COMPUTE
Volume 4, Número 1

Desenvolvimento de uma giga de teste para plataforma robótica móvel

Antonio Valerio Netto[1]

[1]Desenvolvimento Tecnológico e. Extensão Inovadora – DT CNPq
SHIS QI 1 Conjunto B – Brasília – DF – Brasil

Abstract. *In this article we report the development of a Software Development Kit (SDK) to facilitate the creation of applications for a mobile robotic platform used in the areas of education and research, called RoboDeck. Based on this SDK, a test bench was created to be used with the Microsoft Windows operating system. The purpose of this bench was to allow the validation of the proposed SDK as well as to test RoboDeck functionality in a quick and practical way when it is received by some research or teaching institution that acquired it. There was the difficulty of running a checklist to see if the robot was fit to be used. That is, it had not suffered any problems during its transportation or coming with some manufacturing defect. With the development of this bench, it is currently possible, in little more than 15 minutes, to verify the main functionalities of the robot.*

Resumo. *Nesse artigo é relatado o desenvolvimento de um Software Development Kit (SDK) para facilitar a criação de aplicações para uma plataforma robótica móvel empregada nas áreas de educação e pesquisa, chamada RoboDeck. Baseada neste SDK foi criada uma giga de testes para ser utilizado com o sistema operacional Windows da Microsoft. Essa giga teve como objetivo, permitir validar o SDK proposto como também fazer testes das funcionalidades com o RoboDeck de forma rápida e prática quando o mesmo é recebido por alguma instituição de ensino ou pesquisa que o adquiriu. Existia a dificuldade de fazer um checklist de funcionamento para verificar se o robô estava apto para ser utilizado. Isto é, não tinha sofrido nenhum problema durante o seu transporte ou vindo com algum defeito de fabricação. Com o desenvolvimento dessa giga, atualmente é possível, em pouco mais de 15 minutos, verificar as principais funcionalidades do robô móvel.*

1. Introdução

O RoboDeck é uma plataforma robótica móvel que foi desenvolvida com o intuito de promover a criação de aplicações nas áreas de educação técnica e de pesquisa. O mesmo é constituído por um chassi mecânico, motores elétricos, rodas, conjunto de eletrônica embarcada para controle e sensoriamento, um ambiente de programação baseado na linguagem C/C++, entre outros dispositivos e funcionalidades [Netto and Bertone 2018]. Por se tratar de uma plataforma universal de código aberto, o usuário tem total liberdade para desenvolver aplicações, modificar os códigos pré-existentes por meio de bibliotecas computacionais, além de acrescentar novos módulos eletrônicos. Exemplos de aplicações desenvolvidas utilizando a plataforma podem ser encontradas em diversos artigos científicos, como: Be-

zerra (2012), [Zanolla et al. 2017], [Pissardini 2014], [Wei 2015], [Menezes et al. 2017], [Motta 2015], [Valerio Netto et al. 2018] e [Pinto and Netto 2018].

Em sua versão 2.5 (XBot, 2016), o RoboDeck teve mudanças em sua estrutura física (hardware) com a diminuição de peso devido ao novo chassi em alumínio e com a mudança do material da tampa superior para o acrílico. Com essas modificações, a autonomia aumentou em torno de 15% e foi possível eliminar interferências que a bússola vinha sofrendo com o antigo material em que o chassi superior era fabricado. Relacionada à parte de comunicação, o módulo Wi-Fi, após o *upgrade*, permitiu fazer conexões tanto ponto a ponto, quanto multiponto, ou seja, ele poderia se conectar a uma rede Wi-Fi já existente, permitindo acesso a Internet. A comunicação ZigBee foi incorporada na placa de alto-desempenho, permitindo aumentar a velocidade de transmissão e do *buffer* de comandos. Contudo, faltava uma forma de realizar testes das funcionalidades do RoboDeck de forma rápida e prática quando o mesmo era adquirido por alguma instituição de ensino ou pesquisa. Existia a dificuldade de fazer um *checklist* de funcionamento para verificar se o robô estava apto para ser utilizado. Isto é, não tinha sofrido nenhum problema durante o seu transporte ou vindo com algum defeito de fabricação.

Diante desse fato, foi proposto o desenvolvimento de uma giga de testes para ser utilizado com o sistema operacional Windows da Microsoft. Para a criação dessa giga foi necessário gerar um *Software Development Kit* (SDK) para facilitar o desenvolvimento de aplicações para o RoboDeck. Neste artigo é descrito esse desenvolvimento da giga e o aprimoramento do SDK que foi necessário para contemplar à modelagem do sistema proposto. A giga é capaz de informar as leituras de distância dos sensores ultrassônicos e infravermelhos. O sensor ultrassônico é capaz de medir obstáculos até seis metros de distância, porém esses seis metros são distribuídos dentro de um raio de alcance de até 45º. Esse tipo de sensor é de extrema importância, pois são eles que na maioria das vezes impedem colisões. Já os sensores Infravermelhos utilizados nas laterais do robô conseguem medir até 70 cm de distância. Os robôs podem ser controlados tanto de forma autônoma quanto teleoperado (master-slave). A giga também permite capturar os dados do acelerômetro, da bússola e do sensor de temperatura e umidade. Além disso, é capaz de exibir os dados obtidos por meio do GPS (*Global Positioning System*) como: altitude, latitude, longitude, velocidade, sentido, data e hora. A giga também captura dados da câmera que é capaz de transmitir imagens em tempo real e fotos em formato digital. Sua forma de comunicação é por meio de Wi-Fi e ZigBee. Com o Wi-Fi é possível controlar o robô a uma distância de até 100 metrôs. Já o ZigBee, por ser uma rede mais robusta por oferecer resistência ao ruído, é capaz de atingir uma distância de até 1000 metros [Kinney et al. 2003].

2. Modelagem da biblioteca SDK

Trata-se de uma biblioteca de classes na linguagem C++ empregada como interface entre uma aplicação desenvolvida pelo usuário e o robô propriamente dito. Este SDK teve dois objetivos: o primeiro foi dar suporte aos serviços para o controle do robô por meio do *Microsoft Robotics Developer Studio* (Robotics Studio), e o segundo, foi oferecer uma biblioteca independente contendo todos os mecanismos necessários para controlar o robô móvel. O documento de requisitos criado foi baseado nos recursos que a plataforma robótica possuía (seus sensores eletrônicos e possibilidades de controle tanto do motor quanto da comunicação).

