Milton Isidoro

Abstração de uma Plataforma Robótica para Teste e Avaliação de Aplicações Java

Dissertação apresentada para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Informática de Gestão, realizada sob a orientação científica da Mestre Martinha do Rosário Fonseca Piteira e coorientação do Mestre Miguel Jorge Monteiro Roseiro Boavida.

Dissertação em Mestrado de Informática de Gestão

Março 2014

Dedico este trabalho a todos aqueles que acreditaram em mim e no meu trabalho. Em especial aos meus pais e à minha namorada.

“Se vi mais longe, foi porque estava sobre os ombros de gigantes.”

Carta de Isaac Newton à Robert Hooke. 1676.

Agradecimentos

O espaço limitado desta secção de agradecimentos, seguramente, não me permite agradecer, como devia, a todas as pessoas que, ao longo do meu Mestrado em Informática de Gestão me ajudaram, direta ou indiretamente, a cumprir os meus objetivos e a realizar mais esta etapa da minha formação académica. Desta forma, deixo apenas algumas palavras, poucas, mas um sentido e profundo sentimento de reconhecido agradecimento.

À minha orientadora, Mestre Martinha Piteira, agradeço o apoio, a partilha do saber e as valiosas contribuições para o trabalho. Acima de tudo, obrigado por continuar-me a acompanhar nesta jornada e por estimular o meu interesse pelo conhecimento e pela vida académica.

Ao meu coorientador, Mestre Miguel Boavida, agradeço o apoio e orientação disponibilizados na realização deste trabalho, conselhos e sugestões, além das palavras de ânimo que imprimia sempre que achava necessário.

Ao Guilherme Martins da empresa Artica, por me ter disponibilizado os robots utilizados no desenvolvimento do meu trabalho.

Ao meu amigo Luís Nunes, pelas várias horas e noitadas que disponibilizou para me transmitir os seus preciosos conhecimentos, de robótica e afins, que em muito me ajudaram no desenvolvimento do meu projeto.

Aos meus amigos Gil Lopes e Joana Soares Silva, pelas inúmeras horas de conversas e conselhos que em muito contribuíram não só para a conclusão com sucesso desta tese mas também para o meu desenvolvimento como uma pessoa melhor.

Ao meu amigo André Almeida pela sua disponibilidade para traduzir para Inglês o resumo desta tese.

À minha amiga Rubina Latif pela ajuda na leitura e na revisão final de todo o documento.

Ao meu amigo Pedro Gonçalves pelo seu apoio e motivação na fase final desta tese.

À minha namorada Joana Borrega que esteve ao meu lado durante toda esta fase, pelo companheirismo, força e apoio em certos momentos difíceis, pelo incentivo, compreensão e encorajamento, durante todo este período.

Aos pais da minha namorada, prof. Leonor Inácio e prof. João Borrega, por todo o apoio, ajuda e conselhos que me deram. E em especial ao prof. João Borrega que esteve sempre a par de todos os passos que levaram à execução deste trabalho na parte prática, disponibilizando diversos materiais e ajudando a solucionar problemas de elevado grau de dificuldade com o seu conhecimento técnico multifacetado. Agradeço-lhes as muitas horas despendidas comigo, não só em meu benefício e do meu trabalho, como também pela impressão de todas as cópias desta tese.

Por último, tendo total consciência que sozinho nada disto teria sido possível, dirijo um agradecimento especial aos meus pais, por serem modelos de coragem, pelo apoio incondicional, incentivo, amizade e paciência demonstrados e total ajuda na superação dos obstáculos que ao longo desta caminhada foram surgindo. Espero que esta etapa, que agora termino, possa, de alguma forma, retribuir e compensar todo o carinho, apoio e dedicação que, constantemente, me oferecem. A eles, dedico todo este trabalho!

Resumo

A construção de um simples robot envolve conhecimentos em áreas como a eletrónica, mecânica, programação, e a utilização de controlos e sensores. Neste sentido, esta tese tem como propósito verificar a viabilidade do desenvolvimento e teste de uma biblioteca que visa abstrair todos estes conceitos do utilizador final. Pretende-se assim possibilitar a um utilizador, apenas com conhecimentos mínimos de programação ou ainda na fase de aprendizagem de uma linguagem de programação, poder tirar partido da biblioteca desenvolvida e programar o comportamento de um robot com o meio ambiente, sem a necessidade de se introduzir complexidade ao código do programa. Tem-se como objetivo inicial estudar as plataformas micro robóticas existentes, as suas principais características e contextos de utilização assim como identificar as linguagens de programação usualmente utilizadas na comunicação com plataformas robóticas. Propomos então o desenvolvimento de uma biblioteca de abstração para um robot compatível com a plataforma arduíno numa linguagem de programação orientada a objetos, mais concretamente java. Foi também desenvolvida uma aplicação de testes de funcionalidades do robot e um configurador de robots para facilitar a utilização e calibragem dos robots. Como teste da biblioteca resultante, foi desenvolvida uma segunda biblioteca com foco aplicacional no ensino da programação. Esta biblioteca educacional foi construída através da utilização das funções existentes na biblioteca de abstração, e tem como objetivo possibilitar a realização de diversos exercícios de programação desenvolvidos com a colaboração de professores de informática. Após a conclusão da biblioteca de ensino, esta foi utilizada na resolução de todos esses exercícios.

Palavras-chave: Virtualização de Robots, Arduíno, Robots, Bibliotecas, Java, Farrusco, Sensores.

Abstract

In order to construct a simple robot, we need some knowledge in the fields of electronics, mechanics, programming and the use of sensors and controllers. As a result of this, the main goal of this project is to develop and test a library in order to cloak the use of all this concepts from the final user. This way, using this library, it will be possible for someone new in the world of programming or with only the knowledge to write simple code, to program the behavior of a simple robot and his interactions with the world around him without the need to introduce a high level of complexity to the code. The main goal is to study the existing micro robotic platforms, their main characteristics, utilization contexts and to identify the programming languages usually used to communicate between the robotic platforms. So we propose the development of an abstraction library for an arduíno platform compatible robot, in an object oriented programing language, more specifically, Java. There was also developed a robot functionality test application and a robot configurator to help with the use and calibration of the robots. As a test for the resulting library, there was developed a second library focused on programing education. This educational library was built through the use of functions stored in the abstraction library and its main purpose is the execution of a series of programming exercises, developed with the help of informatic teachers. After completion of the educational library, it was used to resolve all of those exercises.

Keywords: Robot Virtualization, Arduino, Robots, Libraries, Java, Farrusco, Sensors.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Acrónimos

DC	*Direct Corrent ou corrente continua*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
LED	*Light Emitting Diode*
FTDI	*Future Technology Devices International*
IDE	*Integrated Development Environment ou Ambiente Integrado de Desenvolvimento*
IR	*Infra Red*
UML	*Unified Modeling Language*
USB	*Universal Serial Bus*
URBI	*Universal Robotic Body Interface*
RL	*Revisão da Literatura*

Capítulo 1
Introdução

O trabalho que nos propomos a fazer encontra-se relacionado com as diversas áreas da robótica.

Assim, o presente capítulo aborda alguns dos aspetos relacionados com a robótica em geral, refere a importância crescente da robótica, referenciando algumas das áreas de atuação, apresentando a importância de se efetuar uma abstração de sistemas robóticos.

Apresentamos igualmente a contextualização do problema e a nossa principal motivação para a resolução do mesmo. Por último serão apresentados os objetivos e hipóteses de investigação, assim como a estrutura do documento.

1.1. Enquadramento

De acordo com o autor Santos, (Santos e colegas, 2013), cada vez mais é necessário realizar tarefas com eficiência e precisão. Algumas destas tarefas necessitam de ser executadas em locais de difícil ou impossível acesso para os humanos, como é o caso do fundo do oceano ou do espaço. Para a execução dessas tarefas com sucesso e sem por em risco a vida humana, existe uma crescente necessidade de criar dispositivos que realizem essas tarefas com segurança. Esses dispositivos são denomidados de robots.

A robótica é a área responsável pelo desenvolvimento de tais dispositivos. Trata-se de uma área multidisciplinar, que tem como objetivo o desenvolvimento e a integração de técnicas e algoritmos para a criação de plataformas robóticas, sendo uma ciência que engloba conhecimentos de engenharia mecânica, engenharia eletrónica, inteligência artificial, entre outras, (Ribeiro, 2006).

