=\"" > /tmp/70-lego.rules && sudo mv /tmp/70-lego.rules /etc/udev/rules.d/70-lego.rules
```

Listagem 1: Regra criada para o gerenciador de dispositivos udev.

```
Sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu oneiric main" > /etc/apt/sources.list.d/ros-latest.list
```

Listagem 2: Inserção de nova fonte de pacotes no gerenciador de pacotes do Ubuntu.

Assim, a infraestrutura necessária para a operação do kit LEGO® por intermédio do *framework* ROS está pronta. A seção 3.4 apresentará a leira de sensores via ROS.

3.4. Leitura de Sensores do LEGO® Mindstorms® NXT

Para leitura de sensores usando o ROS, é necessário certificar-se que a comunicação com

10 Disponível em: http://www.ros.org/wiki/Robots/NXT/electric Acesso em: 17/01/2013.

o bloco inteligente está sendo executada de forma correta. Isso pode ser realizado por meio do comando `roscore` (sem parâmetros), como segue na Figura 1.

```
iohan@iohan-HP-Compaq-8200-Elite-SFF-PC:~$ roscore
... logging to /home/iohan/.ros/log/44aa3a86-280d-11e2-b468-e839351653cf/roslaun
ch-iohan-HP-Compaq-8200-Elite-SFF-PC-2530.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://iohan-HP-Compaq-8200-Elite-SFF-PC:51711/
ros_comm version 1.6.6

SUMMARY
========

PARAMETERS
 * /rosversion
 * /rosdistro

NODES

auto-starting new master
process[master]: started with pid [2545]
ROS_MASTER_URI=http://iohan-HP-Compaq-8200-Elite-SFF-PC:11311/

setting /run_id to 44aa3a86-280d-11e2-b468-e839351653cf
process[rosout-1]: started with pid [2558]
started core service [/rosout]
```

Figura 1: Resultado do comando roscore.

A saída do comando apresentada na Figura 1 demonstra como deverá estar o sistema, caso o ROS esteja em execução e a conexão tenha sido realizada com sucesso. A conexão com o bloco inteligente fica evidente considerando a última linha mostrada na Figura 1, (“`started core servisse[/rosoutr]`”). Isso indica que o serviço foi iniciado com sucesso e que a conexão foi estabelecida com o bloco inteligente.

O bloco inteligente possui quatro portas para fixação dos sensores (porta 1, 2, 3, e 4, em sua parte inferior (Figura 2). Na parte superior existem três portas para fixação dos motores e uma porta para conexão via módulo USB. Esta última permite estabelecer conexão com o bloco inteligente, o qual será controlado pelo computador via porta USB. O sensor de toque pode ser conectado à porta 1, como mostrado na Figura 2.

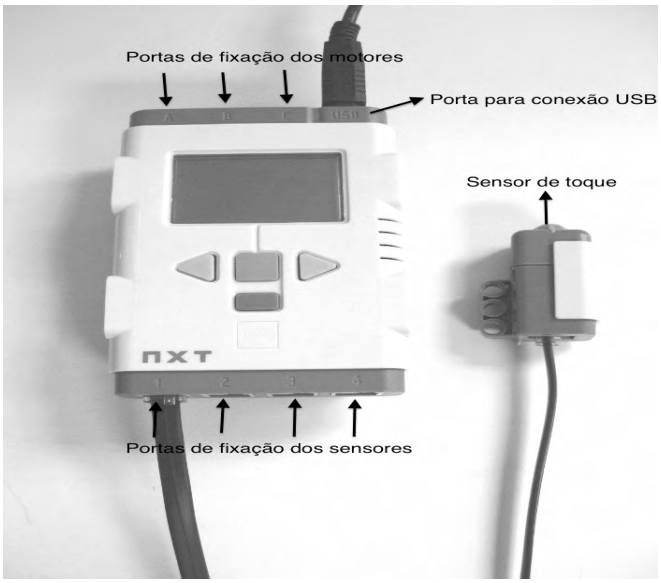

Figura 2: Bloco inteligente e sensor de toque.

A descrição do sensor a ser usado é fornecida ao ROS mediante a especificação de um arquivo do tipo `.yaml`, o qual fornece a descrição do sensor, seu tipo e um identificador. O arquivo robot.yaml é especificado para um sensor de toque (Listagem 3).