O RoboDeck foi equipado com um dispositivo de rede ZigBee e com um módulo de alta performance que permite a comunicação por meio das redes: *bluetooth* e Wi-Fi. O vídeo proveniente da câmera do robô é usado exclusivamente pela rede Wi-Fi, pois esta oferece melhor desempenho para transmissão de imagens digitais. Para um robô executar ordens de um controlador, há necessidade de abrir uma sessão de comunicação entre este robô e esse controlador. 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. A tabela 1 mostra as operações necessárias para efetuar o controle de sessão. O robô também é equipado com quatro motores de direção e dois motores de tração usados para a movimentação do robô. O SDK oferece comandos elementares para o controle independente sobre cada roda do robô. A tabela 2 exibe tais comandos.

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.

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.

O SDK oferece também comandos de movimentação comumente usados. O desenvolvedor não precisa controlar o movimento de cada roda. Em vez disso, deve informar 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. O robô também é equipado com diversos tipos de sensores. Os sensores infravermelhos e ultrassônicos 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, além de um dispositivo de GPS. A tabela 4 mostra os comandos que atuarão nestes sensores.

Tabela 3. Comandos de movimentação comumente usados

Operação / descrição	Parâmetros
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.	
Zerar a quantidade de giros das rodas.	
Obter o período das rodas.	

O robô foi construído de forma a suportar módulos de expansão. Diante disso, o SDK suporta a troca de mensagens com estes módulos. A tabela 5 exibe os comandos que o SDK contém. Com a presença de um módulo de alta performance que suporte à rede Wi-Fi, o robô poderá estar equipado com uma câmera capaz de transmitir vídeo. A tabela 6 exibe os comandos que o SDK oferece para transmissão de vídeo.

3. Modelo de comunicação empregado

Em geral, qualquer comando enviado ao robô corresponde a uma resposta. Tal resposta retorna quase que imediatamente. Contudo, essa resposta depende do tempo que o robô leva para processar a mensagem e dos atrasos encontrados durante o processo de 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 mais complexo, contudo não bloqueia o programa controlador. Quando a resposta chegar, o programa será devidamente avisado. Avaliando estes aspectos, optou-se por adotar a forma ASSÍNCRONA em todo o desenvolvimento do SDK. Assim, apesar dos programas ficarem mais complexos, seu desempenho é favorecido.

Adotando uma das redes de comunicação existentes no robô (ZigBee, bluetooth ou Wi-Fi), duas classes são envolvidas no modelo de comunicação: *SerialCommunication* e uma classe específica da rede de comunicação adotada. Na figura 1 é usada a classe ZigBee como exemplo. As mensagens manipuladas pelo SDK seguem dois protocolos distintos: o protocolo do robô e o protocolo da rede ZigBee. As mensagens da rede ZigBee caracterizam-se por "empacotar" as mensagens do robô.

SerialCommunication:: output e SerialCommunication:: input
Representam respectivamente as filas de saída e de entrada de mensagens. São objetos

Tabela 4. Comandos referentes aos sensores

Operação / descrição	Resposta
Ler valores “crus” dos sensores infravermelhos.	Os valores crus 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 crus do sensor de temperatura e umidade.	Os valores crus 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.
Obter o número de satélites encontrados pelo GPS.	O número de satélites encontrados.
Obter horário com o GPS.	Horário.
Obter data com o GPS.	Data.
Obter latitude com o GPS.	Latitude.
Obter longitude com o GPS.	Longitude.
Obter altitude com o GPS.	Altitude.
Obter a velocidade do robô através do GPS.	A velocidade do robô.
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.

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. Parar transmissão de vídeo.	Um quadro do vídeo.

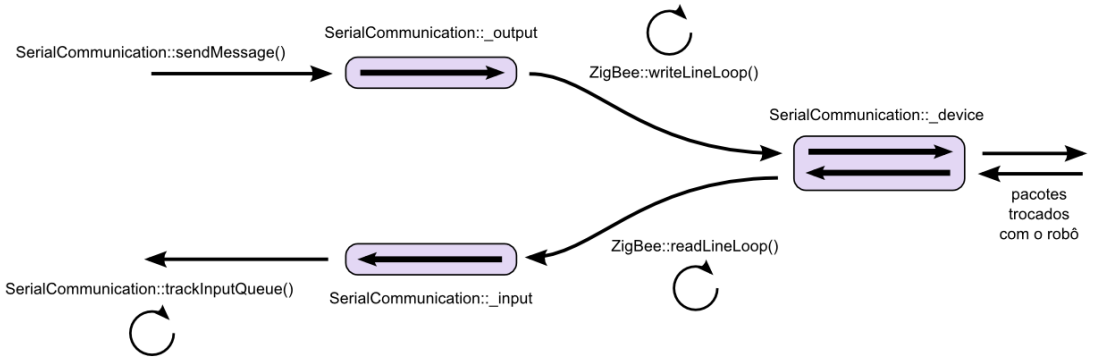

Figura 1. Estruturas envolvidas no modelo de comunicação

da classe *System::Collections::Queue* e armazenam mensagens segundo o protocolo do RoboDeck. A existência destas estruturas torna mais organizada 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 monito-

rando 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.

3.1. Classes e mensagens

O SDK é orientado a objetos, portanto é modelado com um conjunto de classes e seus relacionamentos. Os tópicos a seguir tratam destas classes, destacando seus objetivos e principais aspectos. A classe "*RoboDeck*" funciona como ponto de acesso ao SDK a partir do programa do usuário. Agrega todos os componentes do robô, como os motores de tração, os motores de direção e os diversos sensores. Agrega também o objeto responsável pela comunicação serial e mantém referência a características como endereço do controlador, endereço do robô e sessão aberta. A figura 2 mostra a estrutura desta classe, como também algumas enumerações empregadas ao longo do SDK.

O SDK define uma classe para cada grupo de componentes, onde são definidas as devidas operações. A classe *ComponentBase* representa a superclasse dos componentes, concentrando características comuns a eles. As figuras 3 e 4 mostram a estrutura destas classes.

O SDK é provido de uma hierarquia de classes capaz de organizar as mensagens de acordo com o protocolo do robô. A figura 5 exibe esta hierarquia. *Message* é a classe-base das mensagens do "*RoboDeck*" e mantém as características comuns a qualquer mensagem. *Command* representa as mensagens de comando. Cada comando deverá 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 deve ser sincronizado. Dividida em três subclasses: "*PartialResponse*", "*FinalResponse*" e "*Error*". A classe *Response* representa as mensagens de resposta provenientes do robô.

O SDK define classes responsáveis pela comunicação serial entre o controlador e o robô. A cada uma das redes de comunicação (ZigBee, *bluetooth* e WiFi) 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.