Como referenciado por Chaimowicz e Campos, (Chaimowicz e Campos, 1999), os sistemas robotizados são usados nas mais diversas atividades, desde robots que são utilizados no desarmamento de bombas e minas terrestres, robots que realizam a inspeção de cabos telefónicos submarinos, a robots que são usados na educação (Robótica educacional), sendo que, estes sistemas já fazem parte de várias áreas na nossa sociedade.

No caso da robótica educacional, segundo Castilho (Castilho, 2013), esta é caracterizada por ambientes de aprendizagem onde o aluno pode montar e programar um robot ou sistema robotizado. Uma das abordagens que tem vindo a ser utilizada é a utilização dos robots para ensinar conceitos de programação, permitindo de um modo divertido interagir com os robots e aprender programação. Vários

projetos foram desenvolvidos recorrendo ao uso de robots para estimular o interesse dos alunos pela programação, salientando assim 3 exemplos de projetos na área da robótica educacional.

Temos como primeiro exemplo o projeto de mestrado de Gustavsson, (Gustavsson, 2012), que se baseava num trabalho de laboratório com o intuito de inspirar e cultivar o gosto dos estudantes nas áreas da matemática, física, tecnologia e da programação. Esse projeto fazia parte de um projeto mais abrangente "RoboScope" desenvolvido na Alemanha para motivar os estudantes a estudar mais Engenharia. Baseava-se num tema de "trabalho de salvamento" onde os estudantes deveriam construir um robot para desempenhar uma tarefa específica, um salvamento numa central nuclear. O projeto dava liberdade aos estudantes permitindo que estes tomassem as suas próprias decisões, como por exemplo, os alunos poderiam escolher como criar o seu próprio robot, em que a programação era feita na linguagem Java. O autor Gustavsson, (Gustavsson, 2012), chegou à conclusão que ao não limitar as opções dos estudantes e dar-lhes mais liberdade, permitiam aos estudantes usarem as suas próprias experiências, reflexão e ação durante o trabalho de laboratório, atingindo melhores resultados e resultando em uma maior aprendizagem.

Como segundo exemplo, temos o projeto de Garcia e Patterson-McNeill, (Garcia e Patterson-McNeill, 2002), que consistia na implementação de aplicações robóticas onde os estudantes aplicavam conhecimentos de que adquiriam ao longo do seu curso em áreas como engenharia de software, base de dados e várias cadeiras de programação. Os estudantes tiveram a possibilidade de trabalhar com robots LEGO Mindstorms de modo a possibilitar ao estudante ver, tocar e brincar com o resultado final dos seus projetos. Fazia também parte do projeto uma componente de competição no final do semestre, o que contribuiu uma maior motivação por parte dos alunos relativamente aos seus projetos. Em comparação com o projeto de Gustavsson, (Gustavsson, 2012), descrito anteriormente, este projeto também apresentou resultados bastante satisfatórios, resultando um maior interesse dos alunos e maior aprendizagem.

Por fim, temos um exemplo da biblioteca dLife, que segundo Braught (Braught, 2012), é uma biblioteca livre e *open-source* com foco no ensino e pesquisa envolvendo áreas como a robótica, inteligência artificial e sistemas embebidos. O projeto dLife baseia-se num ciclo de código/teste/*debug*, que disponibiliza um acesso direto a informação dos sensores do robot com um sistema de simulação robótica. Suporta vários robots, assim como o robot Finch, Hemisson (ou Khepera Jr.), Sony Aibo, Khepera 2, Khepera 3, e Pioneer 3, encontrando-se mais em desenvolvimento.

Como já vimos, os micro-robots graças ao seu sistema lógico ou informático, são sistemas bastantes versáteis que podem ser reprogramados e utilizados numa grande variedade de áreas de aplicação. Porém, por vezes pretende-se recorrer ao uso de micro robots mas sem os conhecimentos necessários a sua utilização torna-se bastante complexa. Surge então a necessidade de criar mecanismos intermédios que permitam utilizar essas plataformas robóticas sem grande complexidade, á semelhança da biblioteca dLife.

1.2. Motivação/Contextualização do Problema

Como principal motivação que nos leva a apresentar a proposta de abstração de plataformas micro robóticas, é o fato de termos a necessidade de contribuir para que um utilizador com ausência de conhecimentos nas principais áreas presentes na robótica, possa utilizar e programar micro robots sem grandes detalhes técnicos, possibilitando assim a utilização de robots a um público mais abrangente.

Dada essa necessidade, surgiu o conceito de abstração de uma plataforma micro robótica, concretizado através da criação de uma biblioteca que abstraísse para o utilizador final todos os aspetos relacionados com o hardware do robot, permitindo ao utilizador final programar o comportamento de um micro robot com o meio ambiente, sem a necessidade de se introduzir complexidade ao código do programa. Note-se que não é objetivo deste projeto a implementação de algoritmos de inteligência artificial para o robot, e apenas a abstração de toda a componente de hardware do utilizador final.

A biblioteca resultante permitirá o acesso a sistemas robotizados a um maior número de pessoas nas mais diversas áreas de aplicação.

1.3. Objetivos de Investigação / Hipóteses de Investigação

Após o enquadramento e a contextualização do problema foram definidos dois objetivos gerais para o nosso trabalho de investigação:

No primeiro objetivo pretende-se responder às seguintes questões:

1. Quais as plataformas micro robóticas existentes?
2. Qual ou quais as principais características dessas plataformas?
3. Qual ou quais os contextos de utilização dessas plataformas robóticas?
4. Qual ou quais as linguagens de programação que são mais utilizadas na comunicação com as plataformas robóticas?

No segundo objetivo pretende-se responder à seguinte questão:

1. Em que medida é possível criar mecanismos de abstração para uma plataforma robótica?
2. Se for possível, em que medida esses mecanismos facilitam a utilização dessa (s) plataforma (s)?

1.4. Estrutura do documento

O presente documento encontra-se estruturado da seguinte forma:

- **Capítulo 1 – Introdução:** Inicialmente é feita uma introdução e enquadramento do problema, apresentando os objetivos e hipóteses de investigação.

- **Capítulo 2 – Revisão da Literatura:** Neste capítulo é apresentada a revisão da literatura, relativamente às plataformas micro robóticas. São descritas varias plataformas apresentando uma análise comparativa entre elas. Por fim é apresentado algum do trabalho relacionado.

- **Capítulo 3 – Analise de Requisitos:** É descrita uma análise e especificações de requisitos, são identificados os principais requisitos: funcionais, não funcionais, de qualidade, e ambientais que a biblioteca deverá satisfazer. São identificadas as principais funcionalidades da biblioteca, assim como os seus atores e respetivas prioridades. São também apresentados os diagramas de *Use cases* e respetivas descrições.

- **Capitulo 4 – Solução proposta da Biblioteca de abstração:** É feita a apresentação da solução proposta, com uma breve descrição das principais características do robot e a linguagem utilizada na conceção da biblioteca. É descrita a interação do utilizador com a biblioteca. São descritas algumas características mais técnicas do sistema desenvolvido, tal como o seu modelo e arquitetura da framework, é explicado o processamento da comunicação entre o IDE (Integrated Development Environment) do computador do utilizador e o robot. São detalhadamente explicadas todas as funções da biblioteca responsáveis pelo controlo da locomoção básica do robot, controlo das estruturas móveis do robot e o controlo dos seus atuadores sensoriais, assim como é especificado o processo de obtenção de informação sensorial dos seus sensores. É apresentado o modelo de código da biblioteca de abstração, e descritos alguns dos aspetos de controlo do robot no arduíno. São apresentadas as aplicações de interação com o robot que foram desenvolvidas. Por último é apresentada a biblioteca educacional que foi desenvolvida para permitir a avaliação da biblioteca de abstração.

- **Capítulo 5 – Avaliação e discussão de resultados:** é descrita a integração da biblioteca em aplicações java, detalhando passo a passo o processo de criação de um projeto java em dois IDE's distintos que utilizam a biblioteca para interagir com o robot farrusco, descrevendo esses mesmos IDE's. Essencialmente descrevemos a avaliação efetuada e discutimos os resultados.

