Estudo de Caso Usando a Plataforma Microsoft® Robotics Developer Studio (RDS)

Jéssica S. Guimarães, Dalton M. Tavares

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

{jessica.silva.gui}@gmail.com, {dalton.tavares}@catalao.ufg.br

Abstract. *This paper presents a case study showing the control of a robot via an application developed in C#, in order to demonstrate the use of Microsoft® Robotics Developer Studio (MRDS) for controlling a prototype developed from a LEGO® Mindstorms® NXT kit.*

Resumo. *Este artigo apresenta um estudo de caso apresentando o controle de um robô por meio de um aplicativo desenvolvido em C#, para demonstrar a utilização do Microsoft® Robotics Developer Studio (MRDS) para controlar um protótipo desenvolvido a partir de um kit LEGO® Mindstorms® NXT.*

1. Introdução

Uma forma bastante eficaz para estudos envolvendo programação é a aplicação da mesma em desenvolvimento robótico, colocando em prática os conhecimentos vistos em cursos de graduação e programando protótipos de robôs, por meio de softwares existentes. Estes protótipos podem ser construídos a partir de kits de desenvolvimento disponíveis em mercado, de fácil manuseio e custo acessível.

Com os avanços tecnológicos na área da robótica, surgiram diversos kits de desenvolvimento como o robô NAO da Aldebaran Robotics[1], o Bioloid[2] e o LEGO® Mindstorms®[3], entre outros. Para o presente estudo de caso, o kit desenvolvido pela LEGO® Mindstorms® foi escolhido, devido a sua flexibilidade e baixo custo frente às demais opções.

Em 2006, a LEGO® lançou uma nova linha de produtos, o LEGO® Mindstorms® NXT, incentivando a educação tecnológica de forma descontraída [Kim e Jeon 2008]. O kit de desenvolvimento é constituído por peças para montagem rápida, incluindo um bloco de controle programável, sensores de toque, para medição de intensidade luminosa, ultrassônico, motores, entre outros. Seu ambiente de programação nativo permite a aplicação de comportamentos que utilizam as ferramentas que o acompanham através da programação icônica[4].

O bloco de controle pode ser programado usando outros softwares, diferentes do ambiente de programação padrão fornecido pela LEGO®. Uma das opções é o Microsoft® Robotics Developer Studio (MRDS), aplicável ao controle e simulação de robôs, destinado

[1] Para maiores informações, vide: http://www.aldebaran-robotics.com/en/ Acesso em: 28/01/2013.
[2] Para maiores informações, vide: http://robosavvy.com/store/product_info.php/products_id/82 Acesso em: 28/01/2013.
[3] Para maiores informações, vide: http://mindstorms.lego.com/en-us/Default.aspx Acesso em: 28/01/2013.
[4] Uma forma simples de programar, utilizando metodologia gráfica, ao invés de código fonte inserido como texto, permitindo a construção de cenários e a interpretação visual.

a desenvolvedores acadêmicos, amadores e comerciais.

Este artigo tem por objetivo apresentar um estudo de caso envolvendo o controle de um robô móvel, desenvolvido usando o kit LEGO® Mindstorms® NXT e o software RDS. Para tanto, será fornecida uma breve descrição das principais funcionalidades do software RDS (seção 2), seguida da apresentação do ambiente de desenvolvimento e dos resultados alcançados (seção 3). Uma breve conclusão e propostas de trabalhos futuros serão apresentadas na seção 4.

2. Estrutura do RDS

A arquitetura básica do RDS é apresentada na Figura 1. De forma sucinta, o RDS fornece serviços de alto nível trabalhando normalmente com abstrações de serviços de baixo nível, as quais, por sua vez, tratam da comunicação com dispositivos. Os serviços de baixo nível são fornecidos pelos fabricantes de robôs ou desenvolvidos pelo próprio usuário, por meio de aplicativos escritos em C#, por exemplo. É através de sua camada de abstração de hardware, com interface bem definida, que os serviços de alto nível têm controle de entrada e saída (I/O). Os serviços CCR e DSS (explicados abaixo) são componentes característicos do RDS e atuam no contexto de comunicação entre nós. Esse modelo é baseado no *framework* .NET. Esse conjunto de serviços se comunica com o *firmware* do robô através de uma porta serial – USB - ou conexão via rede, através do Windows, o qual é fundamental para que o serviço de interação do robô com o aplicativo seja realizado [Microsoft Robotics Group 2012].