3.2. Estrutura de *callbacks*

O SDK também define uma estrutura de *callbacks*. Eles 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, corresponderá uma resposta proveniente do

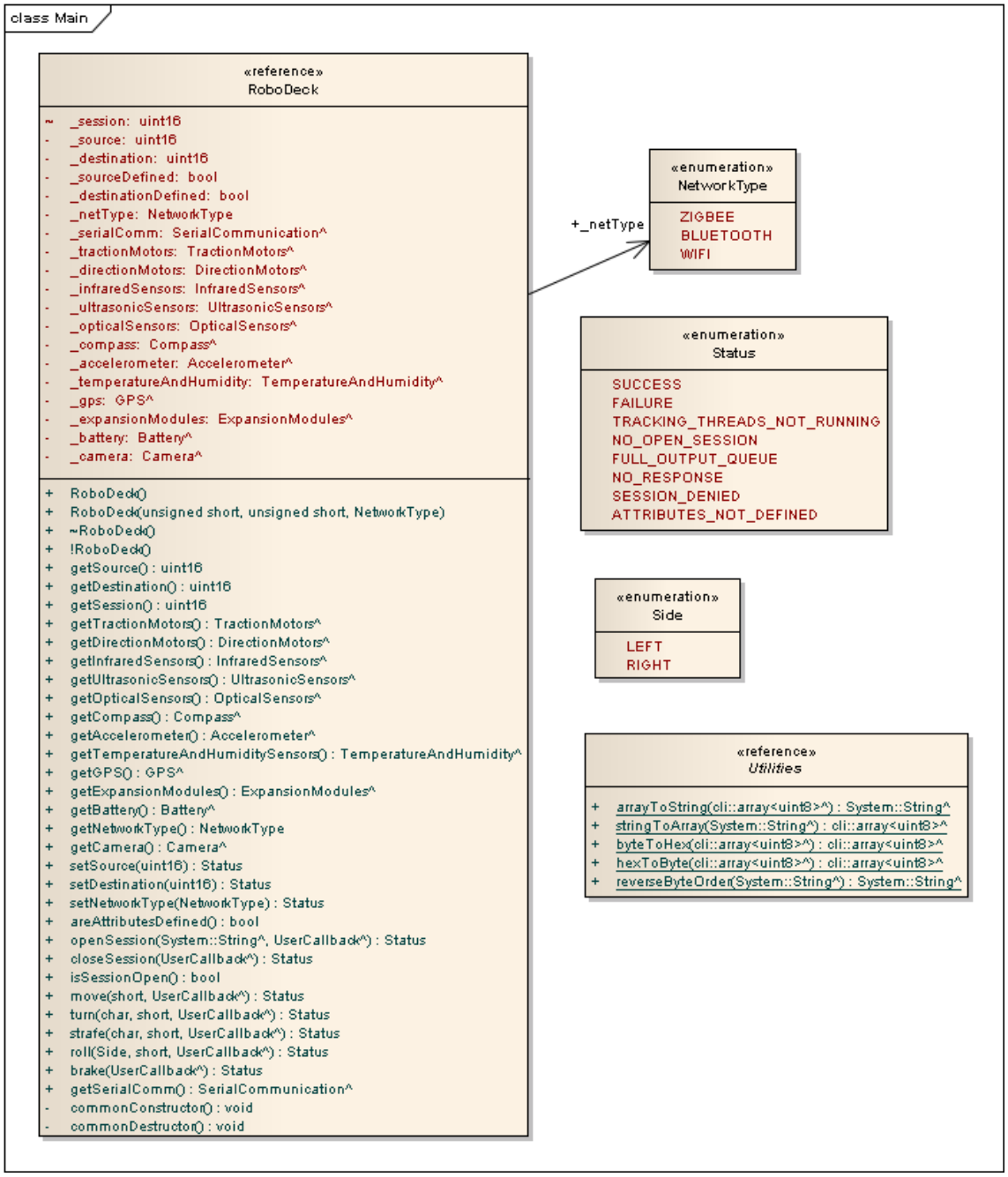

Figura 2. Classe ”RoboDeck”e algumas enumerações

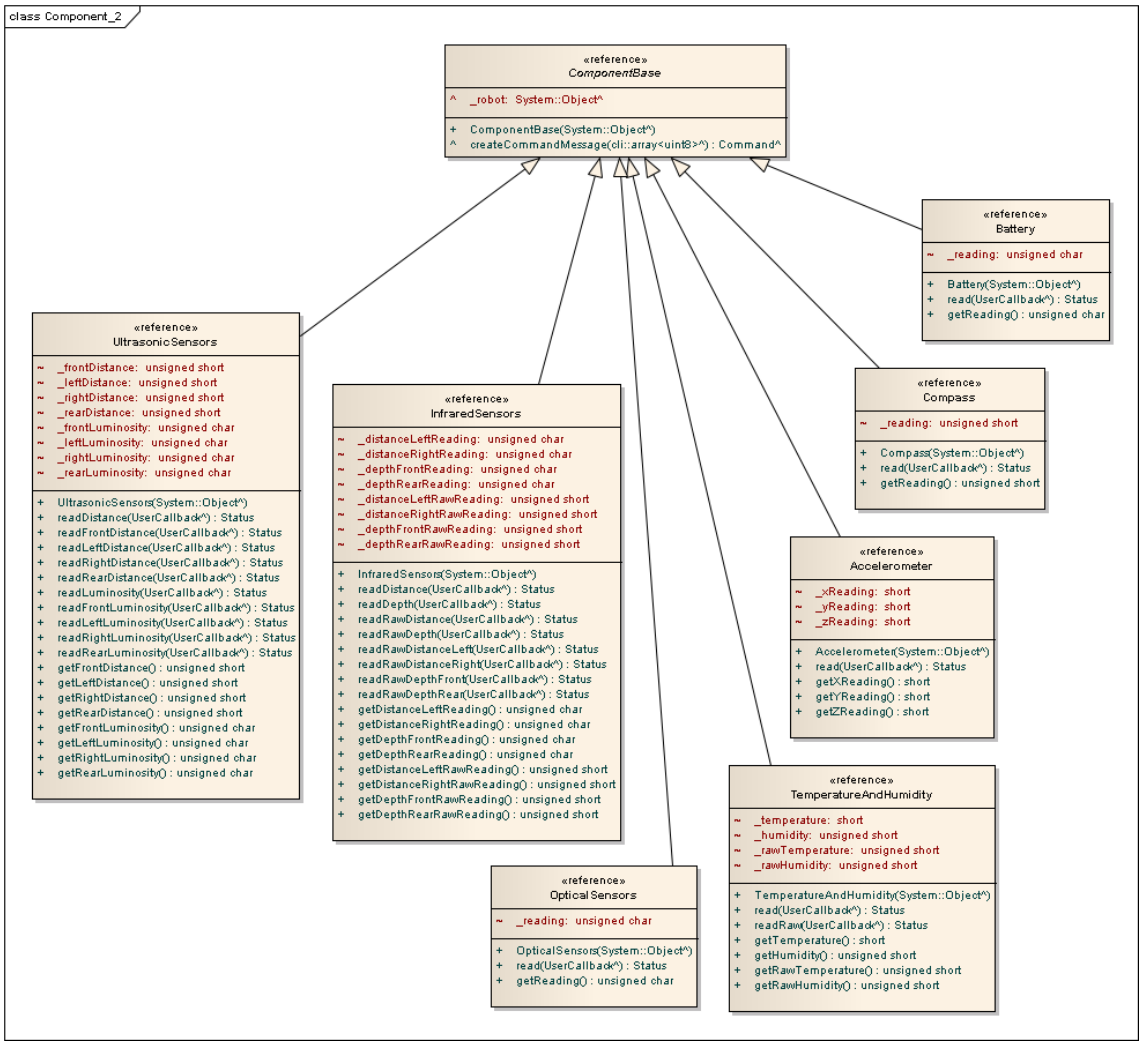

Figura 3. Estrutura das classes componentes

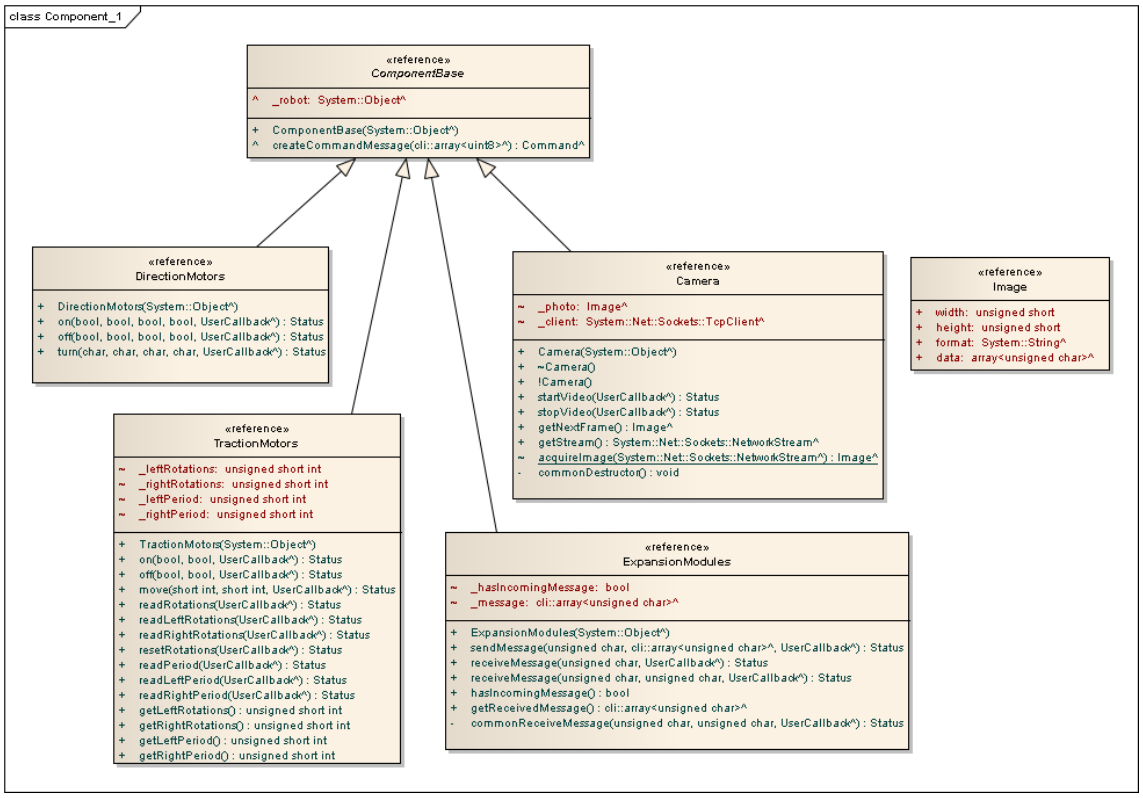

Figura 4. Estrutura das classes componentes

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 será 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 podem ser quebrados em múltiplos comandos pelo SDK. Para estes casos o SDK emprega classes derivadas de *SyncCallback* (*Synchronized Callback*), que agregam objetos de sincronização (*CallbackSynchronizer*) 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 robô.

4. Diagramas de sequência em um exemplo de uso

A seguir é ilustrada a situação onde o usuário deseja ler o valor da bússola. Para tanto, ele utiliza o método *read* de *Compass*. O usuário define também um programa a ser executado no momento da chegada da resposta referente ao comando *read*. As sequências indicadas nos diagramas são resumidas, exibindo apenas os passos mais relevantes.