- **Capítulo 6 – Conclusões e Trabalho futuro:** Neste capítulo podemos identificar as diversas conclusões a que chegámos com a conclusão do trabalho proposto. Está também presente neste capítulo as propostas de trabalho futuro, que poderá ser realizado e que poderá definir uma evolução positiva de todo o projeto e que poderá satisfazer algumas possíveis e previsíveis necessidades.

- **Capítulo 7 – Bibliografia**

- **Anexos**

Capítulo 2
Revisão da Literatura

1.5. Introdução

No presente capitulo é efetuada a Revisão da Literatura (RL) relativa as plataformas micro robóticas, como também os contextos de utilização dessas plataformas. Apresentamos uma análise comparativa entre as diferentes plataformas baseadas num conjunto predefinido de dimensões.

Por último apresentámos algum do trabalho relacionado.

2.1. Plataformas micro robóticas

2.1.1. *Análise histórica do surgimento dos micro robots*

Como referido por Castilho (Castilho, 2013) e Souza (Souza, 2005), a palavra robot foi usada pela primeira vez, em 1921, numa peça de teatro que tinha como título R.U.R (*Russum´s Universal Robots)*, na Tchecoslováquia, escrita por Karel Čapeks. Aparentemente a palavra descende de ROBOTA, que em checo significa trabalho servil ou trabalho pesado e foi usado Robot no sentido de uma máquina substituir o trabalho humano.

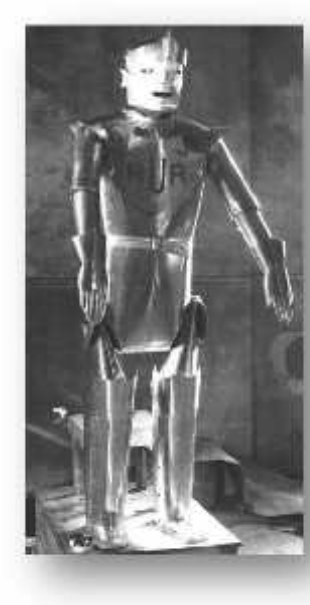


Figura 2.1-1 R.U.R. Um robot da peça de de Karel Čapeks "*Rossum´s Universal Robots"*.

No fundo, a palavra robot surgiu do que seria a ideia atual de androide, que por sua vez é o que a grande maioria das pessoas pensa quando falamos em robots. Uma definição mais realista, do que um escravo autômato, seria a do Instituto de Robótica Americano (Robot Instituto of América), 1979:

> *"Robot é um manipulador reprogramável e multifuncional projetado para mover materiais, partes, ferramentas ou dispositivos especializados através de movimentos variáveis programados para desempenhar uma infinidade de tarefas." (Ullrich., 1987).*

Aqui, as palavras-chaves são multifuncionais e reprogramáveis. Diferentemente da automação convencional, os robots são projetados para realizarem, dentro dos limites especificados, um número restrito de diferentes tarefas.[1]

> "Um robot é uma máquina projectada para imitar algumas acções humanas. Não precisa se parecer com o ser humano, mas tem que executar as tarefas automaticamente." (Souza, 2005)

(Brandão e Vaz, 1999), apresentam algumas definições de um robot:

- *"Robot é uma máquina que tem aspeto, humano e é capaz de agir de um modo automático para desempenhar uma determinada função."*
- *"O robot é um aparelho capaz de substituir o homem nas suas funções motoras, sensoriais e intelectuais."*
- *"O robot é um braço mecânico que possui vários eixos de deslocamento e que é capaz de reproduzir diversos movimentos humanos graças ao recurso a um computador programado com essa finalidade."*
- *"O robot é um engenho que pode ser programado, destinado a manipular peças ou utensílios, e cujos movimentos são determinados com vista a completar tarefas específicas."*
- *"O robot é um manipulador de múltiplas funções e que é programável."*
- *"O robot é um utensílio de trabalho capaz de perceber o ambiente no qual se movimenta e se adapta a ele, a fim de realizar de um modo autónomo manutenções pré-determinadas."*

Várias são as definições de um robot, estando todas elas em parte corretas. Essencialmente um robot é composto por uma parte mecânica e outra eletrónica. A parte mecânica composta por uma base fixa ou móvel, por braços e garras possuindo, geralmente diversos graus de liberdade de movimentos (translação ou rotação). A parte eletrónica é constituída por uma unidade de processamento central constituinte de uma memória interna (cérebro do robot). Esta unidade central pode encontrar-se ligada a motores, micro interruptores e sensores (de proximidade infravermelhos, temperatura, contacto, cheiros, sons, peso, pressões, deteção e reconhecimento de imagens, entre outros).

---

[1] Fontes: http://www.eletronlivre.com.br/devbot/doc/introducao-robotica.globalcode e http://www.pucrs.br/eventos/desafio/mariaines.php

Das diferentes definições de robot analisadas na literatura consideramos que aquela que mais se aproxima ao nosso conceito de robot é a última definição, pelo que no âmbito desta tese será a que iremos utilizar.

De acordo com Rosheim, (Rosheim, 2006), *"Os primeiros robots podem ter sido criados pelo engenheiro grego Ctesibius (c. 270 aC). Ctesibius aplicou conhecimento de pneumática e hidráulica para desenvolver o primeiro órgão e relógios de água com figuras em movimento ".*

Também Leonardo Da Vinci, aproximadamente em 1495, desenhou um esboço de um robot humanoide. O robot foi projetado com capacidade de se sentar, mover os braços, cabeça e mandibula, (Rosheim, 2006).

Rosheim (Rosheim, 2006) refere que Leonardo Da Vinci desenvolveu uma extensiva investigação no domínio da anatomia humana que permitiu o alargamento de conhecimentos para a criação de articulações mecânicas. Como resultado deste estudo desenvolvido, surgiram diversos exemplares de bonecos que moviam as mãos, os olhos e as pernas, e que conseguiam realizar ações simples como escrever ou tocar alguns instrumentos.

Segundo (Dowling, 1996), o primeiro robot industrial foi o Unimates, desenvolvido por George Devol e Joe Engleberger, no final da década de 50, início da década de 60. As primeiras patentes de máquinas transportadoras pertenceram a Devol, máquinas essas que eram robots primitivos que removiam objetos de um local para outro. Engleberger, por sua vez, pela construção do primeiro robot comercial foi apelidado de "pai da robótica". Outro dos primeiros computadores foi o modelo experimental chamado Shakey, desenhado para pesquisas em Standford, no final da década de 60.

Os robots podem ser equipados para sentir ou perceber calor, pressão, impulsos elétricos e objetos, e podem ser usados juntamente com sistemas de visão rudimentar. Desta forma é possível monitorizar o trabalho realizado. Ou seja, o robot moderno pode aprender e lembrar-se das tarefas, reagir ao seu ambiente de trabalho, operar outras máquinas, e comunicar com o pessoal da fábrica quando há ocorrência de mau funcionamento. Isto representa uma nova tecnologia – que pode levar à reformulação de nossa maneira de pensar e trabalhar. (Ullrich, 1987, p.8)

Conforme referido por Gustavsson (Gustavsson, 2012), novos campos de aplicação em robótica estão em constante desenvolvimento, como por exemplo, na central nuclear de Fukushima (Figura 2.1-2), após o acidente nuclear no início de 2011. Os Robots foram utilizados para inspecionar a extensão dos danos e essenciais nos processos de limpeza. Robots em hospitais médicos, de saúde e robots de serviço, robots militares são apenas alguns dos novos robots que os pesquisadores utilizam.

Figura 2.1-2 Robot de Fukushima, projetado para a limpeza da central nuclear

Os robots têm vindo gradualmente a evoluir, estes são cada vez mais usados nas mais diversas áreas. Por vezes os robots são utilizados para substituir os humanos em trabalhos considerados de risco, outras para tornar os processos mais rápidos, pois com o robot certo este pode realizar uma tarefa com uma maior precisão e rapidez do que um humano o faria. Segundo Gustavsson (Gustavsson, 2012), nem todos os projetos de robótica atuais são considerados benéficos para a nossa sociedade.

