Desenvolvimento de módulos adicionais para a plataforma de robótica móvel educacional - RoboDeck

Antonio Valerio Netto[1], Osvaldo Hugo Bertone[2]

[1]Bolsista DT CNPq
SHIS QI 1 Conjunto B - Blocos A, B, C e D - Lago Sul – Brasília, DF, Brasil

[2] Coordenador de Engenharia Elétrica - Faculdade Anhanguera campus Sumaré
Av. Eugênia Biancalana Duarte, 501 - Jardim Primavera, Sumaré, SP, Brasil

antonio.valerio@pq.cnpq.br, hugobertone@yahoo.com

Abstract. *This paper describes the development of computational modules related to the choice of wireless technology to support remote data control and transmission of the robotic mobile platform called RoboDeck and the development of a High Performance Robotic Module (MRAP). In addition, it also describes the development of a Software Development Kit (SDK) to enable users to develop their applications for the robotic platform. In order to facilitate this construction, an API (Application Programming Interface) written in C language was created that will enable the use of all MRAP interaction commands. This API is part of the RoboDeck SDK, whose users are the platform's robotic application programmers. The applications developed in this project serve as examples, for RoboDeck programmers, of how to use the robot's computational vision capabilities, as well as the API available.*

Resumo. *Este artigo descreve o desenvolvimento de módulos computacionais relacionados à escolha da tecnologia de comunicação sem fio para suportar o controle e a transmissão de dados a distância da plataforma robótica móvel chamada RoboDeck e o desenvolvimento de um Módulo Robótico de Alta Performance (MRAP). Além disso, também descreve o desenvolvimento de um Software Development Kit (SDK) para permitir que os usuários desenvolvam suas aplicações para a plataforma robótica. Para facilitar esta construção, foi criada uma API (Application Programming Interface) escrita em linguagem C que possibilitará a utilização de todos os comandos de interação com o MRAP. Essa API faz parte do SDK do RoboDeck, cujos usuários são os programadores de aplicativos robóticos da plataforma. Os aplicativos desenvolvidos nesse projeto servem de exemplos, para programadores do RoboDeck, de como utilizar os recursos de visão computacional do robô, bem como a API disponibilizada.*

1. Introdução

A Robótica móvel possui aplicações em muitos segmentos da sociedade, passando pela automação de processos industriais, sistemas de monitoramento, exploração, entretenimento, educação, entre outros. Especificamente, na área de tecnologia educacional, a robótica tem um papel importante no emprego de *Edutainment*, como uma metodologia de engajamento no aprendizado de alunos, principalmente de ensino técnico. Esta metodologia utiliza elementos divertidos, como games, filmes, seriados de TV, aparelhos móveis, além da aplicação dos robôs, desenhados para se tornarem educativos. O *edutainment* além de atrair e prender a atenção dos estudantes, também busca reduzir a evasão escolar. Isso acontece porque a metodologia aumenta a motivação e engajamento dos alunos, contribuindo para uma aprendizagem mais eficaz. A metodologia surgiu ao aproveitar as três principais ações que as pessoas desenvolvem quando se entretém (envolvimento, interação e imersão) a favor do aprendizado.

A empresa XBot (*www.xbot.com.br*) nos últimos anos, tem se dedicado ao desenvolvimento e produção de robôs móveis que são utilizados por estudantes do ensino médio técnico até alunos de graduação e pós-graduação. Uma das tecnologias robóticas desenvolvidas é chamada de RoboDeck. Trata-se de uma plataforma robótica que pode ser usada em sala de aula para auxiliar o professor no ensino de conteúdos relacionados à linguagem de programação, eletrônica digital, robótica, entre outras de modo prático. Essa plataforma possui uma "arquitetura modular aberta", possibilitando que os alunos possam desenvolver, eles mesmos, uma nova aplicação ou módulo de *hardware* ou *software* (Menezes *et al.*, 2017) (Zanola *et al.*, 2017). No caso dos pesquisadores e hobbistas, esta plataforma pode ser utilizada para o desenvolvimento de aplicações mais sofisticadas, como a teleoperação. Exemplos de novos módulos e funcionalidades podem ser encontrados em Motta (2016), Wei (2015) e Pissardini (2014).

Em linhas gerais, a proposta de uma arquitetura com *hardware* aberto e *software* livre permite que alunos, pesquisadores ou mesmo empresas, possam criar novos acessórios (módulos) com diferentes aplicações em um tempo mais rápido focando especificamente na funcionalidade desejada. Para área educacional é possível trabalhar o aprendizado de conteúdos envolvendo disciplinas tradicionais, como matemática, além de disciplinas transversais, nas quais os alunos poderão aprender na prática a contextualizar problemas e por meio da programação, construir soluções (aprendizado baseado em problemas). Nesta linha é possível trabalhar conteúdos pedagógicos envolvendo, por exemplo, educação no trânsito, seleção de lixo reciclável, ecologia (reconhecimento de águas), inclusão social, entre outros. Para apoiar os alunos e professores do curso técnico de mecatrônica no processo de introdução da plataforma robótica no aprendizado prático, foram desenvolvidos módulos adicionais atrelados a plataforma RoboDeck. Além disso, foi criado também uma giga de testes para examinar se determinadas atividades foram executadas com sucesso utilizando o RoboDeck. Isto permitiu verificar possíveis problemas com os sensores, motores e a própria comunicação do robô.

O objetivo desse artigo é descrever o desenvolvimento de módulos computacionais relacionados à escolha da tecnologia de comunicação sem fio para suportar o controle e a transmissão de dados a distância para o RoboDeck, além do

Módulo Robótico de Alta Performance (MRAP) e de um *Software Development Kit* (SDK) para permitir que os usuários desenvolvam com mais facilidade suas aplicações. O artigo está divido em mais cinco seções. Na Seção 2 é apresentada a discussão da melhor tecnologia de comunicação sem fio para suportar tanto o controle quanto a transmissão de dados da plataforma robótica. Na Seção 3 é descrito o desenvolvimento do Módulo Robótico de Alta Performance (MRAP) do robô e na Seção 4 é descrito o desenvolvimento do SDK. Na Seção 5 são discutidos resultados do desenvolvimento e na Seção 6 são realizadas as considerações finais do projeto.

1.1 Visão geral da plataforma RoboDeck

O RoboDeck é uma plataforma robótica móvel baseada em rodas para terrenos *indoor* e *outdoor*. A composição mecânica do robô está assim constituída: quatro rodas independentes (omnidirecionais), cada uma controlada por um servo-motor de direção (com capacidade de direcionar ângulos de 45º para dentro e 45º para fora), as duas rodas dianteiras também são controladas por motores de tração; dois motores de tração responsáveis por colocar o robô em movimento, cada motor de tração possui um *encoder* capaz de medir a quantidade de giros do motor; um chassi de sustentação do robô com *payload* de 15kg.