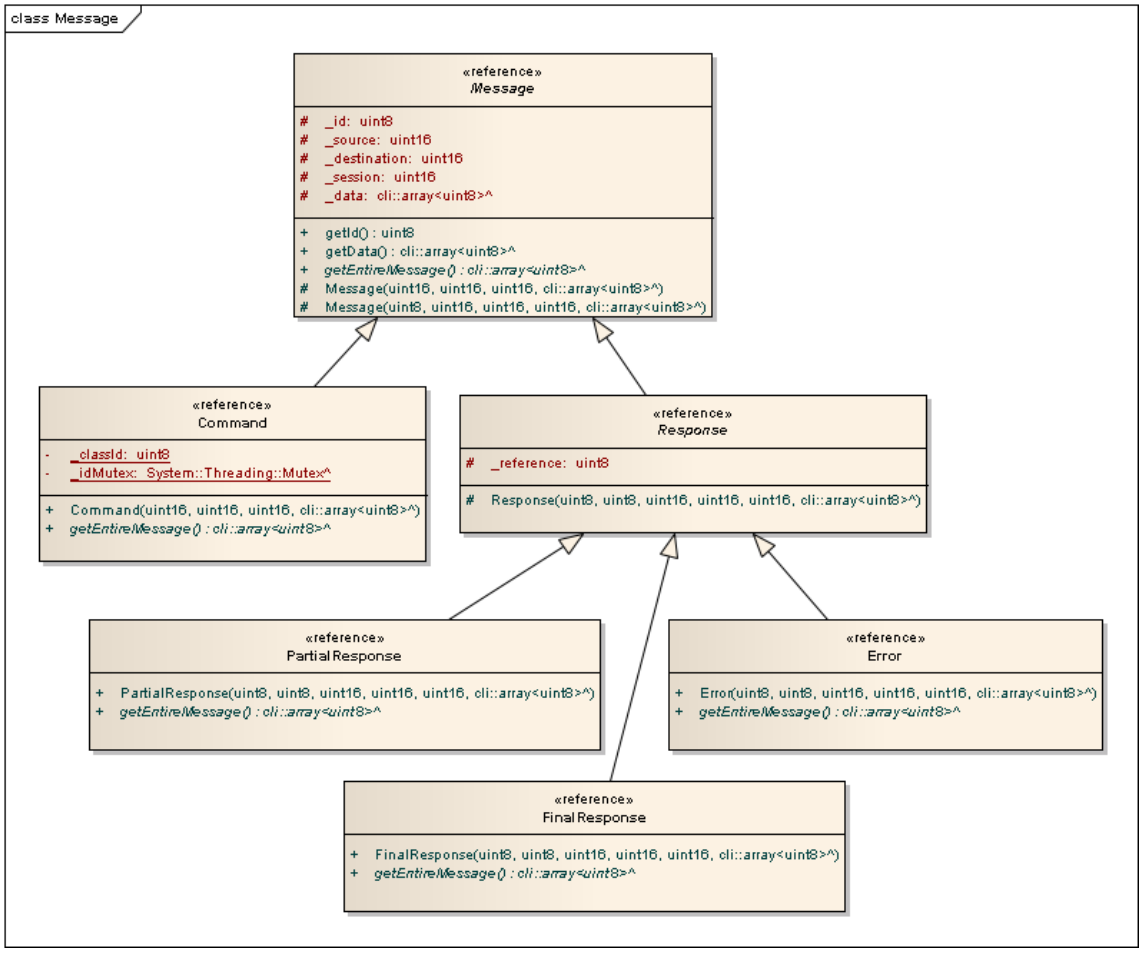

Figura 5. Hierarquia de mensagens

No caso do envio de uma mensagem (figura 6). Primeiramente, o usuário cria (e instancia) uma classe *UserCallback*. Obtém-se então a referência ao componente que representa a bússola (*Compass*) por meio do método *getCompass* da classe "*RoboDeck*". Invoca-se então o método *read*, fornecendo a referência ao *callback* definido pelo usuário. Dentro do método *read*, o SDK cria um objeto de *CompassCallback*, que estende *SDKCallback*. Este é o *callback* intermediário responsável por atualizar o valor da leitura da bússola na classe *Compass*. O método *sendMessage* de *SerialCommunication* é então invocado. Ali dentro, a mensagem de comando é criada. Adiciona-se o respectivo *callback* na tabela *callbackTable* e insere-se o comando na fila de saída (*output*). A execução agora retorna ao programa do usuário, informando o status desta primeira etapa. Este programa fica livre para continuar seu trabalho, inclusive enviar novos comandos ao robô.

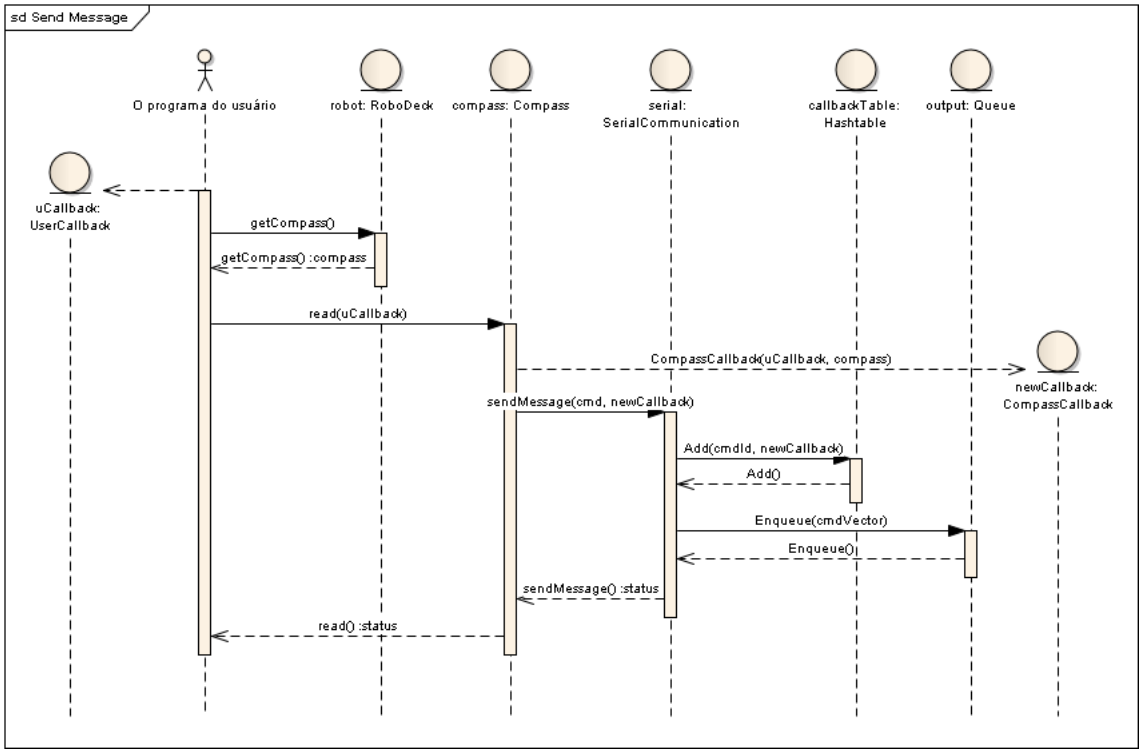

Figura 6. Envio da mensagem

Neste exemplo apresentado, é assumido que a rede ZigBee esteja sendo empregada. A *thread* "*writeLineLoop*" está rodando, monitorando constantemente a fila de saída. Uma vez encontrado o comando na fila, este é empacotado segundo o protocolo ZigBee e de fato enviado através do dispositivo de comunicação serial (*device*). Outra *thread*, "*readLineLoop*", também está rodando, mas está monitorando o dispositivo de comunicação serial. Uma vez recebida à resposta proveniente do robô, esta é desempacotada e adicionada na fila de entrada (*input*) (figura 7).

A thread "*trackInputQueue*" está rodando e monitorando a fila de entrada. Encontrada a resposta proveniente do robô, procura-se por seu campo de referência para determinar a qual comando esta resposta corresponde. Busca-se então na tabela *callbackTable* o devido *callback* previamente armazenado (*newCallback*). Feito isso, inicia-se uma nova *thread* e invoca-se o método *doCallback* de *newCallback*, com a devida resposta. Este

callback intermediário verificará a resposta e, não havendo problemas, atualizará o valor de leitura na classe *Compass*. O *callback* do usuário é finalmente invocado, reportando-se o status de toda a execução. Este último *callback* estará livre para adquirir o valor da leitura da bússola (*getReading*) (figura 8).