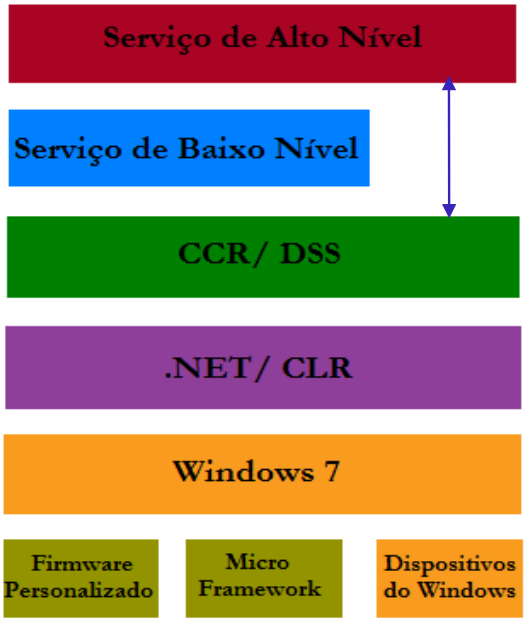

Figura 1. Arquitetura do software.

A Concorrência e Coordenação Runtime (CCR – *Concurrency and Coordination Runtime*) é um modelo que trabalha com programação multi-thread e sincronização de

tarefa, e permite interação com robôs em tempo real sem a complexidade de trabalhar com semáforos, exclusão mútua, tratamento de possíveis assincronias, entre outras, devido o uso de protocolo aberto. Já os Serviços de Software Distribuídos (DSS - *Distributed Software Services*) é um programa que atuam juntamente com CCR, utilizando vários serviços ou instancias de serviços, tais como: componentes de hardware (sensores e atuadores), componentes de software (repositório, interface de usuário), fusão entre sensores e tarefas relacionadas aos mesmos. [Cepeda, Chaimowicz e Soto 2010].

3. Estudo de caso: utilizando sensor de toque para encontrar obstáculos

Utilizando um kit LEGO® Mindstorms® NXT, foi estruturado um protótipo que tem como principais características o uso de dois motores, para acionar as rodas, o uso de um sensor de toque, para reconhecer os obstáculos que lhe forem impostos através de contato direto e um bloco de controle NXT. Utilizando o RDS foi desenvolvido um aplicativo que controla o movimento dos motores do protótipo fazendo com que ele se movimente no espaço e, ao identificar um toque por meio do sensor, emite uma mensagem informando que encontrou um obstáculo. Este aplicativo foi desenvolvido na linguagem C#, usando o Microsoft® Visual C# Express, um dos softwares utilizados na plataforma RDS. A Figura 2 mostra um exemplo de robô utilizando um sensor de toque e motores para se movimentar.

Para utilizar a plataforma RDS, primeiramente instalamos o sistema operacional Windows 7, o Microsoft® Visual C# Express, o Kinect para Windows SDK, o qual pode ser utilizado futuramente em projetos envolvendo o reconhecimento de movimentos por meio de um sensor Microsoft® Kinect[5] e o RDS em sua última versão[6].

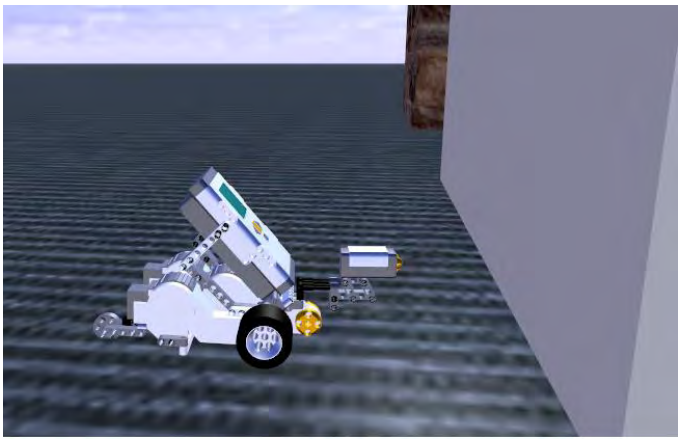

Figura 2. Protótipo indo em direção a uma barreira.