O sistema de sensores do RoboDeck está conFigurado da seguinte forma: quatro sensores infravermelho (com capacidade de detecção de objetos na faixa de 4 a 30 cm), sendo três localizados na parte da frente e um na parte traseira do robô; quatro sensores ultrassom, localizados na parte da frente, traseira, esquerda e direita do robô (com capacidade de detecção de objetos na faixa de 3 a 600 cm, cobrindo um feixe cônico de aproximadamente 45º); um sensor de temperatura (em Cº); um sensor de umidade relativa do ar (%); um acelerômetro para medir a força de gravidade aplicada ao robô (decomposta nos eixos x,y,z); uma bússola para informar a orientação do robô; um dispositivo de GPS capaz de informar a posição geográfica do robô (latitude, longitude e altitude), a data e horário do meridiano de Greenwich, a velocidade de deslocamento, e o sentido de deslocamento do robô em relação ao norte magnético; e uma câmera para captura de imagens, com interface USB (esse é um recurso opcional, disponível junto à placa de alta performance). A Figura 1 ilustra a disposição dos sensores do RoboDeck.

O RoboDeck suporta até seis baterias (recarregáveis) de 18 V / Ah, embora o aconselhado seja o uso de quatro baterias (uma de cada lado). As baterias do RoboDeck lhe dão uma autonomia de até 4,5 horas. O sistema eletrônico do RoboDeck está conFigurado da seguinte forma: uma placa mãe com um processador ARM9, responsável pela execução dos comandos robóticos de baixo nível; uma placa ZigBee responsável pelo sistema de comunicação de controle do robô; uma placa de alta performance AMD GeodeTM para a execução do MAP (Módulo de Alta Performance), responsável por habilitar o sistema de captura de imagens do robô e aplicativos autônomos (outras placas de alta performance podem ser usadas, desde que suportem execução do sistema operacional Linux) ; uma fonte reguladora de tensão.

A arquitetura de *software* do RoboDeck está dividida em quatro módulos: Módulo de Controle de Seção (MCS), Módulo de Controle Robótico (MCR), Módulo de Controle de Comunicação (MCC) e Módulo de Alta Performance (MAP). A Figura 2 apresenta uma visão geral da arquitetura de *software* do RoboDeck.

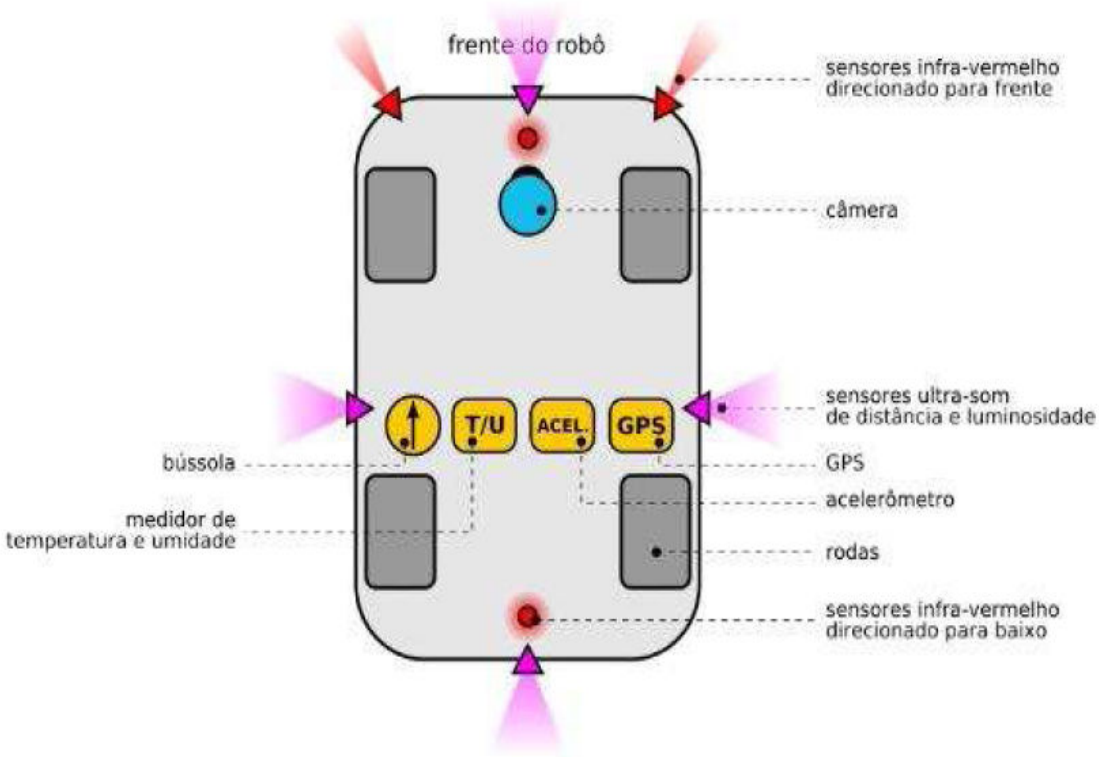

Figura 1. Disposição dos sensores do RoboDeck (Fonte: própria)

O MCR roda no processador ARM9, e é responsável por comandar os sensores e atuadores do robô. O MCS implementa o controle de identidade do robô, sendo responsável por armazenar informações como identificador único, nome, versão do robô, versão do protocolo de comunicação, entre outras. O MCC é responsável pelo controle de comunicação do robô, permitindo que controladores externos enviem comandos ao robô, que passa a atuar na modalidade *master-slave*. O controle de comunicação foi implementado com base no protocolo de comunicação sem fio zigbee. O MAP permite a integração de aplicativos robóticos capazes de atuar como controladores do RoboDeck. Os aplicativos podem expandir os comandos básicos do robô, fornecendo ao mesmo maior autonomia, como por exemplo, navegação autônoma inteligente.