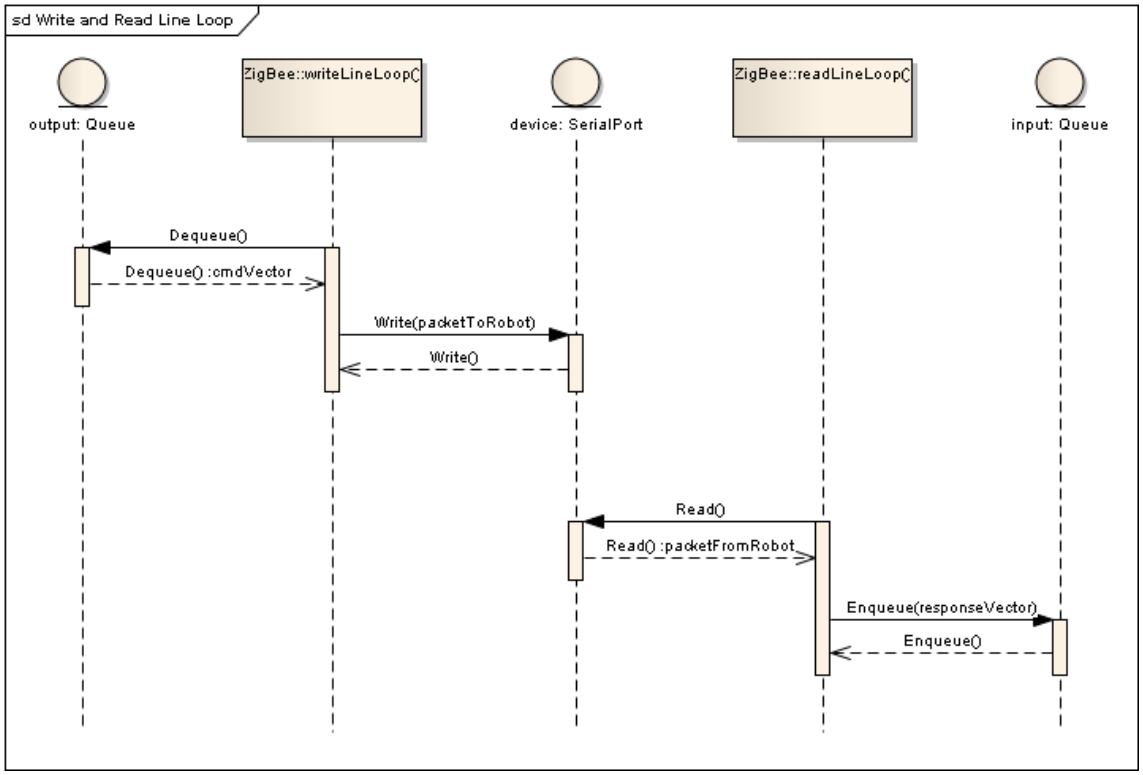

Figura 7. As threads writeLineLoop e readLineLoop

5. Interface da giga de testes

Como comentado anteriormente, baseado no SDK foi criado um aplicativo de giga de testes para, inicialmente, validar o desenvolvimento do próprio SDK, e posteriormente, permitir testar todas as funcionalidades existentes no RoboDeck de forma rápida e prática. Por exemplo, é possível utilizar a giga de teste para verificar problemas com sensores, motores, comunicação entre outros. O aplicativo é constituído por um conjunto de abas, onde cada uma delas possui uma funcionalidade específica. Na figura 9 é apresentada uma imagem da tela principal com as abas na parte superior do aplicativo. Da esquerda para direita é possível verificar a seguinte sequência de abas: Zigbee, WiFi, Sessão, Motores, Movimentação, JoyPad, Sensores (1), Sensores (2), GPS, Expansão, e por fim, Câmera.

Na aba "ZigBee" é possível gerencia a comunicação entre o adaptador UZBee(+) [Uzbee 2017] e o dispositivo de ZigBee conectado ao robô. Primeiramente, deve-se conectar o adaptador UZBee(+) a uma entrada USB do computador ou notebook que está instalado o aplicativo. Depois de conectado, deve-se executar o sistema, e em seguida, acessar a aba ZigBee e escolher a porta serial em que o UZBee(+) foi conectado. No caso da comunicação pelo Wi-Fi é mais simples, basta configurar a comunicação entre o robô e o PC, como se fosse configurar uma rede doméstica comum. Após a configuração do Wi-Fi, basta executar a giga de testes para efetivar a conexão. Estes passos certificam

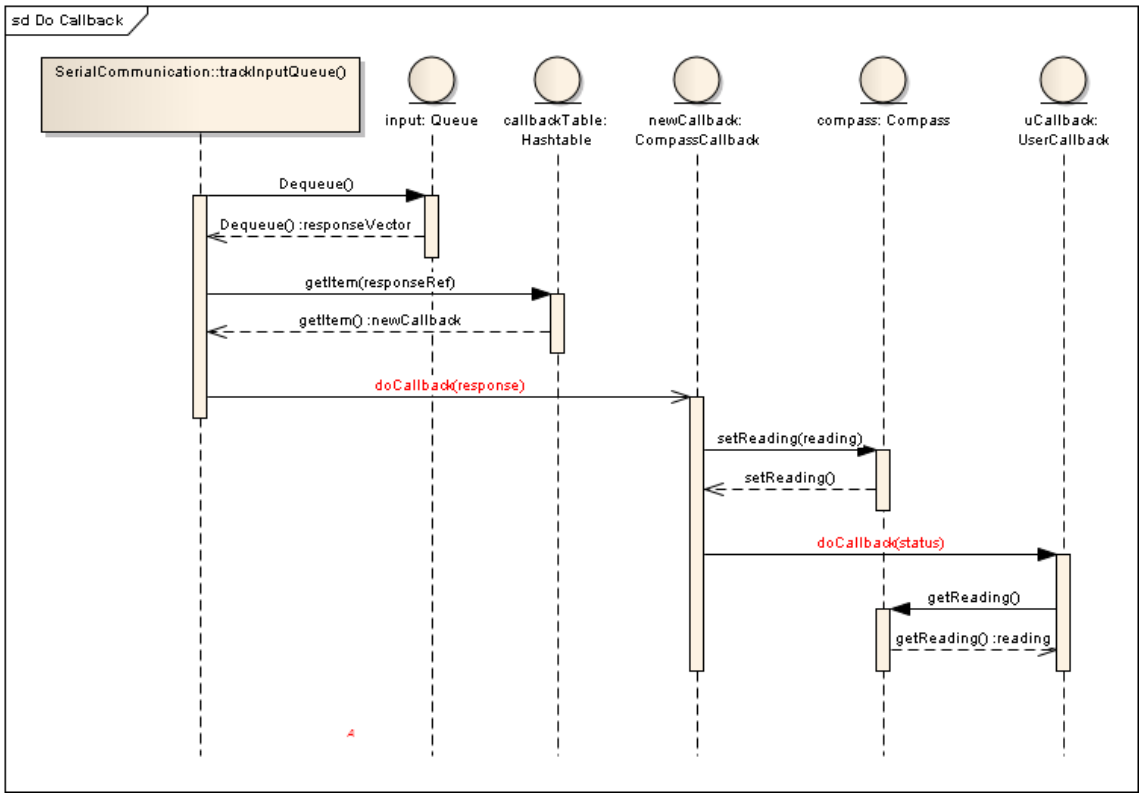

Figura 8. Execução dos *callbacks*

que as redes estão ligadas e prontas para abrir uma sessão. Com a aba "sessão" permite efetivar a transmissão e recepção de dados entre o robô e a giga.