Após a preparação do ambiente de desenvolvimento, a comunicação entre o bloco de controle e o computador foi configurada, com o intuito de permitir o controle do protótipo utilizando o programa desenvolvido na plataforma RDS. De forma sucinta, foi criada uma conexão do notebook com o bloco de controle via *Bluetooth*, adicionando o bloco de controle como dispositivo do notebook, de forma que ambos se reconhecessem e

[5] Para maiores informações, vide: http://www.microsoft.com/en-us/kinectforwindows/

[6] Até a data de submissão do artigo, o software RDS estava em sua versão 4.0. Fonte: http://www.microsoft.com/robotics/. Acesso em: 11/01/2013.

pudessem trocar informações.

Para obter um melhor desempenho é indicado que se faça uma atualização da versão do *firmware* do bloco de controle para uma versão mais recente[7]. Esta atualização é feita usando o software que é fornecido juntamente com o kit LEGO® Mindstorms® NXT, instalado em um computador[8].

3.1. Controle usando C#

Usando o Visual C# Express, foi desenvolvido um aplicativo, o qual imprime uma mensagem ao colidir com um obstáculo durante seu percurso. O primeiro passo é criar um novo serviço utilizando o interpretador de comandos DSS, o qual fornece suporte ao protocolo descentralizado de serviços de software (Decentralized Software Services Protocol – DSSP). No Visual Studio é necessário adicionar uma referência à biblioteca `RoboticsCommon.Proxy.dll`, a qual fornece suporte ao sensor de toque. Observe que ao iniciar o aplicativo (neste caso chamado de `MyTutorial1`) é gerado um manifesto que aparece no lado direito da tela (Figura 3 e 4), e diz respeito as configurações de depuração fornecidas pelo `DSSNewService`.

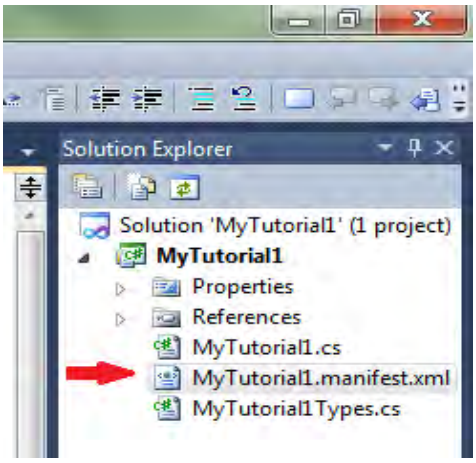

Figura 3. Manifesto criado na inicialização do aplicativo.

[7] A versão usada foi 1.29, entretanto existe uma versão mais recente – a 1.31. Fonte: http://mindstorms.lego.com/en-us/support/files/default.aspx#Firmware Acesso em: 11/01/2013.

[8] Usando a opção *update*, conecta-se o bloco de controle à porta USB do computador e se faz o download do novo *firmware*. Fonte: http://msdn.microsoft.com/en-us/library/bb483050.aspx Acesso em: 28/01/2013.

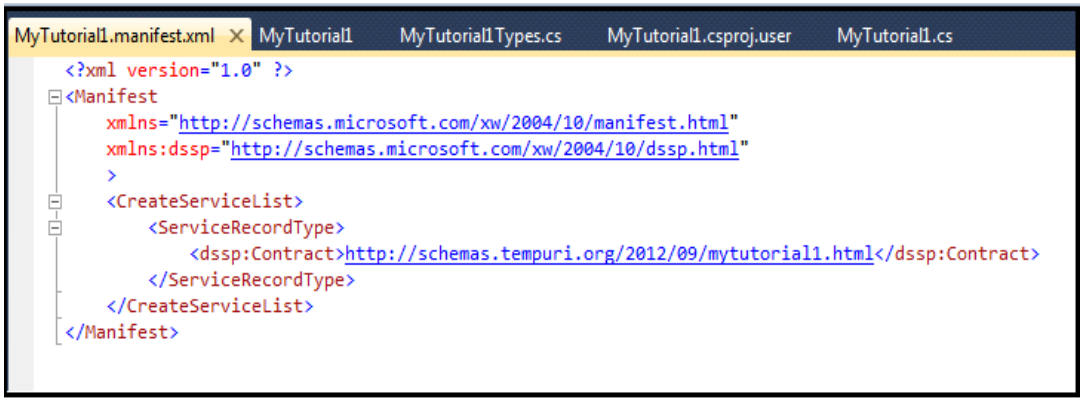

Figura 4. Descrição do manifesto.