Apesar do gradual crescimento dos robots e dos benefícios identificados na sua utilização existem alguns autores que consideram que a sua utlização pode não ser benéfica. Por exemplo, existe um projeto no japão em que os robots são desenvolvidos para fazer companhia a pessoas de idade. Estes robots são criados com o objetivo de substituir os humanos em situações sociais. Neste caso o autor levanta varias questões sociais, na medida em que demonstra que os robots são um item comum para a nossa sociedade, mas que temos de discutir sobre como devemos e podemos usá-los.

Seguidamente faremos a descrição dos micro-robots.

2.1.2. *LEGO Mindstorm NXT*

Figura 2.1-3 Lego Mindstorm NXT

Como referenciado no site oficial da loja Lego[2], LEGO Mindstorms NXT é uma linha do brinquedo LEGO, lançada comercialmente em 2006, voltada para a Educação tecnológica. Consistem em um robot construído com o módulo NXT e sensores. Vem equipado com um processador, software próprio e sensores de luz, de toque e de som, permitindo a criação, programação e montagem de robots com noções de distância, capazes de reagir a movimentos, ruídos e cores, e de executar movimentos com razoável grau de precisão.

Este robot tem como características um processador Atmel 32-bit ARM [3]; Três portas de saída digital; Quatro portas de entrada (uma IEC 61158, tipo 4); Display tipo matriz; Alto-falante; Bateria recarregável de lítio; Bluetooth; Porta de Comunicação USB (*Universal Serial Bus*) 2.0; Três motores servo [4] interativos (com encoder acoplado); Quatro sensores: ultra-som, som, luz, cor e contato; Programa de computador com uma versão LEGO do LabVIEW.

Como linguagens de programação utiliza: NTX-G e LabView (programação gráfica drag and drop), Java, Lua e RobotC.

---

[2] ***Fontes***: *http://shop.lego.com/en-PT/LEGO-MINDSTORMS-EV3-31313?fromListing=listing~ e http://www.robotshop.com/en/lego-mindstorms-nxt-20-english.html*

[3] **Processador Atmel 32bit ARM** - *http://www.atmel.com/products/microcontrollers/arm/default.aspx?utm_source=Blog&utm_medium=Social%2BMedia&utm_campaign=CES*

[4] **Motores Servo:** Segundo Mott (MOTT, 1999), McManis (CHUCK MCMANIS, 2006) e Braga (BRAGA, 2002), motores servo são dispositivos de malha fechada, ou seja: Recebem um sinal de controlo, verificam a atual posição e atuam no sistema movendo para a posição desejada. Ao contrário dos motores contínuos que rodam indefinidamente o eixo dos servo motores possui a liberdade de apenas cerca de 180º graus, mas são precisos quanto a posição. Fonte: http://robolivre.org/conteudo/servomotor

2.1.3. *Finch*

O Finch é um robot desenhado para a educação e o ensino de ciências da computação. Segundo Tom Lauwers e Illah Nourbakhsh este robot tem um design inovador resultado de um estudo de quatro anos conduzido no Carnegie Mellon's CREATE lab.

Figura 2.1-4 Robot Finch

De acordo com informação disponibilizada no site oficial do Finchrobot[5], este robot suporta mais de uma dúzia de linguagens e ambientes de programação, incluindo diversos ambientes apropriados para alunos desde os oito anos de idade. Este robot foi projetado para permitir aos alunos escrever programas interativos.

Este robot utiliza uma placa com um chip tb6552 [6] e um microcontrolador Atmega 16/32U8 [7]. Como características deste robot, este possui sensores de luz, temperatura e obstáculo, um acelerômetro, dois motores de rotação continua [8](DC), um buzzer[9], um LED ***(Light Emitting Diode)*** RGB[10], vem equipado com um suporte para uma caneta (para permitir desenhar) e possui uma entrada para porta USB (não necessita de bateria, pois funciona com a alimentação do USB).

Como linguagens de programação, este robot oferece suporte num conjunto variado de linguagens, como: Finch Dreams, Python/Jython, Processing, Snap, Scratch, National Instruments LabVIEW, Calico, Java, Javascript, Greenfoot, Scala, C++, C#, RoboRealm, Matlab e Visual Basic.

---

[5] ***Fonte***: http://www.finchrobot.com/

[6] **Chip Tb6552** - *http://www.alldatasheet.com/view.jsp?Searchword=TB6552*

[7] **Microcontrolador Atmega 16/32U8** - *http://www.atmel.com/devices/atmega328.aspx*

[8] **Motores de Rotação Continua:** são motores que precisam de uma fonte de corrente contínua, ou de um dispositivo que converta a corrente alternada comum em contínua. Podem funcionar com velocidade ajustável entre amplos limites e se prestam a controles de grande flexibilidade e precisão. Estes motores rodam continuamente com um angulo completo de 360º. Fonte: http://www.coladaweb.com/fisica/eletricidade/motores-de-corrente-continua

[9] **Buzzer –** Um buzzer ou um beeper é um dispositivo para emissão de áudio. É um aparelho de sinalização elétrica, como uma campainha, que faz um som vibrante. Fonte: http://www.thefreedictionary.com/buzzer

[10] **LED RBG** – Um Led tem como funcionalidade básica a emissão de luz. Com o LED RGB pode-se acender as três cores primárias: Vermelho (**R**ed), Verde (**G**reen) e Azul (**B**lue). Fonte: https://www.sparkfun.com/products/105

2.1.4. *O robot Farrusco - Arduíno*

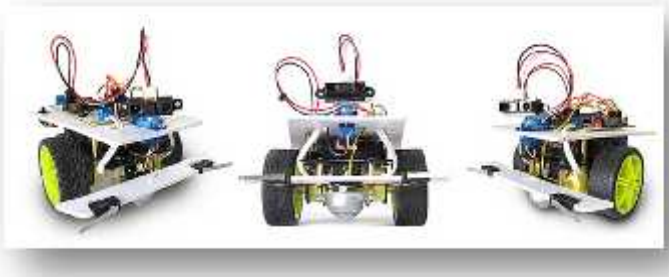


Figura 2.1-5 Robot Farrusco

Segundo o site oficial dos criadores do robot Farrusco[11], é um pequeno robot com base na plataforma Arduíno, utiliza o Motoruino que é um derivado do Arduíno mas com algumas modificações, como a possibilidade de controlar até 2 motores de rotação contínua, fichas para ligar motores servo e sensores diretamente.

Este robot utiliza uma placa original motoruino com um chip L293D e um microcontrolador Atmega 168/328[12]. Como características possui uma placa Motoruino (compatível com Arduíno) com um chip L293D [13] e um microcontrolador Atmega 168/328, dois Motores de corrente contínua (DC), um Motor Servo, um Led RGB, sensores (de Luz, de colisão Infravermelhos e de Som), Bumpers de colisão, um Buzzer, um Bluetooth e funciona com uma bateria (ou pilhas).

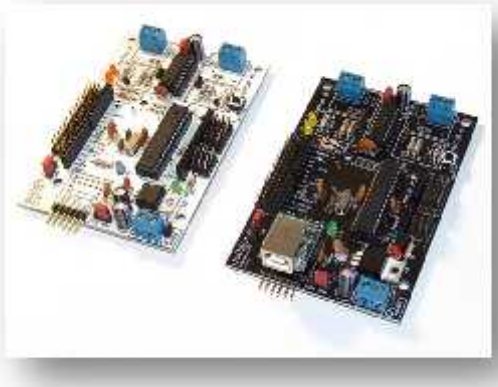


Figura 2.1-6 Placa Motoruino

SOBRE O MOTORUINO

A placa Motoruino [14] foi desenhada especificamente para controlar motores de corrente contínua (DC), motores servos, e sensores de diversos tipos. Foram adicionados filas de 3 pins para facilitar a ligação dos motores servos e dos sensores. Com a ponte-H L293D é possível ligar 2 motores de corrente contínua com 600 miliamperes por canal.

O Motoruino é 100% compatível com as placas originais Arduíno Diecimilla / Duemilanove e respetivos shields, com a exceção de que não tem uma entrada USB normal. É necessário um conversor TTL/USB.

Este robot, sendo um robot compatível com placa arduíno, utiliza como linguagem de programação a linguagem padrão do Arduino, Processing.