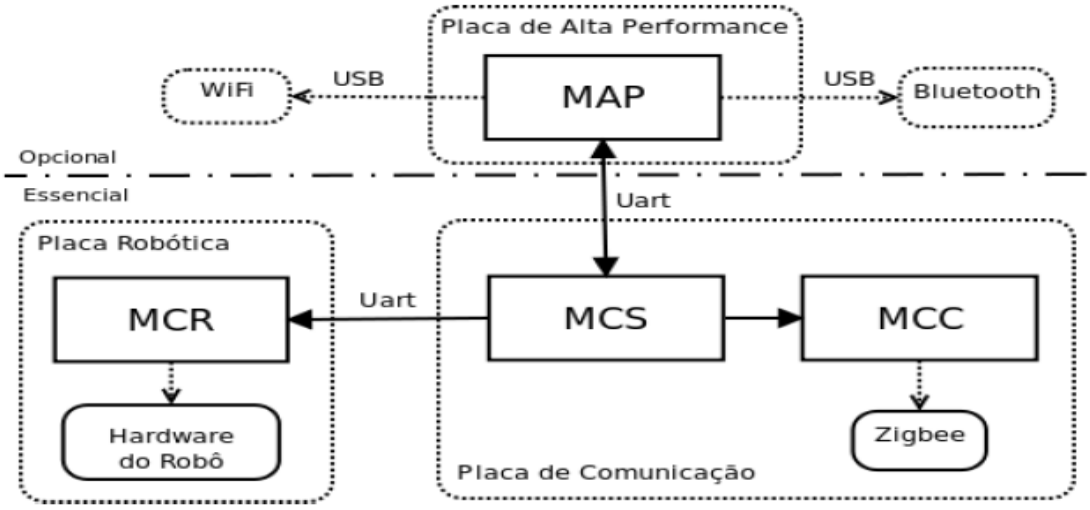

Figura 2. Visão geral da arquitetura de *software* do RoboDeck (Fonte: própria)

2. Tecnologia para comunicação da plataforma

Para o RoboDeck foram estudadas tecnologias que pudessem atender um nível de comunicação bidirecional sem fio que era necessário para permitir que existisse uma conectividade para, entre outras funcionalidades, auxiliar na integração de um robô a outros robôs (comunicação "peer to peer" entre vários robôs). A proposta foi utilizar *hardware* e protocolos de comunicação padrões para garantir interoperabilidade. Mesmo dentro da comunicação externa foram identificados dois subtipos de comunicação: *link* para controle e para transmissão de dados de alta velocidade. A primeira teve como características o uso contínuo (sempre ativo e não opcional), baixo consumo de energia, baixa velocidade (>10kbps) e alta latência (<0.1s). O segundo teve como características: opcional e de uso não continuo, alto consumo de energia elétrica, altas velocidades e baixa latência.

No estudo para identificar a melhor implementação de conectividade do robô tanto interna como externa, foram estudados: o Wi-Fi (IEEE 802.11x), o IEEE 802.15 *based* WPAN - Bluetooth (IEEE 802.15.1), ZigBee (IEEE 802.15.4) e 6lowpan: IPv6 *over Low power Wireless Personal Area Network*. É importante considerar que a RF (Rádio Frequência) não foi considerado, pois possui limitações na sua funcionalidade e não permite interoperabilidade, pois não considera padrões. As redes de sensores sem fio (WSN - *Wireless Sensor Networks*) têm importância estratégica no avanço do conceito M2M (*machine-to-machine*). Existem muitas ofertas proprietárias e padrões, comerciais e acadêmicas, mais as que estão em evidência são às baseadas no padrão IEEE 802.15.4 (WPAN - *Wireless Personal Area Network*).

Com respeito ao *link* para transmissão de dados de alta velocidade foi decidido que ficaria aberta a utilização de qualquer tecnologia disponível. Diante disso, a conexão com a plataforma robótica foi decidido que seria pela entrada USB usando um Wi-Fi USB *dongle*. Sobre o *link* para controle, depois de uma análise detalhada das alternativas disponíveis e considerando a experiência do grupo de desenvolvimento, foi decidido que deveria ser implementado usando ZigBee (Lee *et al.* 2007).

O ZigBee é um padrão definido por uma aliança de empresas de diferentes segmentos do mercado, chamada "ZigBee Alliance". Este protocolo foi projetado para permitir comunicação sem fio confiável, com baixo consumo de energia e baixas taxas de transmissão para aplicações de monitoramento e controle. Para implementar as camadas MAC (*Medium Access Control*) e PHY (*Physical Layer*) o ZigBee utiliza a definição 802.15.4 do IEEE, que opera em bandas de frequência livres. Atualmente, existem diversos padrões que definem transmissão em médias e altas taxas para voz, vídeo, redes de computadores pessoais, entre outros. Entretanto, até o presente momento ainda não surgiu um padrão que esteja de acordo com as necessidades únicas da comunicação sem fio entre dispositivos de controle e sensores. Os principais requisitos deste tipo de rede são baixa latência, otimização para baixo consumo de energia, possibilidade de implementação de redes com elevado número de dispositivos e baixa complexidade dos nós de rede (Kinney *et al.*, 2003).

Com relação aos modelos de rede, os componentes integrantes são o "coordenador", os "roteadores" e os "*end devices*". O "coordenador" inicia a rede definindo o canal de comunicação usado, gerencia os nós da rede e armazena informações sobre eles. Os "roteadores" são responsáveis pelo encaminhamento das mensagens entre os nós da rede. Já um "*end device*" pode ser bem um dispositivo bem mais simples, só se comunicando com outro nó da rede. Nas redes ZigBee um dispositivo pode permanecer um longo tempo sem ter que se comunicar. Além disso, o tempo de acesso à rede é muito pequeno, tipicamente 30 mseg. Outra característica importante é o tamanho reduzido dos pacotes de dados que trafegam na rede.

Além disso, a topologia Mesh que a rede ZigBee utiliza permite que essa rede se ajuste automaticamente em sua inicialização, na entrada de novos dispositivos ou perda de dispositivos. Nesta situação existem múltiplos caminhos entre os diferentes nós e a rede é autossuficiente para otimizar o tráfego de dados. Usando esta conFiguração é possível ter redes muito extensas, cobrindo largas áreas geográficas.

3. Desenvolvimento do MRAP

Finalizada a escolha da parte de comunicação, o foco foi transferido para o desenvolvimento do Módulo Robótico de Alta Performance (MRAP) para agregar poder de processamento e de comunicação de banda larga ao robô móvel, assim como propiciar-lhe a capacidade de navegação autônoma. Esse módulo foi desenvolvido para atuar como um controlador embarcado, substituindo ou colaborando com o controlador externo no controle do robô. Trata-se de um módulo de extensão e, portanto, o robô móvel não dependerá dele para realizar suas tarefas básicas. Por ser um módulo opcional também é permitido que o usuário tivesse a liberdade de desenvolver sua própria versão do módulo. Para facilitar o desenvolvimento por terceiros, tanto o protocolo de comunicação, como o módulo de comunicação de controle e o controlador foram planejado para serem robustos, abertos e bem documentados. Da mesma forma as bibliotecas de comunicação disponibilizadas.