É necessário modificar o manifesto de acordo com o tipo de hardware usado no âmbito da guia `Debug`. Neste caso, a configuração ficou conforme a descrição fornecida na Listagem 1.

```
/p:50000  /t:50001
/m:"samples/MyTutorial1/MyTutorial1.manifest.xml"
/m:"samples/config/LEGO.NXT.MotorTouchSensor.manifest.xml"
```

Listagem 1. Referência ao manifesto adequado ao nosso hardware.

Feito isso, um serviço de comunicação é criado e inicializado entre o aplicativo e o sensor de toque. Ao final do código insere-se a assinatura do código que permitirá a execução do serviço de notificação, o qual recebe a notificação de que o sensor foi pressionado e então chama o método `BumperHandler()`. Este método envia a mensagem para o console, de forma que a informação possa ser visualizada. Na Listagem 2 são apresentadas as linhas que foram inseridas no código.

```
        //Criando o serviço de comunicação entre este aplicativo e o sensor de
toque
        [Partner("bumper", Contract = bumper.Contract.Identifier,
           CreationPolicy = PartnerCreationPolicy.UseExisting)]
        private bumper.ContactSensorArrayOperations _bumperPort = new
bumper.ContactSensorArrayOperations();


            // Iniciando serviço que ouve o bumper.
            SubscribeToBumpers();
              }

        /// <summary>
        /// Subscrevendo serviço de bumper.
        /// </summary>
        void SubscribeToBumpers()
        {
```

```
            // Criando porta de notificação do bumper.
            bumper.ContactSensorArrayOperations bumperNotificationPort = new
bumper.ContactSensorArrayOperations();

            // Subscreve o seriço bumper e recebe notificações no
bumperNotificationPort.
            _bumperPort.Subscribe(bumperNotificationPort);

            // Recebe atualização das notificações do bumperNotificationPort .
            Activate(
                Arbiter.Receive<bumper.Update>
                    (true, bumperNotificationPort, BumperHandler));
        }


        /// <summary>
        /// Recebe a notificação de que o sensor foi pressionado e envia a
informação para o console
        /// </summary>
        /// <param name="notification">Atualiza notificação</param>
        private void BumperHandler(bumper.Update notification)
        {
            if (notification.Body.Pressed)
            LogInfo(LogGroups.Console, "Ops, um obstáculo foi encontrado!");
        }
```

Listagem 2. Código fonte do aplicativo desenvolvido.

Ao executar o programa, é aberta uma janela no navegador com o endereço padrão http://127.0.0.1:50000/, que mostra as configurações do bloco de controle, do sensor e informações referentes à comunicação entre o aplicativo e o protótipo (Figura 5).

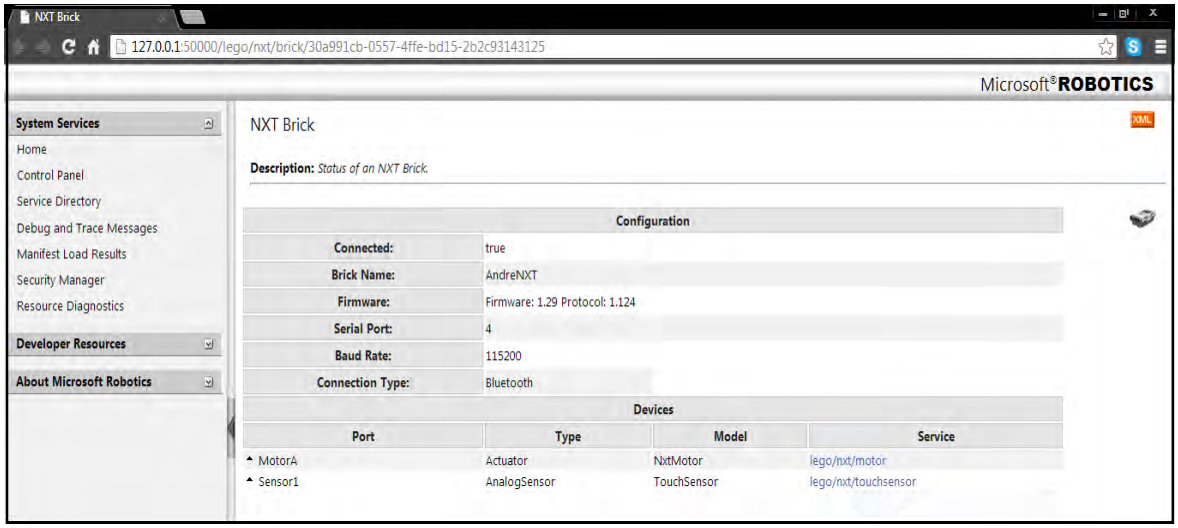

Figura 5. Página inicial do console de saída do serviço.