No caso da aba "motores", o mesmo gerencia os controles de acionamento do robô. Com ela é possível acionar independentemente um dos motores de tração ou os dois. É possível controlar os motores de direção (servos motores) acionando o desejado e indicando o grau de movimentação das rodas. Com o *encoder* é possível medir o número de rotação de cada roda e saber seu período, que é o tempo necessário para que um movimento realizado por um corpo volte a se repetir. Essa aba também informa os valores máximos de rotação e de inclinação. Para isso, basta parar o ponteiro do mouse em cima dos comandos de acionamento. Os controles de motores de tração e direção são de extrema importância para identificar possíveis problemas técnicos.

A aba "movimentação" gerencia os controles do robô por meio de coordenadas. Nela simplesmente é informado o ângulo de inclinação das rodas e a sua intensidade. Realizada esta ação, basta escolher o movimento desejado. A aba "joypad" é responsável pelo acionamento imediato do robô. É por meio dela que pode ser testada se a comunicação está enviando e recebendo mensagens em tempo real, conforme os movimentos do robô. Sua configuração é simples, basta informar a intensidade do movimento e o ângulo desejado. É possível controlar os movimentos do robô pelo teclado. Isto facilita os testes para saber se o robô está operando corretamente (figura 10).

Na aba "sensores (1)"é possível medir as distâncias dos sensores ultrassônicos e infravermelhos. A leitura é realizada por meio do objeto mais próximo ao robô. No sensor ultrassônico tem-se a opção de verificar a luminosidade. Quanto mais escuro menor

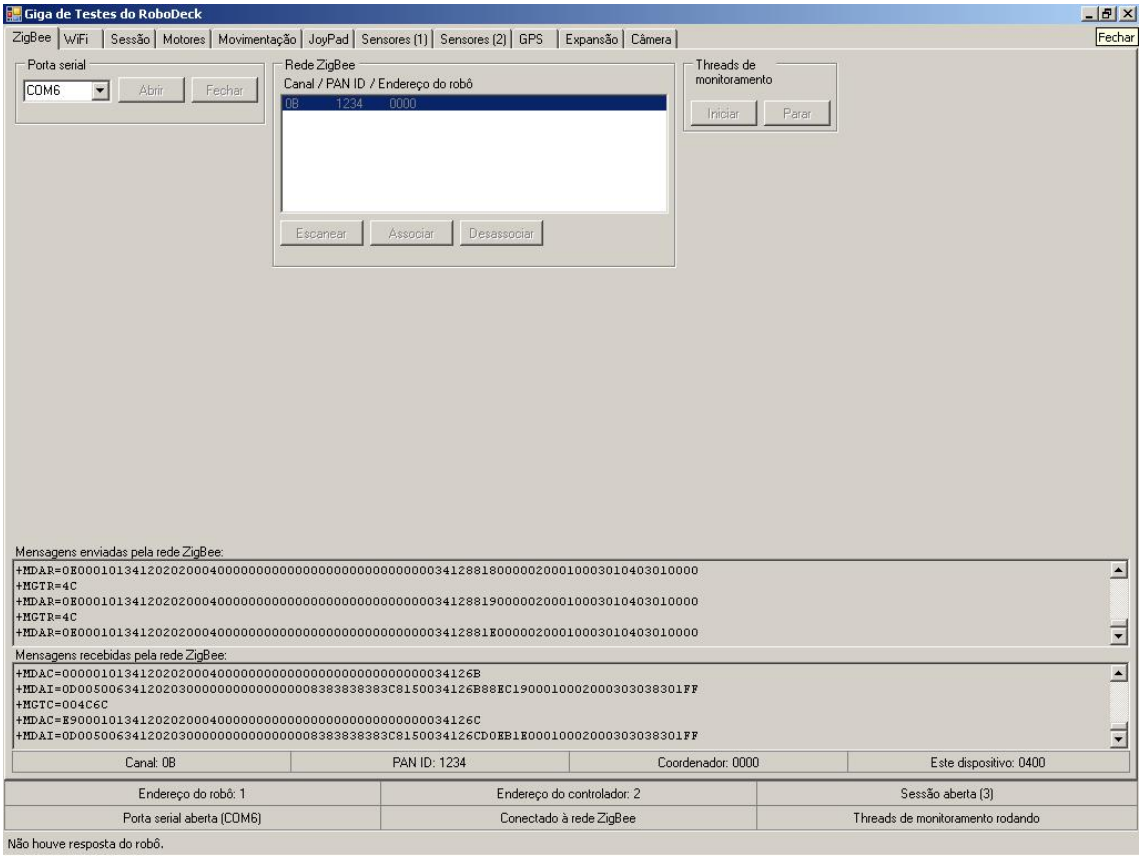

Figura 9. Visão geral da tela da giga de teste

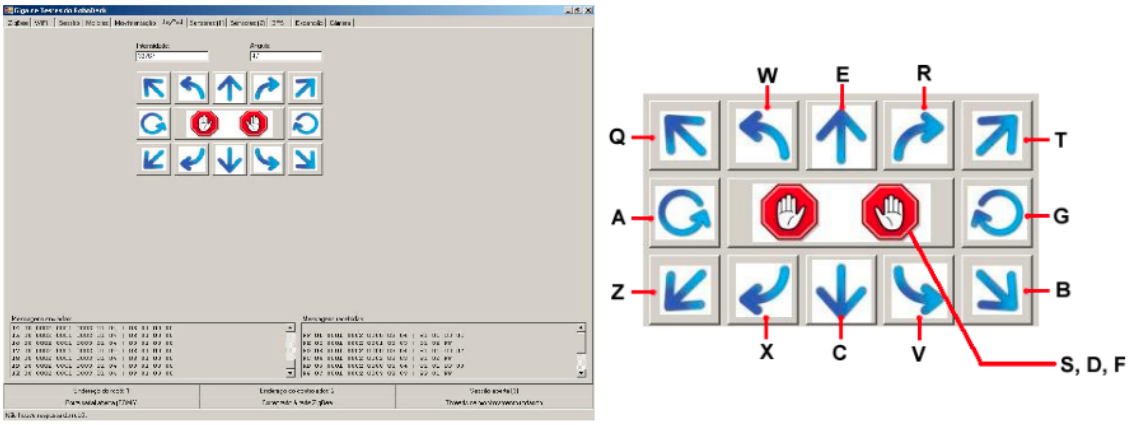

Figura 10. Aba com a opção de movimentação com os detalhes dos comandos pelo teclado

a leitura e quanto mais claro maior a leitura. Isso também vale para os sensores infravermelhos. Na aba "sensores (2)" é possível interagir com os outros sensores como o acelerômetro, sensor de temperatura e umidade, bússola e as baterias.