Uma característica importante das aplicações robóticas que podem ser executadas neste módulo é a independência em relação ao *hardware*. Para isso, as aplicações foram desenvolvidas com o suporte de um sistema operacional Linux e com a capacidade de

interagir com os outros módulos valendo-se de interfaces e conectores conhecidos no mercado e suportados pelo sistema escolhido. Assim, espera-se que a portabilidade das aplicações desenvolvidas estará garantida para futuros *hardwares* permitindo que o núcleo robótico possa acompanhar o desenvolvimento tecnológico na área de processamento embarcado, prolongando a competitividade da plataforma no mercado. Como exemplo, pode-se pensar inclusive na utilização de celulares ou PDAs como substitutos deste módulo. Outro ponto importante do isolamento entre o *hardware* e as aplicações desse módulo foi à possibilidade de desenvolver, depurar e testar as aplicações em um computador *desktop* (PC) antes de compilá-las e carregá-las para o MRAP. Esta característica foi alcançada com o desenvolvimento de uma interface de controle robótico comum ao controlador e ao MRAP.

Na Figura 3 é exposta a organização computacional do MRAP e suas relações, onde são definidos os módulos computacionais e os requisitos de *hardware* necessários à implementação dos módulos computacionais. Antes da definição dos requisitos de *hardware* do MRAP foi preciso mapear e definir os módulos computacionais necessários para atingir seus objetivos.

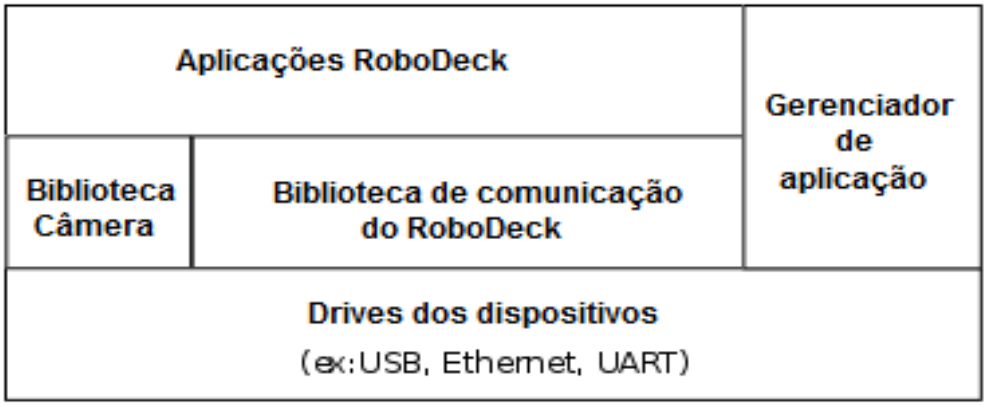

Figura 3. Módulos computacionais básicos do MRAP (Fonte: própria)

No desenvolvimento do *"SO device drivers"* o objetivo foi que este módulo representasse todos os *drivers* que o sistema operacional pudesse fornecer para a comunicação com cada uma das interfaces a serem utilizadas pelo MRAP. É o caso do USB para porta de comunicação genérica para dispositivos como câmera, Wi-Fi e *bluetooth*; Ethernet (porta para conexão por fio com a Internet ou para a carga de novos aplicativos); ou UART (*Universal Asynchronous Receiver/Transmitter*) como porta para a comunicação com o núcleo robótico. A instalação de aplicativos pode ser realizada tanto pela porta Ethernet quanto pela porta USB (Wi-Fi ou *bluetooth*). Esses módulos são parte integrante do sistema operacional (SO) e são dependentes exclusivamente do SO e do *hardware* utilizado.

O *camera library* é um módulo responsável por fornecer uma interface para a conexão de uma câmera a ser utilizada para o sistema de visão embarcada do robô. A câmera é conectada fisicamente à placa por uma porta USB e essa biblioteca fornece uma interface programática para o controle da câmera e para a captura de imagens. O "*Robotdeck communication library"*, assim como a biblioteca da câmera, fornece uma

interface programática para a comunicação com o módulo robótico implementando métodos que utilizam o "*device driver serial"* (UART) do sistema utilizado para este fim.

O módulo *Application manager* tem a responsabilidade de gerenciar os aplicativos a serem executados no MRAP. Como tarefas dessa gerência podem ser citadas: a carga e remoção de aplicativos, execução programada de aplicativos e leitura dos estados (propriedades) dos aplicativos. Esse módulo também é responsável por alternar entre o modo de execução "escravo" (o MRAP é utilizado apenas para repassar ao núcleo robótico, os comandos provenientes de uma de suas interfaces, como Wi-Fi, *bluetooth* ou Ethernet) e o modo de execução "autônomo" (o MRAP assume o controle do núcleo robótico passando o controle do robô a um de seus aplicativos). Por fim, o módulo "*Robotdeck application"* representa uma aplicação desenvolvida externamente e carregada por meio do "*application manager"* para o módulo MRAP.

As especificações do *hardware* do MRAP adotadas neste projeto foram: memórias (*flash* e RAM) e processador capazes de suportar alguma versão embarcada do sistema operacional (SO) Linux; uma porta USB conectando um adaptador para *bluetooth* ou Wi-Fi para a comunicação com o controlador; uma porta USB conectando uma câmera de captura de vídeo; e uma porta Ethernet para a reprogramação do MRPA e carga dos aplicativos.

O principal papel do SO Linux nesse módulo foi o de isolar as aplicações do *hardware* utilizado. Isto é, que os aplicativos desenvolvidos pudessem ser utilizados por *hardwares* distintos, mesmo que fosse necessário o ajuste ou substituição do SO. A interface de comunicação entre o aplicativo e o SO foi estabelecida usando chamadas ao sistema (*system calls*). Como todas as distribuições de Linux embarcadas também implementam a mesma interface (libc) ou um pelo menos um subconjunto dela, ao se adotar o Linux como SO tem-se a garantia de poder utilizar a distribuição que melhor se ajusta ao *hardware* adotado e que fornece uma API (*Application Programming Interface*) padrão para o desenvolvimento dos aplicativos. Outra vantagem ao se utilizar o Linux como SO é sua fácil personalização para novos *hardwares* e a existência de grupos e projetos internacionais preocupados com a portabilidade de alguma versão de Linux embarcado para as mais diversas plataformas de *hardware*.

4. Desenvolvimento do RoboDeck SDK

O kit de desenvolvimento de software (*Software Development Kit* – SDK) do RoboDeck (chamado RoboDeck SDK) têm dois objetivos: dar suporte aos serviços para o controle do RoboDeck por meio do Microsoft Robotics Developer Studio (MRDS ou Robotics Studio); e oferecer uma biblioteca independente do Robotics Studio, contendo todos os mecanismos necessários para controlar o robô.