Clicando em `Debug` e mensagens de rastreamento, à esquerda da página podemos visualizar detalhes do serviço, como detalhes das mensagens, detalhes da comunicação com o hardware, erros, a mensagem de que o sensor foi pressionado, entre outros (Figuras 6 e 7).

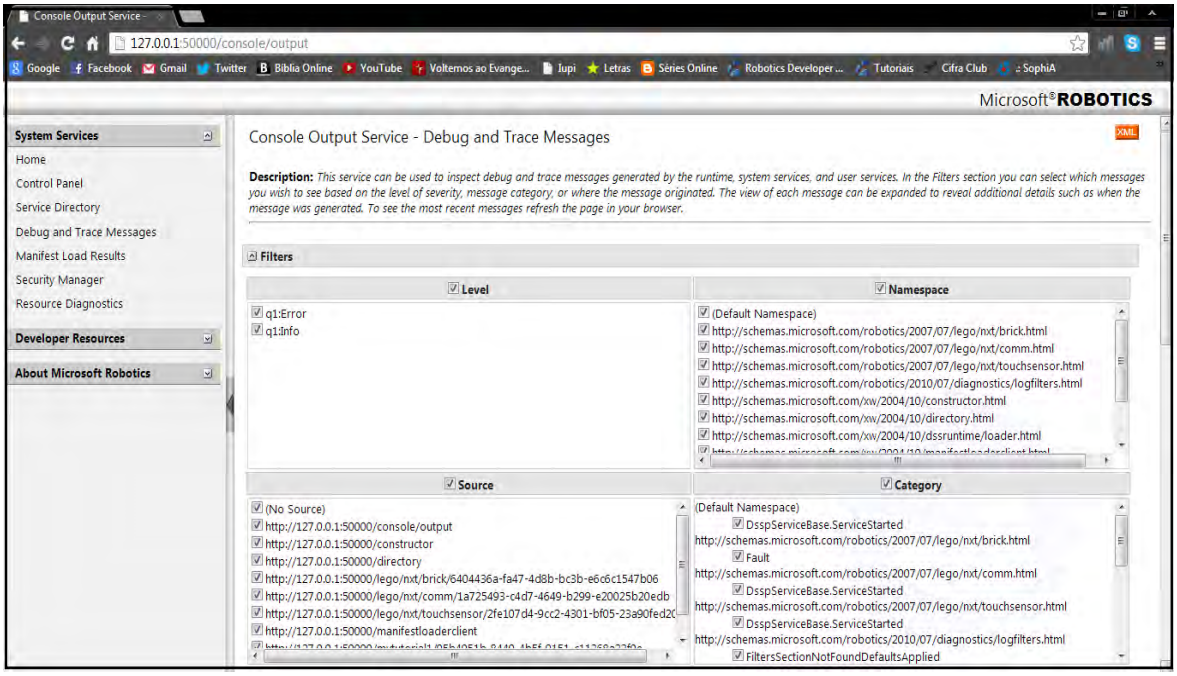

Figura 6. Console de saída do serviço – Debug e rastreamento de mensagens.