```
nxt_robot:              // o nome do robô
       - type: touch    // tipo do sensor
         frame_id: touch_frame   // identificador para o sensor
         name: my_touch_sensor   // nome do sensor
         port: PORT_1            // porta do bloco inteligente
         desired_frequency: 20.0 // frequência definida
```

Listagem 3: Arquivo robot.yaml contendo a especificação do sensor de toque.

O próximo passo é a criação de um arquivo de carga com extensão `.launch` para o ROS (ROS *launch*), o qual irá iniciar o pacote NXT-ROS (Listagem 4).

```
<launch>
     <node pkg="nxt_ros" type="nxt_ros.py" name="nxt_ros"
      output="screen" respawn="true">
           <rosparam command="load" file="$(find learning_nxt)/
            robot.yaml" />  </node>
</launch>
```

Listagem 4: Arquivo `robot.launch`.

Agora se faz necessário testar as configurações feitas anteriormente. Assim verifica se o bloco inteligente está se comunicando com o sensor pretendido. Para isto, executa-se o comando `roslaunch robot.launch`. O resultado é a inicialização do serviço NXT-ROS e a criação de uma instância do sensor de toque denominada "`my_touch_sensor`" na porta 1.

```
SUMMARY ========
      PARAMETERS
       * /nxt_ros/nxt_robot NODES /
          nxt_ros (nxt_ros/nxt_ros.py)
      starting new master (master configured for auto start)
      process[master]: started with pid [11699]
      ROS_MASTER_URI=http://bvo:11311/
      process[rosout-1]: started with pid [11712]
      started core service [/rosout]
      process[nxt_ros-2]: started with pid [11724]
      [INFO] 1282349960.808368: Creating touch with name
      my_touch_sensor on PORT_1
```

Listagem 5: Resultado do comando roslaunch com as configurações pré-definidas.

A visualização dos sensores pode ser realizada mediante o uso do comando `rostopic list`. Para este estudo de caso, se todos os passos descritos foram executados de forma correta o sensor `my_touch_sensor` aparecerá na lista de sensores em execução. Para verificar o acionamento do sensor de toque execute o seguinte comando rosrun nxt_python my_touch_sensor.py, no terminal aparecerá os resultados, quando o sensor de toque for pressionado será visualizado TOUCH: TRUE, caso contrário TOUCH: FALSE.

A visualização dos demais sensores do kit LEGO® Mindstorms® NXT segue os mesmos procedimentos considerando a criação de arquivos de configuração `.yaml` e `.launch`. Vale observar que ao se usar vários sensores paralelamente, pode ser criado apenas um arquivo de configuração com especificações de todos os sensores envolvidos.

4. Conclusão

Sob a ótica do desenvolvedor, este estudo de caso proporciona o conhecimento mais detalhado do *framework* ROS. É possível abstrair o quanto a ferramenta em questão proporciona em termos de funcionalidades para a operação e interação com robôs. A leitura do sensor de toque representa uma pequena demonstração do que é possível fazer utilizando o *framework* ROS e um kit robótico mais simples.

Embora o estudo de caso tenha utilizado um ambiente simples em termos de hardware, a diversidade de plataformas suportadas, o fato de ser livre e de código aberto, sua eficiência quanto a gerência de sistemas robóticos e sua flexibilidade tornam o ROS apto a lidar com desafios em grande escala. Exemplos como o robô PR2 [Cousins, 2010b] mostram o quanto este *framework* é revolucionário e demonstram seu grande potencial quanto ao desenvolvimento de aplicações voltadas a robôs de serviço.

Como trabalho futuro pretende-se utilizar o estudo de caso apresentado e aprofundá-lo considerando o controle dos motores e demais sensores presentes no kit LEGO® Mindstorms® NXT. Com isso, prevalecerá o intuito de criar um primeiro protótipo de robô completo, o qual usa vários sensores e motores em paralelo.

Referências

Alexander, B., Hsiao, K., Jenkins, C., Suay, B. e Toris, R. (2012). Robot Web Tools. In IEEE Robotics & Automation Magazine, pages 20-23.

Silva, S. R. X. e Porto, L. (2011) “Análise Comparativa de Kits de Robótica Educativa”, In: XXXIX Congresso Brasileiro de Educação em Engenharia.

Cousins, S. (2010a). ROS on the PR2. In IEEE Robotics & Automation Magazine, pages 23-25.

Cousins, S. (2010b). Welcome to ROS Topics. In IEEE Robotics & Automation Magazine, pages 13-14.

Quirino, G. e Gonçalves, V. (2010) “O Uso da Robótica Educacional no Ensino Fundamental”, In: VIII Encontro Anual de Computação.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R. e Ng, A. (2009) “ ROS: an open-source Robot Operating System”, In: International Conference on Robotics and Automation.