O robô possui um conjunto de sensores capazes de interagir com o meio, a fim de evitar obstáculos, gerar informações sobre sua posição e sobre seu processo do deslocamento, bem como sobre seu status de funcionamento. É o caso dos quatro sensores ultrassônicos Devantech® SRF08 UltraSonic Ranger que detectam obstáculos entre 3 cm e 600 cm, com um feixe de 45º. Além disso, existem mais quatro entradas disponíveis para possíveis expansões ou acréscimo de novos sensores. Como interface entre os sensores e o microcontrolador, existe um chip atuando como *voltage-translator*. O robô ainda conta com quatro sensores Sharp® GP2D12 IR, que detectam obstáculos entre 4 cm e 30 cm, com um feixe de 5º. Embora não possua nenhum sensor ótico conectado ao robô, existem portas de acesso já preparadas para receber o sensor Fairchild® QRB113x Phototransistor. Para auxiliar no sistema de navegação, o primeiro sensor utilizado é uma bússola eletrônica Devantech® CMPS03. A orientação é dada pelo ângulo formado entre o robô e o norte magnético, que varia de 0ºaté 359,9ºe possui exatidão de 0,1º.

Na aba "GPS" o usuário tem acesso a informações relacionadas à altitude, velocidade, sentido, hora, data, latitude e longitude. Na aba "expansão" são oito *slots* oferecidos para expandir os recursos do robô. Foi desenvolvida a leitura dos dados de transmissão e recepção. Na aba "câmera", tem-se o controle em tempo real de uma webcam. Basta clica sobre o botão "Iniciar Vídeo" e a partir desse momento será possível obter imagens em tempo real. Para fotografar, deve-se apenas configurar o diretório e dar um nome para o arquivo. Logo em seguida deve-se clicar no botão "fotografar".

6. Considerações finais

Durante o desenvolvimento do sistema observou-se grande dificuldade de desacoplar completamente a biblioteca SDK do sistema operacional Windows. Características como o controle de *threads* e o acesso à porta serial são dependentes do sistema operacional. No Windows, há pelo menos três formas de se trabalhar com *threads*: via C *Run-Time Library* (CRT); *Microsoft Foundation Classes* (MFC); e por meio do Microsoft ".NET" *Framework*. No SDK foi adotado esta última forma. O emprego do ".NET" tornou mais simples o desenvolvimento do SDK pois não opera com a linguagem C++ pura, mas com uma extensão definida pela Microsoft como C++/CLI. Essa linguagem surge exatamente para dar suporte ao ".NET". Consequentemente, o SDK emprega a linguagem C++/CLI.

Como trabalho futuro, é possível promover a portabilidade do SDK para outros sistemas operacionais além do Windows. Para isto, é adequado separar as partes dependentes do sistema operacional, e então implementar as partes independentes usando a linguagem C++ pura. O C++/CLI define elementos denominados gerenciados, que são manuseados pelo ambiente de execução do ".NET" 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 o desenvolvimento dos programas. É importante lembrar que os serviços para o *Robotics Studio* são escri-

tos em C#, uma linguagem voltada ao ".NET", 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) pode ser empregada com o mesmo objetivo que a classe *cli::array* (C++/CLI). Avaliando todos estes aspectos, reforçou a decisão de somente utilizar a linguagem C++/CLI ao longo do desenvolvimento de todo o SDK.

7. Agradecimentos

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 RHAE Pesquisador na empresa. Além disso, agradecer a FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) pelo seu programa PIPE para pequenas empresas.

Referências

Robodeck v2.5 é disponibilizado para o mercado! `http://www.xbot.com.br/robodeck-v2-5-e-disponibilizado-para-o-mercado/`. Acesso em: março de 2016.

Uzbee plustm. `http://www.mouser.com/ds/2/158/UZBee\%20Plus\20DS514-2448.pdf`. Acesso em: janeiro de 2017.

Bezerra, T. J. (2012). Arquitetura para integração de módulos de reconhecimento de fala em plataforma robótica móvel" dissertação de mestrado. universidade metodista de piracicaba – unimep.

Kinney, P. et al. (2003). Zigbee technology: Wireless control that simply works. In *Communications design conference*, volume 2, pages 1–7.

Menezes, M. C., Nascimento, C. G. M., de Oliveira, A. C., and Luıs-Brasil, S. (2017). Mapeamento e localizaç ao para o kit robótico robodeck.

Motta, B. d. C. (2015). Aprendizagem por demonstração baseada em redes neurais artificiais aplicada à robótica móvel.

Netto, A. V. and Bertone, O. H. (2018). Desenvolvimento de sistemas computacionais para plataforma robótica aplicada à área de educação. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação*, 1(8).

Pinto, M. A. G. and Netto, A. V. (2018). Desenvolvimento de um driver de controle para robô móvel usando uma biblioteca open source. *Revista Ciência e Tecnologia*, 21(38):39–45.

Pissardini, R. d. S. (2014). *Veículos autônomos de transporte terrestre: proposta de arquitetura de tomada de decisão para navegação autônoma.* PhD thesis, Universidade de São Paulo.

Uzbee (2017). Uzbee plustm.

Valerio Netto, A., Martins, L. E., and Orlandini, G. (2018). Development of api for autonomous navigation of robotic platform. *Revista de Sistemas e Computação-RSC*, 8(1).

Wei, D. C. M. (2015). *Método de desvio de obstáculos aplicado em veículo autônomo.* PhD thesis, Universidade de São Paulo.

Zanolla, L., de Sousa, D. R., Furlaneto, R. M., Botelho, W. T., and Marietto, M. d. G. B. (2017). Implementaç ao com validaç ao real de um controle proporcional, integral e derivativo na plataforma robótica robodeck.