O RoboDeck é equipado com um dispositivo de rede ZigBee. Pode também estar equipado com um módulo MRAP que permite a comunicação por meio das redes Bluetooth e/ou Wi-Fi. Entretanto, para um vídeo proveniente da câmera do robô foi usado exclusivamente a rede Wi-Fi, pois esta oferece melhor desempenho para o respectivo uso. Para um robô executar ordens de um controlador, foi necessário que uma

sessão de comunicação entre o robô e este controlador esteja aberta. Cada robô pode ter no máximo uma sessão aberta, ou seja, pode receber ordens de apenas um controlador. Para que outro controlador possa acionar o mesmo robô, é necessário que a sessão mais antiga seja fechada, para que seja aberta uma nova sessão com este outro controlador. A tabela 1 mostra as operações necessárias para efetuar o controle de sessão.

Tabela 1. Operações para o controle de sessão

Operação / descrição	Parâmetros	Resposta
Abrir sessão. Tenta adquirir uma sessão para comandar um robô.	Um nome a ser usado para a sessão.	Referência à sessão aberta ou notificação de fracasso.
Fechar sessão. Encerra uma sessão previamente aberta.		Notificação de sucesso ou de fracasso.

O RoboDeck é equipado com quatro motores de direção e dois motores de tração, usados para a movimentação do robô. Para o SDK desenvolvido foi criado comandos elementares para o controle independente sobre cada roda do robô. A tabela 2 exibe os comandos gerados. Também foram criados comandos de movimentação comumente usados, isto é, o desenvolvedor não precisa controlar o movimento de cada roda. Em vez disso, informa apenas o movimento que o robô deverá descrever. Estes comandos, por sua vez, coordenam o acionamento de cada roda a fim de realizar a movimentação desejada. A tabela 3 lista estes comandos.

Tabela 2. Comandos para controle sobre cada roda

Operação / descrição	Parâmetros	Resposta
Ligar / desligar motores de tração.	Quais motores serão ligados e quais serão desligados.	
Acionar motores de tração.	A velocidade que cada correspondente roda deverá tomar.	
Ligar / desligar motores de direção.	Quais motores serão ligados e quais serão desligados.	
Acionar motores de direção.	O ângulo que cada correspondente roda deverá tomar.	
Obter a quantidade de giros das rodas.		A quantidade de giros das rodas.
Zerar a quantidade de giros das rodas.		

Obter o período das rodas.		O período das rodas.

Tabela 3. Comandos de movimentação comumente usados

Operação / descrição	Parâmetros	Resposta
Mover o robô. Movimenta o robô para frente ou para trás.	A intensidade a ser aplicada nos motores de tração.	
Fazer curva. Movimenta o robô numa trajetória curvilínea.	O ângulo a ser aplicado nos motores de direção e a intensidade a ser aplicada nos motores de tração. Na verdade, estes valores oferecerão apenas uma base para que o próprio robô distribua os ângulos e intensidades definitivos para cada motor.	
Mover diagonalmente o robô. Movimenta o robô mantendo um mesmo ângulo em todos os motores de direção.	O ângulo que os motores de direção deverão tomar e a intensidade a ser aplicada nos motores de tração.	
Girar o robô. Gira o robô em torno de seu eixo.	O lado em que o robô deverá girar e a intensidade a ser aplicada nos motores de tração.	
Frear o robô. Trava todos os motores de tração e de direção do robô.		

O RoboDeck foi equipado com sensores infravermelhos e ultrassônicos que são empregados para a detecção de obstáculos relativamente distantes, bem como sensores ópticos podem ser conectados ao robô com a mesma finalidade. O robô possui também uma bússola, um acelerômetro, um sensor de temperatura e umidade, e um dispositivo de GPS. A tabela 4 mostra os comandos criados para atuarem no controle desses sensores.

Tabela 4. Comandos de sensoreamento

Operação / descrição	Resposta

Ler valores "brutos" dos sensores infravermelhos.	Os valores brutos obtidos pelos sensores.
Ler distância dos sensores infravermelhos.	A distância dos obstáculos detectados pelos sensores.
Ler distância com sensores ultrassônicos.	A distância dos obstáculos detectados pelos sensores.
Ler luminosidade com sensores ultrassônicos.	A luminosidade obtida pelos sensores.
Verificar sensores ópticos.	A detecção ou não de algum objeto em cada um dos sensores.
Ler a bússola.	A orientação do robô segundo a bússola.
Ler o acelerômetro.	A força "g" que atua sobre o robô.
Ler valores brutos do sensor de temperatura e umidade.	Os valores brutos de temperatura e umidade.
Ler valores normalizados do sensor de temperatura e umidade.	Os valores normalizados de temperatura e umidade.
Testar a validade de dados provenientes do GPS.	Validade ou invalidade dos dados. O comando serve para verificar se o sinal está mal calibrado ou valores incoerentes.
Obter o número de satélites encontrados pelo GPS.	O número de satélites encontrados.
Obter horário com o GPS.	Horário no formato hora/minutos/segundos.
Obter data com o GPS.	Data no formato mês/dia/ano.
Obter latitude com o GPS.	Latitude no formato graus/minutos/segundos.
Obter longitude com o GPS.	Longitude no formato graus/minutos/segundos.
Obter altitude com o GPS.	Altitude no formato graus/minutos/segundos.
Obter a velocidade do robô através do GPS.	A velocidade do robô no formato onde o valor é de metros por segundo.
Obter a direção e sentido de deslocamento do robô através do GPS.	A direção e sentido de deslocamento do robô.
Verificar carga das baterias.	A carga das baterias.

O RoboDeck foi desenvolvido de forma a suportar módulos de expansão. Diante disso, o SDK desenvolvido precisou suportar a troca de mensagens com estes novos módulos. A tabela 5 exibe os comandos criados. Com a presença de um MRAP com rede Wi-Fi, o RoboDeck equipado com uma câmera é capaz de transmitir vídeo em tempo real. A tabela 6 exibe os comandos desenvolvidos especificamente para transmissão de vídeo.

Tabela 5. Comandos para módulos de expansão

Operação / descrição	Parâmetros	Resposta
Enviar mensagem.	O slot e a mensagem a ser enviada.	
Receber mensagem.	O slot.	A mensagem recebida, quando houver.

Tabela 6. Comandos associados à transmissão de vídeo

Operação / descrição	Resposta
Iniciar transmissão de vídeo.	
Obter próximo quadro (*frame*). Uma vez iniciada a transmissão, obtém o próximo quadro (relativo à execução anterior desta mesma operação) obtido através do canal existente entre o controlador e o robô.	Um quadro do vídeo.
Parar transmissão de vídeo.	