---

[11] ***Fontes***: http://www.guibot.com/robotFarrusco e *http://www.guibot.pt/farrusco/*

[12] **Microcontrolador Atmega 168/328** - *http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet.pdf*

[13] **Chip L293D** - *http://arduino.cc/documents/datasheets/L293D.pdf*

[14] ***Fonte***: *http://www.guibot.pt/motoruino/*

2.1.5. *Robot Hemisson*

Figura 2.1-7 Robot Hemisson

Como referido no site oficial do robot Hemisson[15], este robot foi projetado para o ensino e educação. Vem equipado com vários sensores e um processador MCU de 8 bits programável, o robot é capaz de evitar obstáculos, detetar a intensidade da luz ambiente e seguir uma linha no chão.

Como características este robot contem um processador PIC16F877 [16], dois Motores de corrente contínua (DC), oito sensores de luz ambiente (infravermelhos), seis sensores de obstáculos (infravermelhos), dois sensores de deteção de linhas, um buzzer, quatro leds e quatro botões programáveis, possui uma bateria de 9v e para comunicação utiliza uma porta serial com conector DB9[17] e um recetor de TV remoto.

Como linguagem de programação, este robot utiliza C/C++, Matlab para se comunicar.

---

[15] ***Fontes***: *http://www.k-team.com/mobile-robotics-products/hemisson e http://www.k-team.com/mobile-robotics-products/hemisson/specifications e • http://www.k-team.com/mobile-robotics-products/hemisson/introduction e • http://www.k-team.com/mobile-robotics-products/hemisson/applications-for-hemisson.*

[16] **Processador PIC16F877** - http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010241

[17] **Conector DB9 -** *http://www.l-com.com/customer-service?ID=4878*

2.1.6. *Comparação das características dos robots*

Tabela 2.1.6-1 Tabela comparativa dos Robots: Mindstorm, Farrusco, Finch e Hemisson.

Características	Mindstorm	Farrusco	Finch	Hemisson
Placa	Proprietária - Modulo NXT	Motoruino (Compatível com Arduíno - Open Source)	Proprietária	Proprietária
Chip		L293D	tb6552	
Microcontrolador		Atmega 168/328	Atmega 16/32U8	PIC16F877
Motor DC		2	2	2
Motor Servo	3	1		
Sensor de colisão IR		1	3	6
Sensor de Ultra-som	X			
Sensor de Luz	X	1	2	8 (infravermelhos)
Sensor de Temperatura			1	
Sensor de Cor	X			X
Acelerómetros			X	
Buzzer	1	1	1	4
LED	1	1 (RGB)	1	4
Suporte de Caneta			X	
Bumpers de colisão (Contacto)	2	2		
Botões programáveis				4
Comunicação	Bluetooth e USB	Bluetooth e USB	USB	DB9 e recetor TV
Bateria / Pilhas	X	X		X
Linguagens	NTX-G e LabView (programação gráfica drag and drop), Java, Lua, RobotC	Processing	Finch Dreams, Python/Jython, Processing, Snap, Scratch, National Instruments LabVIEW, Calico, Java, Javascript, Greenfoot, Scala, C++, C#, RoboRealm, Matlab, Visual Basic	C/C++, Matlab

2.2. Plataforma Arduíno

Figura 2.2-1 Logotipo Arduino

Segundo Souza (Souza e colegas, 2011), Arduíno é uma plataforma de harware *open-source* de fácil utilização que se baseia numa placa com um microcontrolador simples, que disponibiliza um ambiente de desenvolvimento para a criação de software para a placa. Esta plataforma é ideal para a criação de dispositivos que permitam a interação com o ambiente, essa interação é possível através da utilização de sensores de temperatura, luz som, etc., para a recolha de dados e leds, motores, displays, buzzers etc., para interagir com o ambiente.

Os projetos do Arduíno [18] podem ser *stand-alone*, ou podem comunicar com outros softwares (por exemplo, *Flash*, *Processing*, *MaxMSP*) As placas podem ser montadas à mão ou comprados pré-montados e o IDE *open-source* pode ser adquirido gratuitamente através de download.

A linguagem de programação Arduino é a linguagem *Processing*, baseada na linguagem C/C++, a qual é também open-source. Através do *bootloader* dispensa-se o uso de programadores para o chip facilitando ainda mais o seu uso uma vez que não exige compiladores ou *hardware* adicional. Neste ambiente de desenvolvimento, são disponibilizadas bibliotecas que permitem a interface com outros *hardwares*, permitindo o desenvolvimento de aplicações simples ou complexas em qualquer área.

2.3. Trabalho relacionado

Considerando as características do sistema proposto, analisamos um conjunto de aplicações que melhor se caracterizam com o problema e que seguidamente passamos a descrever.

2.3.1. *O ProjetoTekkotsu*

Segundo informação disponível no site oficial do projeto Tekkotsu[19], este é um Software que pretende oferecer uma estrutura onde se pode construir e lidar com rotinas, para que o utilizador possa se concentrar apenas na programação de alto nível.

Tekkotsu é baseado numa programação orientada a objetos e uma arquitetura de *event-passing*, ao fazer pleno uso do modelo e recursos de herança do C++. Esta linguagem foi originalmente desenvolvida para interação com o robot Sony AIBO, mas desde então tem sido expandida para

[18] ***Fontes***: *http://arduino.cc/en/Guide/Introduction* e *http://arduino.cc/en/Main/FAQ*

[19] ***Fontes***: http://tekkotsu.org/ e http://www.tekkotsu.org/about.html

funcionar numa variedade de robots.

Como o projeto Tekkotsu está desenvolvido na linguagem padrão C++, não é necesssário proceder-se à aprendizagem de uma nova linguagem de programação, não existindo também uma distinção muito nítida entre o "alto nível" e "baixo nível". Neste projeto pode-se começar logo a interagir com a aplicação para personalizar as funcionalidades sendo um projeto criado como um meio para lidar com problemas avançados de investigação.

Alguns dos serviços Tekkotsu incluem: processamento visual, localização, ferramentas e controlo de movimento em tempo real.

A Tekkotsu é um projeto *open-source*, e baseia-se em diversas bibliotecas não propriertárias. Exemplos dessas bibliotecas são a *NEWMAT (matriz), libjpeg E libpng, libxml2, e zlib.*

2.3.2. *URBI (UNIVERSAL REAL-TIME BEHAVIOR INTERFACE)*

Segundo Baillie, (Baillie, 2005), hoje em dia cada robot vem com a sua própria linguagem de programação, impondo aos programadores a reaprenderem o que já sabem. Para além disso, algumas linguagens são notavelmente difíceis de dominar. Foi entao criada uma interface, a URBI [20] (U*niversal Robotic Body Interface*) com o intuito de fornecer a comunidade de programadores um Software *open-source* padrão para o controlo e o desenvolvimento da interface de robots que seja tão potente como simples de usar.

Esta interface é utilizada para desenvolver aplicações móveis para robots e inteligência artificial. A linguagem utilizada é composta por *scripts*, orientada a eventos e baseia-se numa arquitetura de processamento paralelo. A falta de padrões em robótica faz da URBI uma plataforma particularmente sensível às diferentes interfaces de programação de aplicativos (APIs) entre os robots, componentes e assim por diante.

Como indica Baillie, (Baillie e colegas, 2008), o principal objetivo da URBI é de tornar-se o Software padrão para o controlo de “baixo nível” robótico. Baillie, (Baillie e colegas, 2008), afirma também que as aplicações práticas e a utilização da versão atual da URBI têm demonstrado a validade da abordagem, quer em termos de desempenho quer em facilidade de uso.

[20] ***Fonte***: http://www.conscious-robots.com/en/reviews/robots/urbi.-universal-real-time-behavior-interface.html

2.3.3. *dLife: Biblioteca Java para Multiplataforma de Robótica*

A biblioteca dLife é uma biblioteca java *open-source* desenvolvida para ser utilizada em disciplinas de informática no ensino superior e a sua utilização prevê conceitos de robótica, inteligência artificial, aprendizagem máquina e visão computacional. A conceção da dLife aborda diversas preocupações levantadas por relatos experientes na literatura educacional de CS (*Computer Sience* / informática), incluindo um ciclo reduzido de código/teste/debug e um acesso célere a informações de sensores robóticos e uma integração com um sistema de simulação robótico. A biblioteca dLife fornece um suporte completo aos seguintes robots: Finch, Hemisson (ou Khepera Jr.), Sony Aibo, Khepera 2, Khepera 3, and Pioneer 3, com mais em desenvolvimento, (Braught, 2012).