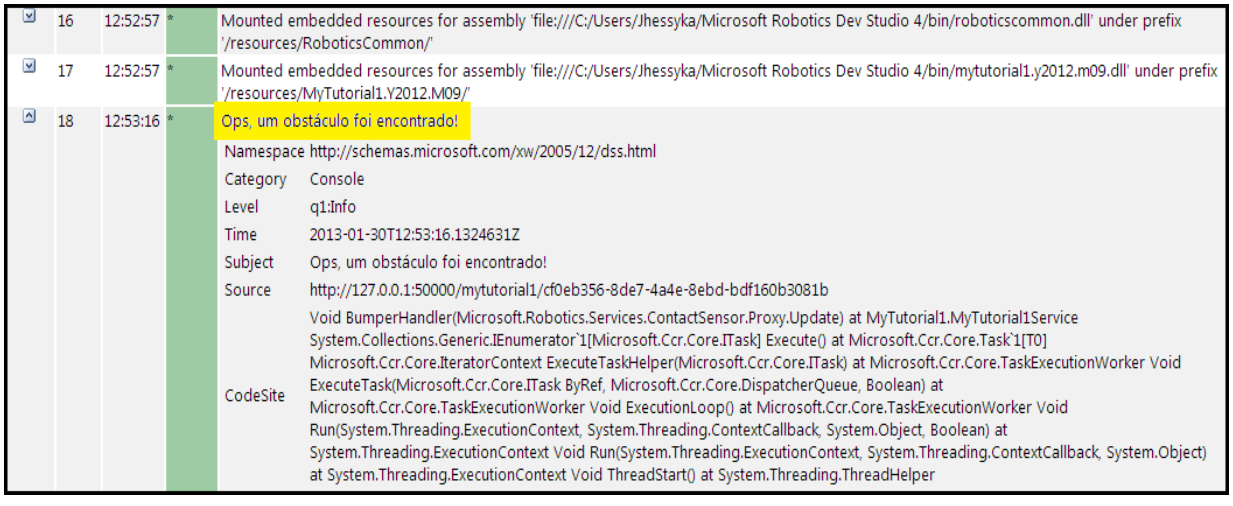

Figura 7. Mensagem apresentada quando o sensor é pressionado.

4. Conclusão

Depois de preparar o computador, configurar o bloco de controle NXT e confeccionar o aplicativo, observa-se a interação do *framework* RDS com o bloco de controle. A mensagem referente à detecção de barreira foi impressa em uma página do navegador (Figura 7), a qual também contém informações detalhadas sobre o protótipo (Figura 5), detalhes da comunicação com o hardware, entre outros (Figura 6).

Com este experimento, utilizando o Visual Basic é possível controlar um protótipo desenvolvido a partir do kit LEGO® Mindstorms® NXT e aplicativos escritos em C#. Também é possível observar a existência de uma boa interação entre o *software* RDS e o bloco de controle, o que permite a realização de uma ampla gama de experimentos.

Um ponto negativo observado durante a pesquisa é que não há simultaneidade com relação à detecção de toque e a impressão da mensagem na tela, sendo necessário atualizar a página aberta pelo programa para visualizar as informações atuais.

Como trabalho futuro, podemos realizar o controle de protótipos usando a interatividade com o sensor Microsoft® Kinect, capaz de captar movimentos através de rastreamento do corpo humano, o que pode permitir futuras implementações em escala maior, como por exemplo, no controle de braços mecânicos, os quais são amplamente utilizados pelas indústrias, como a automobilística, entre outras [Suay e Chernova 2011].

Projetos como o SMErobot[9], desenvolvido pela União Europeia, propuseram a aplicação de tecnologias inovadoras aplicáveis ao controle de robôs de modo a tornar a interação com seus programadores e operadores mais intuitiva. Uma aplicação sugerida e bastante promissora neste ramo é o reconhecimento de gestos e voz, o qual permite uma fácil interatividade do usuário com o robô [Neto e Pires 2009]. Ao invés de descrição de serviços utilizando linguagens de programação, os operadores mostrariam o serviço que deve ser feito por meio de gestos. Esses dados seriam captados por sensores, armazenados e enviados ao robô através de *Bluetooth*.

Referências

Kim, S. H. e Jeon, J. W. (2008) "Using visual programming kit and Lego Mindstorms: An introduction to embedded system", In: Proceedings of the IEEE International Conference on Industrial Technology (ICIT).

Neto, P. e Pires, J.N. (2009) "High-Level Programming for Industrial Robotics: using Gestures, Speech and Force Control". Submitted to the IEEE International Conference on Robotics and Automation (ICRA2009), Kobe, Japan, 2009.

Suay, H.B. e Chernova, S. (2011) "Humanoid robot control using depth camera", In: 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI).

Cepeda J.S., Chaimowicz L. e Soto R. (2010) "Exploring Microsoft Robotics Studio as a Mechanism for Service-Oriented Robotics", In: 2010 Latin American Robotics Symposium and Intelligent Robotics Meeting.

Microsoft Robotics Group (2012) "Robotics Developer Studio: Reference Plataform Design v1.0", In: Documentation of the Microsoft Robotics Developer Studio 4.

[9] Maiores informações em: http://www.smerobot.org/ Acesso em: 28/01/2013.