Em geral, todos os comandos enviados ao robô retornam/geram uma resposta ao solicitante. Tal resposta retorna quase que imediatamente. Contudo, este "imediatamente" depende do tempo que o robô leva para processar a mensagem e dos atrasos encontrados no meio da comunicação. Um modelo síncrono de comunicação é mais simples, mas fará com que o programa controlador fique bloqueado, aguardando a resposta do robô. Este tempo de bloqueio poderá ser considerado grande demais, dependendo da natureza da aplicação. Um modelo assíncrono é, por sua vez, mais complexo, mas não bloqueia o programa controlador. Quando a resposta chega, o programa é devidamente avisado. Avaliando estes aspectos, optou-se por adotar a forma assíncrona em todo o RoboDeck SDK. Assim, apesar dos programas se tornarem mais complexos, seu desempenho é favorecido.

O Robotics Studio é baseado em linguagens direcionadas à *Common Language Infrastructure* (CLI), como C#, VB.NET e C++/CLI. Entretanto, a maior parte da documentação do Robotics Studio está em C#. Fica claro que a Microsoft vem dando mais atenção a esta linguagem. Portanto, optou-se por empregar C# como linguagem para elaboração dos serviços do RoboDeck para o Robotics Studio. O objetivo primário do RoboDeck SDK é dar suporte aos serviços para o Robotics Studio. No entanto, objetiva-se também ter este SDK como um produto independente. Assim, este SDK deve empregar uma linguagem bem consolidada, mas que possa ser integrada aos serviços em C#. A linguagem C++ vem então a ser a opção mais conveniente. Contudo, como é possível observar a seguir, não se empregou a linguagem C++ pura, mas sim uma extensão desta linguagem definida pela Microsoft e chamada C++/CLI.

É impossível desacoplar completamente do sistema operacional um *software* como o RoboDeck SDK escrito em C++. Características como o controle de *threads* e o acesso à porta serial são muito dependentes do sistema operacional. Diante disso, haverá necessariamente certa dependência, que no mínimo aparecerá na biblioteca empregada para o controle de *threads*, por exemplo. No Windows, há pelo menos três formas de se trabalhar com estas *threads*: por meio da C Run-Time Library (CRT); do Microsoft Foundation Classes (MFC); e do Microsoft ".NET" *Framework*. Observando que a Microsoft vem incentivando o emprego desta última tecnologia, optou-se por adotá-la no RoboDeck SDK. Mais ainda, o emprego do ".NET" *Framework* torna mais simples o desenvolvimento do SDK. O ".NET" *Framework* não opera com a linguagem C++ pura, mas sim com a C++/CLI, uma extensão da linguagem C++ definida pela Microsoft. Esta linguagem estendida surge exatamente para dar suporte ao ".NET" *Framework*. Consequentemente o RoboDeck SDK emprega a linguagem C++/CLI.

Visando a portabilidade do RoboDeck SDK para outros sistemas operacionais além do Windows, torna-se atrativo separar as partes dependentes das independentes do sistema operacional, e então implementar as partes independentes através da linguagem C++ pura. C++/CLI define elementos denominados gerenciados, que são elementos manuseados pelo ambiente de execução do ".NET" *Framework* (chamado *Common Language Runtime*, ou CLR). Já a linguagem C++ pura só define elementos não-gerenciados. Acontece que surgem alguns problemas de compatibilidade quando se misturam elementos gerenciados (provenientes do uso da C++/CLI) com elementos não-gerenciados (provenientes do uso da C++ pura). Como exemplo, uma classe gerenciada não permite que se agregue diretamente um objeto não-gerenciado. Isto não impede, mas atrapalha muito o desenvolvimento dos programas. Deve-se lembrar ainda que os serviços para o Robotics Studio são escritos em C#, uma linguagem voltada ao .NET Framework e consequentemente baseada em objetos gerenciados.

A mistura das duas linguagens também quebra a uniformidade do código. Estruturas diferentes são utilizadas com o mesmo propósito ao longo do programa. Como exemplo, a classe std::vector (C++ pura) que pode ser empregada com o mesmo objetivo que a classe cli::array (C++/CLI). Avaliando todos estes aspectos, adotou-se somente a linguagem C++/CLI ao longo de todo o RoboDeck SDK.

A essência do RoboDeck SDK está na troca de mensagens com o robô. Assumindo uma das redes de comunicação (ZigBee, Bluetooth ou Wi-Fi), duas são as principais classes envolvidas no modelo de comunicação: SerialCommunication e uma classe específica à rede adotada. Aqui usa-se a classe ZigBee como exemplo. A Seção As Classes do RoboDeck SDK exibe a estrutura das classes SerialCommunication e ZigBee. As mensagens manipuladas pelo SDK seguem dois protocolos distintos: o Protocolo do RoboDeck e o Protocolo da Rede ZigBee. As mensagens da rede ZigBee caracterizam-se por "empacotar" as mensagens do RoboDeck.

O modelo assíncrono de comunicação entre o controlador e o robô exige que o programa do usuário fique pouco tempo bloqueado a espera do trabalho do SDK. Portanto a arquitetura do RoboDeck SDK está baseada em diversas *threads* trabalhando concorrentemente. A Figura 4 ilustra as estruturas envolvidas no processo de comunicação. Na sequência, descrevem-se cada um dos componentes deste modelo.

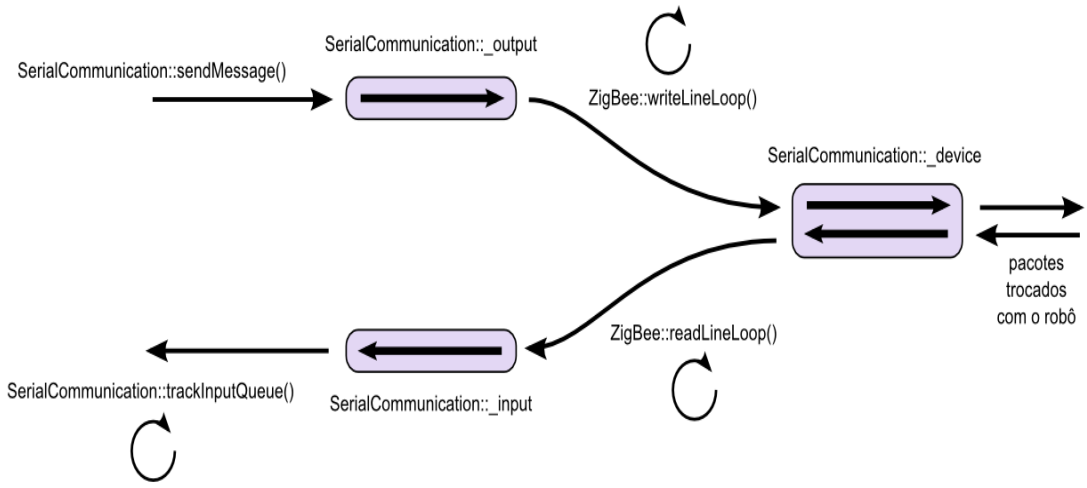

Figura 4. As estruturas envolvidas no modelo de comunicação (Fonte: própria)