Segundo Braught (Braught, 2012), o uso da robótica no ensino da informática tem recebido um aumento gradual de atenção durante a última década. Com o passar do tempo, o número, variedade, capacidades e o preço do *hardware* robótico utilizado no ensino e pesquisas ampliou consideravelmente. Á medida que as opções do *hardware* robótico aumentaram, a escolha de linguagens de programação, bibliotecas e aplicações também aumentaram. A diversidade de opções fez com que se torna-se mais fácil para os docentes encontrarem o *hardware* robótico correto para a plataforma, com a linguagem de programação preferida e aplicação apropriada para o público-alvo.

De acordo com o autor, dLife é uma das pequenas (mas em constante crescimento) bibliotecas que estão concebidas para suportar a programação de determinadas plataformas de *hardware* robótico utilizando a mesma linguagem e aplicação através dos múltiplos níveis de currículo de informatica.

A primeira biblioteca produzida foi a Pyro (Python robotics). Atualmente existem várias bibliotecas que permitem uma diversidade de plataformas de hardware robótico para ser programado com o mesmo ambiente, linguagem e aplicação. Algumas dessas bibliotecas são Microsoft Robotics Studio, *Tekkotsu*, *Robot Operating System (ROS)*, *URBI (Universal Real-Time Behavior Interface)* e a *Player*. Todas a bibliotecas mencionadas foram desenvolvidas para pesquisas académicas e aplicações industriais robóticas, no entanto a dLife foi concebida especificamente para o ensino e pesquisa.

De acordo com o autor atualmente a dLife suporta sete robots educacionais e de pesquisa dentro de uma gama de preços, características e tamanhos.

Refere igualmente o autor que o centro de controlo da dLife fornece uma forma simples de escrever, compilar e executar controlos robóticos utilizando a biblioteca dLife. O centro de controlo é executado num computador host e faz a comunicação através de uma rede sem fios com os robots para enviar e receber dados dos sensores.

Cada vez mais, até mesmo os robots de baixo custo são projetados para o ensino, vêm equipados com camaras que suportam cor. O *input* de vídeo fornecido por essas camaras tem o potencial para a incorporação da localização do objeto, identificação de objetos e marcos baseados no mapeamento e

navegação.

A dLife inclui pacotes para redes neurais, algoritmos genéticos entre outros. Esses pacotes suportam a utilização da sala de aula e pesquisas em relação à inteligência artificial e nas áreas de aprendizagem máquina.

Ainda referenciado pelo autor, dLife contém a simulação de um micromundo chamado de “Simple World”, que proporciona a exploração de tópicos tais como raciocínio automatizado, reforço de aprendizagem (ex. Q-Learning) e a evolução de neuro-controladores; todas as tarefas que são tipicamente mais difíceis com os valores contínuos dos sensores e parâmetros do hardware robótico.

Esta biblioteca foi utilizada numa variedade de níveis de currículos tanto em pequenos componentes como plataforma principal para atribuições. Fora das salas de aula, a dLife tem apoiado uma série de implementações de projetos e pesquisas independentes de alunos graduados, sendo que com a chegada constante de novos robots e dispositivos, o potencial da dLife para implementações de projetos de estudantes parece ilimitado.

Os projetos de pesquisa independentes dos estudantes incluíram: algoritmo genético baseado no algoritmo de localização do robot, melhoria do desempenho dos controlos dos robots envolvidos em simulações quando executados em robots físicos, estudo auto-adaptativo das taxas de mutação em coevolução de neuro-controladores de um jogo tag robot e a utilização do reforço de aprendizagem para selecionar as ações do robot de forma a satisfazer os objetivos pretendidos.

Em suma, a biblioteca java dLife foi concebida para permitir que os estudantes aprendessem o conteúdo do curso de informática através da robótica, sem ficarem presos a detalhes técnicos desnecessários.

2.4. Conclusão

Neste capítulo respondemos às questões identificadas no primeiro objetivo desta tese. Identificámos como principais plataformas o robot Lego Mindstorm NXT, o robot Finch, o robot farrusco e o robot Hemisson. Foram identificadas as principais características destas plataformas, em que contextos estas são utilizadas, sendo o contexto educacional o mais utilizado. Relativamente às linguagens de programação utilizadas para comunicar com estas plataformas, foram identificadas várias, desde linguagens proprietárias e linguagens *open-source*.

Em análise ao trabalho relacionado identificamos alguns projetos desenvolvids nesta área, como o projeto tekkotsu, URBI e a biblioteca dLife. Em análise ao trabalho relacional concluímos que existe uma crescente necessidade de criar mecanismos de abstrair os conceitos de hardware e de programação de baixo nível dos utilizadores de sistemas robotizados.

Capítulo 3
Análise de Requisitos

Neste capítulo encontra-se descrita uma análise e especificações de requisitos, são identificados os principais requisitos: funcionais, não funcionais, de qualidade, e ambientais, que a biblioteca deverá satisfazer. Consequentemente são também identificadas as principais funcionalidades da biblioteca resultante, assim como os seus atores e respetivas prioridades. São também apresentados os diagramas de Use cases e respetivas descrições.

Em função da literatura revista e da comparação efetuada aos diferentes robots, no âmbito deste trabalho foi selecionado o robot farrusco (ver figura 2.1-5), dado que apresenta uma plataforma *open--source* concretamente uma placa compatível com Arduino. Desta forma a biblioteca desenvolvida não se encontra limitada a uma plataforma proprietária podendo ser utilizada em qualquer robot compatível com a plataforma arduino.

3.1. Análise e especificação de requisitos

As atividades de análise concentram-se na identificação, especificação e descrição dos requisitos do sistema de *software*. Em resumo, requisito é uma necessidade que o *software* deve cumprir.

A fase de análise e especificação de requisitos inicia-se com o levantamento das necessidades do cliente, ou seja, quando a equipa responsável pelo desenvolvimento da aplicação tenta compreender as necessidades do seu cliente e o que este deseja do *software*.

Requisitos são características ou funcionalidades que um sistema deve conter e o seu levantamento deve ser efetuado junto do cliente visto que é essencial que este os verifique, de modo a que não se implemente um sistema diferente do pretendido. Contudo, é necessário ter presente que todos os requisitos devem ser realistas e verificáveis. Se um requisito não for realista é impossível que seja implementado utilizando a tecnologia atual e se não for verificável nunca poderá ser bem implementado visto que não poderá ser testado.

Esta fase constitui assim uma etapa fundamental do ciclo de vida do *software*, pois além de definir quais são os seus requisitos, especifica-os detalhadamente.

3.2. Requisitos da aplicação

Nesta secção são apresentados todos os requisitos e especificidades das bibliotecas e aplicações desenvolvidas. Primeiramente é definido o ambiente em que a biblioteca se vai integrar e seguidamente definem-se os requisitos funcionais, apresentando todas as funcionalidades, utilizadores e diagramas.

3.2.1. *Requisitos Funcionais*

Nesta secção são identificados todos os conceitos, funcionalidades, características e informações que o sistema deverá fornecer aos utilizadores finais. Vários cenários são contemplados e as diversas situações vêm a sua implementação ordenada por prioridade.

FUNCIONALIDADES

Nesta secção são identificados os módulos e os diferentes requisitos funcionais inicialmente considerados para esta aplicação.

A prioridade de implementação de cada módulo é atribuída segundo a técnica MoSCoW bem como os seus requisitos. MoSCoW é uma técnica de priorização utilizado em análise de negócios (*business analysis*) e desenvolvimento de software (*software development*) para se chegar a um consenso com as partes interessadas (stakeholders) sobre a importância que se coloca à entrega de cada requisito.

Nomenclatura **M**o**SC**o**W**:

- ✓ **M** – Must Have (é essencial implementar)
- ✓ **S** – Should Have (é importante implementar)
- ✓ **C** – Could Have (é interessante implementar)
- ✓ **W** – Want to Have (poderá ser implementado no futuro)

Apesar de um módulo ter prioridade baixa, os seus requisitos podem ter uma prioridade maior visto que se tornam necessários caso se decida implementar um certo módulo.