Tabela 7. Descrição dos componentes do modelo de comunicação

Componentes	Descrição
SerialCommunication::_output SerialCommunication::_input( )	Representam respectivamente as filas de saída e de entrada de mensagens. São objetos da classe System::Collections::Queue e armazenam mensagens segundo o protocolo do RoboDeck. A existência destas estruturas torna mais organizado o modelo de troca de mensagens. A fila de saída, em particular, impede também que o programa do usuário fique bloqueado por muito tempo, como visto adiante no método sendMessage.
SerialCommunication::_device( )	Representa o dispositivo de comunicação serial. Trata-se de um objeto da classe System::IO::Ports::SerialPort e recebe as mensagens segundo o protocolo ZigBee.
SerialCommunication::sendMessage( )	Método responsável por inserir as mensagens na fila de saída. Sua execução roda na mesma thread do programa do usuário. Portanto, uma vez invocado, o programa do usuário estará esperando o término da execução deste método. Isto justifica o emprego da fila de saída. Desta forma, o método simplesmente insere a mensagem na fila de saída e já retorna a execução ao programa do usuário.
SerialCommunication::trackInputQueue( )	Método responsável por receber as mensagens da fila de entrada e invocar os correspondentes métodos que tratarão cada uma destas mensagens (*callbacks*). A execução de trackInputQueue se dá numa thread própria e permanece em um laço, monitorando constantemente a fila de entrada. Uma nova thread é aberta para cada callback invocado.

ZigBee::writeLineLoop( )	Método responsável por coletar as mensagens da fila de saída, empacotá-las segundo o protocolo ZigBee, e então encaminhá-las para o dispositivo de comunicação serial. Sua execução também se dá numa thread própria e permanece em um laço, mas aqui monitorando a fila de saída.
ZigBee::readLineLoop( )	Método responsável por receber as mensagens provenientes do dispositivo de comunicação serial, extrair as mensagens segundo o protocolo do RoboDeck, e então inserir estas últimas na fila de entrada. Assim como o método anterior, sua execução também se dá numa thread própria e permanece em um laço, aqui monitorando o dispositivo de comunicação serial.

O RoboDeck SDK é orientado a objetos e portanto foi modelado com um conjunto de classes e seus relacionamentos. As classes funcionam como ponto de acesso ao SDK a partir do programa desenvolvido pelo usuário. Agrega todos os componentes do robô, como os motores de tração e direção e os sensores. Agrega também o objeto responsável pela comunicação serial, além das características como endereço do controlador, endereço do robô e sessão aberta. O RoboDeck SDK define uma classe para cada grupo de componentes, onde são definidas as devidas operações. Por exemplo, a classe "ComponentBase" representa a superclasse dos componentes, concentrando características comuns a eles.

O RoboDeck SDK é provido de uma hierarquia de classes capaz de organizar as mensagens de acordo com o protocolo do robô. "Message" é a classe-base das mensagens do RoboDeck e mantém as características comuns a qualquer mensagem. "Command" representa as classes de comando. Cada comando deve ter um identificador único. Para tanto, esta classe mantém um atributo estático (classId) usado como referência para suas instâncias obterem os identificadores. O acesso a este atributo é sincronizado. Dividida em três subclasses – PartialResponse, FinalResponse e Error – a classe "Response" representa as mensagens de resposta provenientes do robô.

O RoboDeck SDK define classes responsáveis pela comunicação serial entre o controlador e o robô. A cada uma das redes de comunicação (ZigBee, Bluetooth e Wi-Fi) corresponde uma classe dentro do SDK. Estas classes contêm características particulares a cada uma das redes. Já a classe SerialCommunication, por sua vez, define características que não dependem da rede utilizada. "_device" representa o dispositivo de comunicação. "_input" e "_output" representam respectivamente as filas de entrada e de saída. "_callbackTable" é uma tabela que associa os identificadores das mensagens de comando (enviadas pelo controlador) aos respectivos métodos (*callbacks*) que tratarão as respostas referentes a estes comandos.

O RoboDeck SDK também define uma estrutura de *callbacks*. Estes *callbacks* são responsáveis por tratar assincronamente as diversas mensagens de resposta que trafegam pelo SDK. Todas definem um método chamado "doCallback", o qual contém o programa a ser executado. A cada comando que o usuário invoca sobre o robô, corresponde a uma resposta proveniente do SDK. Tal troca de mensagens é assíncrona, e o programa do usuário não deve ficar bloqueado esperando estas respostas. Para tanto, o usuário deve definir as ações a serem executadas a cada resposta por ele recebida.

Então ele deve criar classes que estendam “UserCallback” e implementam o método “doCallback”. Este é o método invocado quando a referente resposta chegar.

“SDKCallback”, por sua vez, é a superclasse de todos os *callbacks* empregados internamente pelo SDK. Funcionam como *callbacks* intermediários, por isso mantêm referência a *callbacks* do usuário. Então o sistema chama estes *callbacks* do SDK, os quais realizarão as devidas tarefas e depois chamarão os *callbacks* do usuário.

Alguns comandos solicitados pelo usuário foram quebrados em múltiplos comandos pelo SDK. Para estes casos empregou-se classes derivadas de SyncCallback (*Synchronized Callback*), que agregam objetos de sincronização (*Callbacks*ynchronizer) e coordenam as múltiplas respostas que virão para o SDK. Tal sincronização faz com que o programa do usuário receba apenas uma (e não múltiplas) resposta para cada comando por ele solicitado. GeneralCallback compartilha ações comuns a grande parte dos comandos empregados pelo SDK. Diversos outros *callbacks* são definidos para ações particulares nos mais variados comandos do RoboDeck.

5. Resultados

Para a escolha do *hardware* adotado para o MRAP do robô alguns quesitos práticos foram utilizados para guiar esta escolha. Um dos principais foi averiguar a performance, capacidade e limitações da especificação mínima do hardware para o MRAP. Diante dos quesitos adotados e pela própria natureza "opcional" do MRAP, não foi possível utilizar um único *hardware* para o desenvolvimento do MRAP sem o ônus de desenvolver uma infraestrutura computacional "dependente" no SO do *hardware* adotado, além do ônus de não saber os limites impostos pelas características de hardware como memória disponível. Por isto optou-se por utilizar inicialmente os kits de desenvolvimento SMDK2500 da Samsung e equipamento NAS200 da Linksys. Contudo eles demoraram muito para chegar e isto causou um atraso no projeto.

Com base na arquitetura adotada para o robô móvel, tanto as responsabilidades do *hardware* quanto do *software* puderam ser atribuídas a módulos distintos, dependendo do poder de processamento e da banda de comunicação exigida. Como exemplo, no caso de um módulo de visão pode-se restringir o *hardware* a apenas uma câmera digital USB e implementar todo o processamento como um aplicativo no MRAP. Isto facilitou a adoção de aplicativos de visão computacional pela plataforma robótica.