Tabela 3.2.1-1 Módulos dos requisitos

Módulos	Prioridades
Comunicação Serie e Sincronismo	Must Have
Controlo da Locomoção do Robot	Must Have
Controlo das Estruturas Móveis do Robot	Must Have
Controlo de Outros Atuadores do Robot	Must Have
Controlo de Sensores	Must Have
Controlo de Variáveis na EEPROM do Robot	Should Have
Aplicação de Testes de Funcionalidades do Robot	Should Have

Nas tabelas seguintes estão definidos os requisitos funcionais que são compostos por um *ID*, de forma a serem facilmente identificados, uma *Descrição* e a sua *prioridade*.

Tabela 3.2.1-2 Requisitos Funcionais para o Módulo Comunicação Serie e Sincronismo

Comunicação Serie e Sincronismo		
ID	**Descrição**	**Prioridade**
RF1	O sistema deverá permitir o envio de mensagens para o robot.	Must Have
RF2	O sistema deverá permitir a receção de mensagens do robot.	Must Have
RF3	O robot deverá permitir o sincronismo da execução do código no IDE do aluno simultaneamente com as ações do robot.	Must Have
RF4	O sistema deverá criar mensagens num padrão específico para o ser interpretado pelo robot.	Must Have
RF5	O sistema deverá permitir a comunicação com o robot através de Bluetooth.	Must Have
RF6	O sistema deverá permitir que a conexão com a porta COM seja estabelecida automaticamente sem intervenção do aluno.	Must Have
RF7	O sistema deverá permitir o término da comunicação com o robot.	Must Have
RF8	O sistema deverá permitir o envio de uma mensagem padrão para identificar o robot automaticamente sem a interveniência do utilizador.	Must Have

Tabela 3.2.1-3 Requisitos Funcionais para o Módulo Controlo da Locomoção do Robot

Controlo da Locomoção do Robot		
ID	**Descrição**	**Prioridade**
RF9	O sistema deverá permitir mover o robot em frente durante uma distância em centímetros definida por parâmetro.	Must Have
RF10	O sistema deverá permitir mover o robot em frente definindo as velocidades dos motores, o delay de duração do movimento e o delay de descanso do robot após conclusão do movimento.	Should Have
RF11	O sistema deverá permitir mover o robot para trás durante uma distância em centímetros definida por parâmetro.	Must Have
RF12	O sistema deverá permitir mover o robot para trás definindo as velocidades dos motores, o delay de duração do movimento e o delay de descanso do robot após conclusão do movimento.	Should Have
RF13	O sistema deverá permitir rodar o robot para a direita, um angulo em graus definido por parâmetro.	Must Have
RF14	O sistema deverá permitir rodar o robot para a direita definindo as velocidades dos motores, o delay de duração do movimento e o delay de descanso do robot após conclusão do movimento.	Should Have

RF15	O sistema deverá permitir rodar o robot para a esquerda, um angulo em graus definido por parâmetro.	Must Have
RF16	O sistema deverá permitir rodar o robot para a esquerda definindo as velocidades dos motores, o delay de duração do movimento e o delay de descanso do robot após conclusão do movimento.	Should Have
RF17	O sistema deverá permitir virar o robot para a direita, avançando a sua posição para a frente, com velocidades e delays de default.	Should Have
RF18	O sistema deverá permitir virar o robot para a direita, avançando a sua posição para a frente, com velocidades e delays definidos pelo utilizador.	Should Have
RF19	O sistema deverá permitir virar o robot para a esquerda, avançando a sua posição para a frente, com velocidades e delays de default.	Should Have
RF20	O sistema deverá permitir virar o robot para a esquerda, avançando a sua posição para a frente, com velocidades e delays definidos pelo utilizador.	Should Have
RF21	O sistema deverá permitir virar o robot para a direita, recuando a sua posição para trás, com velocidades e delays de default.	Should Have
RF22	O sistema deverá permitir virar o robot para a direita, recuando a sua posição para trás, com velocidades e delays definidos pelo utilizador.	Should Have
RF23	O sistema deverá permitir virar o robot para a esquerda, recuando a sua posição para trás, com velocidades e delays de default.	Should Have
RF24	O sistema deverá permitir virar o robot para a esquerda, recuando a sua posição para trás, com velocidades e delays definidos pelo utilizador.	Should Have
RF25	O sistema deverá permitir mover o robot continuamente para a frente, com velocidades de default.	Must Have
RF26	O sistema deverá permitir mover o robot continuamente para a frente, com velocidades definidas pelo utilizador.	Must Have
RF27	O sistema deverá permitir mover o robot continuamente para trás, com velocidades de default.	Must Have
RF28	O sistema deverá permitir mover o robot continuamente para trás, com velocidades definidas pelo utilizador.	Must Have
RF29	O sistema deverá permitir mover o robot continuamente, com uma direção e velocidades definidas pelo utilizador.	Should Have
RF30	O sistema deverá permitir o envio de níveis correspondentes as velocidades dos motores, nas funções de movimentos contínuos.	Could Have
RF31	O sistema deverá permitir parar o movimento do robot.	Must Have

Tabela 3.2.1-4 Requisitos Funcionais para o Módulo Controlo das Estruturas Móveis do robot

Controlo das Estruturas Móveis do Robot		
ID	**Descrição**	**Prioridade**
RF32	O sistema deverá permitir mover o motor servo do robot para um determinado angulo (entre um valor máximo e mínimo definidos).	Must Have
RF33	O sistema deverá permitir consultar a posição (em graus) do motor servo do robot.	Must Have
RF34	O sistema deverá possuir uma funcionalidade pré programada no robot que faça o servo entrar em “estado de alerta” e movimentar-se continuamente de um angulo inicial para um angulo final e inverter esse movimento, até que seja enviada uma mensagem para terminar. Todos os valores devem se encontrar definidos por default.	Must Have
RF35	O sistema deverá permitir correr a função descrita no requisito 33 com variáveis definidas pelo utilizador.	Must Have
RF36	O sistema deverá permitir parar o movimento do servo descrito pelo requisito 33.	Must Have
RF37	O sistema deverá permitir alterar a velocidade do movimento descrito no requisito 33.	Must Have
RF38	O sistema deverá permitir consultar a velocidade do movimento descrito no requisito 33.	Must Have

Tabela 3.2.1-5 Requisitos Funcionais para o Módulo Controlo de Outros Atuadores do Robot

Controlo de Outros Atuadores do Robot		
ID	**Descrição**	**Prioridade**
RF39	O sistema deverá permitir acender cada uma das cores do LED RGB (vermelho, verde e azul).	Must Have
RF40	O sistema deverá permitir apagar cada uma das cores do LED RGB (vermelho, verde e azul).	Must Have
RF41	O sistema deverá permitir acender cada uma das cores do LED RGB (vermelho, verde e azul), durante um período de tempo definido por parâmetro. Após a conclusão desse tempo a cor do led deverá se apagar automaticamente.	Must Have
RF42	O sistema deverá permitir apitar o buzzer do robot com uma frequência durante um delay (duração) definidas em parâmetro.	Must Have
RF43	O sistema deverá permitir ativar o apito do buzzer do robot com uma frequência definida em parâmetro.	Must Have
RF44	O sistema deverá permitir desativar o apito do buzzer do robot.	Must Have