Neste projeto, como exemplo de aplicação robótica, foi desenvolvida uma aplicação de visão embarcada. Esta aplicação foi utilizada para testar tanto o *framework* de desenvolvimento de aplicações robóticas, quanto à comunicação de banda larga (foi testado o envio de imagens digitais "cruas" do robô para algum controlador). Para ser implantado o modelo computacional exposto foi necessário se apoiar em um *hardware* computacional mínimo. Para o MRAP dentro desse projeto, duas especificações de *hardware* foram definidas. A primeira foi à conFiguração mínima necessária à implementação de qualquer *hardware* que venha a ser utilizado para o MRAP. A segunda especificação foi uma conFiguração utilizada no projeto e que teve como objetivo exercitar as potencialidades do módulo ao mesmo tempo em que serviu de

exemplo para a construção de módulos mais avançados. As especificações mínimas do *hardware* do MRAP adotadas foram:

- Memórias (Flash e RAM) e processador capazes de suportar alguma versão embarcada de um sistema operacional que implemente as chamadas básicas de sistema (*system calls*) para utilização dos "*devices*" utilizados (USB, Ethernet e UART). Também, deve haver espaço na memória *Flash* para o armazenamento de ao menos um aplicativo que poderá ser tomado como "residente" ao robô e que será executado após o "*boot*" do sistema.
- Uma porta (USB, Ethernet ou serial), a ser utilizada para a reprogramação do módulo e carga dos aplicativos;
- Ao menos um conector (porta) UART exclusivo para a comunicação como o núcleo robótico.

6. Considerações finais

Foi criada uma abstração para os módulos lógicos para representar os componentes eletrônicos que compõem o robô e que possuem independência física (placas eletrônicas) e lógica (processamento). Todos esses módulos possuem um *hardware* e um *software* que juntos permitem que exerça a função para a qual foi projetado. Isto auxiliou na parte do projeto referente à comunicação em tempo real para execução de manobras complexas. Com os comandos robóticos introduzidos nos aplicativos, (por exemplo: abertura de Seção do RoboDeck, solicitação de leitura de sensores, solicitação de acionamento de motores de tração e direção do robô, solicitação de parada e fechamento de Seção, entre outros) os mesmos se comportaram como “controladores” do robô. Para tanto, os controladores interagiram diretamente com o MAP (Módulo de Alta Performance), que funcionou como um *broker* entre os “controladores” robóticos e o MCR (Módulo de Controle Robótico, que acessa diretamente o hardware do robô).

Para facilitar a construção de futuros aplicativos robóticos, foi criada uma API (escrita em linguagem C) que possibilitará a utilização de todos os comandos de interação com o MAP. Essa API faz parte do SDK do RoboDeck, cujos usuários são os programadores de aplicativos robóticos da plataforma. Os aplicativos desenvolvidos nesse projeto servem de exemplos, para programadores do RoboDeck, de como utilizar os recursos de visão computacional do robô, bem como a API disponibilizada.

Em relação a projetos futuros, propõe-se o desenvolvimento de sistemas computacionais baseado em robótica envolvendo tecnologias de *Remote Presence System* (RPS) e Processamento de sinais e imagens para a área de tecnologia educacional. Além disso, a geração de material didático baseado em uma metodologia educacional ativa para compor um kit educacional que tem como foco principal o ensino médio técnico de três disciplinas: robótica, microcontroladores e tecnologia de comunicação. Essas disciplinas fazem parte do plano de ensino dos cursos de mecatrônica e eletrônica dos SENAIs e ETECs de todo Brasil. É importante comentar que uma educação técnica de qualidade que permita o domínio da tecnologia de construção de sistemas complexos como, por exemplo, robôs móveis inteligentes, são fatores de diferenciação e valor agregado. Se hoje existe uma empresa brasileira que

desenvolve, fabrica e comercializa aviões e que gera empregos qualificados, impostos de valor agregado, e divisas para o País; muito se deve a uma quantidade adequada de mão de obra qualificada associada a um forte investimento público inicial, e posteriormente, a gestão da iniciativa privada para tornar o negócio competitivo. Atualmente, no País existem poucas ferramentas para o desenvolvimento e aprimoramento de profissionais com alto-desempenho. Nossos técnicos e engenheiros, enquanto estudantes, não têm acesso para atuar com conteúdo multidisciplinar que, por exemplo, a robótica proporciona.

Como contexto estratégico, o aprendizado técnico na área de robótica, pode permitir no médio prazo um diferencial competitivo para o País, pois promove a melhoria do aprendizado de futuros profissionais para trabalhar em projetos com alto valor agregado.

6. Agradecimentos

Ao apoio do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) por meio do seu programa de Desenvolvimento Tecnológico e Extensão Inovadora (DT) e do programa de Formação de Recursos Humanos em Áreas Estratégicas (RHAE).

Referências

Kinney, P. et al. (2003) "Zigbee technology: Wireless control that simply works" In: Communications design conference, p. 1-7.

Lee, J. S. et al. (2007) "A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi" In: Industrial Electronics Society. IECON 2007. 33rd Annual Conference of the IEEE. IEEE, p. 46-51.

Menezes, M. C. et al. (2017) "Mapeamento e Localização para o kit Robótico RoboDeck" http://sistemas.deinf.ufma.br/anaisjim/artigos/2016/201604.pdf [Visitado em maio de 2017]

Motta, B. C. (2016). "Aprendizagem por demonstração baseada em redes neurais artificiais aplicada à robótica móvel" http://www.bdm.unb.br/handle/ 10483/13282 [Visitado em fevereiro de 2017]

Pissardini, R. S. (2014) "Veículos autônomos de transporte terrestre: proposta de arquitetura de tomada de decisão para navegação autônoma" Dissertação de mestrado. Universidade de São Paulo. http://www.teses.usp.br/teses/disponiveis/ 3/3138/tde-26082015-161805/en.php [Visitado em janeiro de 2016]

Wei, D. C. M. (2015) "Método de desvio de obstáculos aplicado em veículo autônomo", Dissertação de mestrado. Universidade de São Paulo, 2015 http://www.teses.usp.br/teses/disponiveis/3/3138/tde-17062016-142254/en.php [Visitado em janeiro de 2016]

Zanola, L. et al. (2017) "Implementação com Validação Real de um Controle Proporcional, Integral e Derivativo na Plataforma Robótica RoboDeck" http://www.xbot.com.br/wp-content/uploads/2012/10/artigo_Implementacao_ ValidacaoReal_leandro.pdf [Visitado em janeiro de 2017]