Tabela 3.2.1-6 Requisitos Funcionais do Módulo Controlo de Sensores do Robot

Controlo de Atuadores Sensoriais do Robot		
ID	**Descrição**	**Prioridade**
RF45	O sistema deverá permitir a leitura dos bumpers do robot, após um deles ser pressionado, no formato de uma string.	Must Have
RF46	O sistema deverá permitir a leitura dos bumpers do robot, após um deles ser pressionado, no formato de um array de inteiros.	Must Have
RF47	O sistema deverá permitir a leitura dos bumpers do robot, no momento, no formato de uma string.	Must Have
RF48	O sistema deverá permitir a leitura dos bumpers do robot, no momento, no formato de um array de inteiros.	Must Have
RF49	O sistema deverá permitir consultar se o bumper esquerdo se encontra pressionado.	Must Have
RF50	O sistema deverá permitir consultar se o bumper direito se encontra pressionado.	Must Have
RF51	O sistema deverá permitir a leitura do valor do sensor Infravermelhos (IR).	Must Have
RF52	O sistema deverá permitir consultar se o valor do sensor Infravermelhos (IR) se encontra acima de um valor recebido por parâmetro.	Must Have
RF53	O sistema deverá permitir a leitura do nível de intensidade do sensor de IR.	Could have
RF54	O sistema deverá permitir a leitura do valor do sensor de Luz.	Must Have
RF55	O sistema deverá permitir a leitura do nível do sensor de Luz.	Could Have
RF56	O sistema deverá permitir consultar se o valor do sensor de Luz se encontra acima de um valor recebido por parâmetro.	Must Have
RF57	O sistema deverá permitir a leitura do valor do sensor de Som.	Must Have
RF58	O sistema deverá permitir a leitura do nível do sensor de som.	Could Have
RF59	O sistema deverá permitir consultar se o valor do sensor de Som se encontra acima de um valor recebido por parâmetro.	Must Have
RF60	O sistema deverá permitir a consulta dos valores de todos os Atuadores Sensoriais (num pedido só), no formato de uma string única.	Must Have
RF61	O sistema deverá permitir a consulta dos valores de todos os Atuadores Sensoriais (num pedido só), no formato de um arrai único de inteiros.	Must Have

Tabela 3.2.1-7 Requisitos Funcionais do Módulo Controlo de Variáveis na EEPROM do robot

Controlo de variáveis na EEPROM do Robot		
ID	**Descrição**	**Prioridade**
RF62	O sistema deverá permitir guardar todos os valores default de todas as funções do robot, na respetiva EEPROM.	Must Have
RF63	O sistema devera guardar os valores das posições das ligações de todos os componentes físicos do robot.	Must Have
RF64	O sistema deverá permitir consultar todos os valores que se encontram guardados na EEPROM.	Must Have
RF65	O sistema deverá permitir alterar todos os valores que se encontram na EEPROM.	Must Have
RF66	O sistema deverá possuir uma aplicação java que permita consultar e alterar todos os valores que se encontram guardados da EEPROM, de forma a permitir proceder a calibragem dos robots.	Must Have
RF67	O sistema deverá permitir que a aplicação de configuração defina a escala para cada um níveis criados, seja níveis de sensores ou níveis de velocidades de motores.	Must Have

Tabela 3.2.1-8 Requisitos Funcionais para o Módulo Aplicação de Testes de Funcionalidades

Aplicação de Testes de Funcionalidades do Robot		
ID	**Descrição**	**Prioridade**
RF68	O sistema deverá possuir uma aplicação java que permita testar todas as funcionalidades do robot implementadas na biblioteca.	Must Have

3.2.2. *Utilizadores*

Os atores que irão interagir com o sistema assim como as responsabilidades de cada um. Indicação das permissões para cada ator.

Utilizador A – Este utilizador tem acesso a todos os componentes do projeto, Biblioteca principal, Bibliotecas resultantes da biblioteca principal, Aplicação de Testes de funcionalidades e Configurador de Robots. É o ator responsável pela correta calibração dos robots.

Utilizador B – Este utilizador tem acesso apenas as Bibliotecas resultantes da biblioteca principal e a aplicação de testes de funcionalidades. Poderá aceder a algumas funções da Biblioteca principal quando autorizado pelo Utilizador A.

3.2.3. *Diagramas de Use Case*

Os Diagramas dos Casos de Utilização enquadram-se nos Diagramas Comportamentais e representam o que o sistema deve efetuar segundo a perspetiva do utilizador ou seja representa as funcionalidades propostas para o sistema. Estas funcionalidades são definidas no levantamento dos requisitos funcionais.

O ator representa o utilizador que interage com o sistema. Assim, um Caso de Utilização representa tudo o que o utilizador pode fazer no sistema desenvolvido.

Para uma melhor perceção da interação dos utilizadores com os diversos componentes do projeto são identificados os casos de utilização que poderão ser implementados ao longo do desenvolvimento do sistema.

BIBLIOTECAS

Sendo a interação com as bibliotecas realizada através de um ambiente de desenvolvimento java, toda a interação do aluno com as bibliotecas resume-se a construção de blocos de código utilizando funções da biblioteca utilizada e respetiva compilação num IDE java a escolha. Podemos visualizar uma representação dessa interação no seguinte diagrama de Use Case:

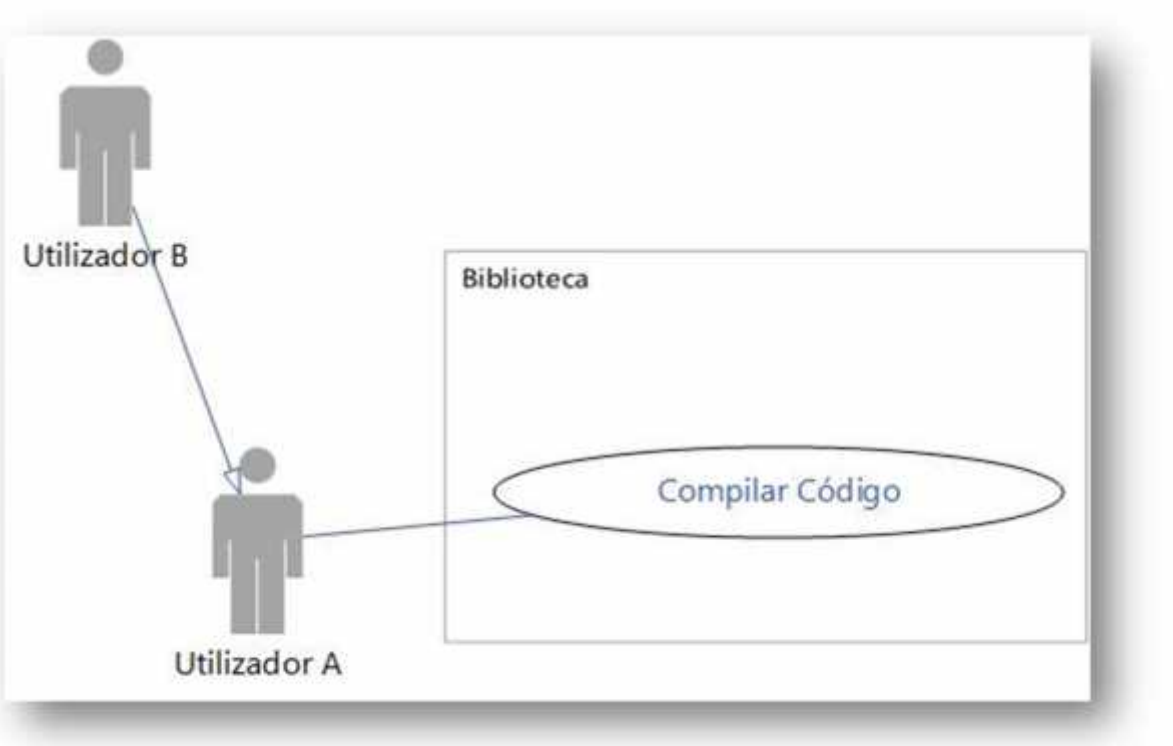


Figura 3.2-1 Diagrama de Use Cases para a Biblioteca

Tabela 3.2.3-1 Descrição do Caso de Utilização: “Compilar Código”

Caso de Utilização	CompilarCodigo
ID	CU1.
Descrição	Permite ao ator compilar o compilar o código por si efetuado.
Atores	Utilizador A Utilizador B
Pré-condições	1. O Bluetooth do computador deve estar emparelhado com o Bluetooth do robot. 2. O código construído pelo ator tem de conter a inicialização do objeto do robot. 3. O código construído pelo ator tem de acabar com a instrução para terminar a comunicação entre o computador e o robot.
Fluxo Principal	1. O ator escreve o código desejado. 2. O ator clica em “Run” no IDE. 3. O IDE compila o código. 4. Caso o código não tenha erros, o robot executa as suas instruções.
Pós-condições	Caso não existam erros de código, as instruções são executadas com sucesso.

Aplicação de Testes de Funcionalidades do Robot

Nesta aplicação pretende-se definir uma aplicação simples onde é possível testar todas as funcionalidades do robot através do simples clique de um botão.

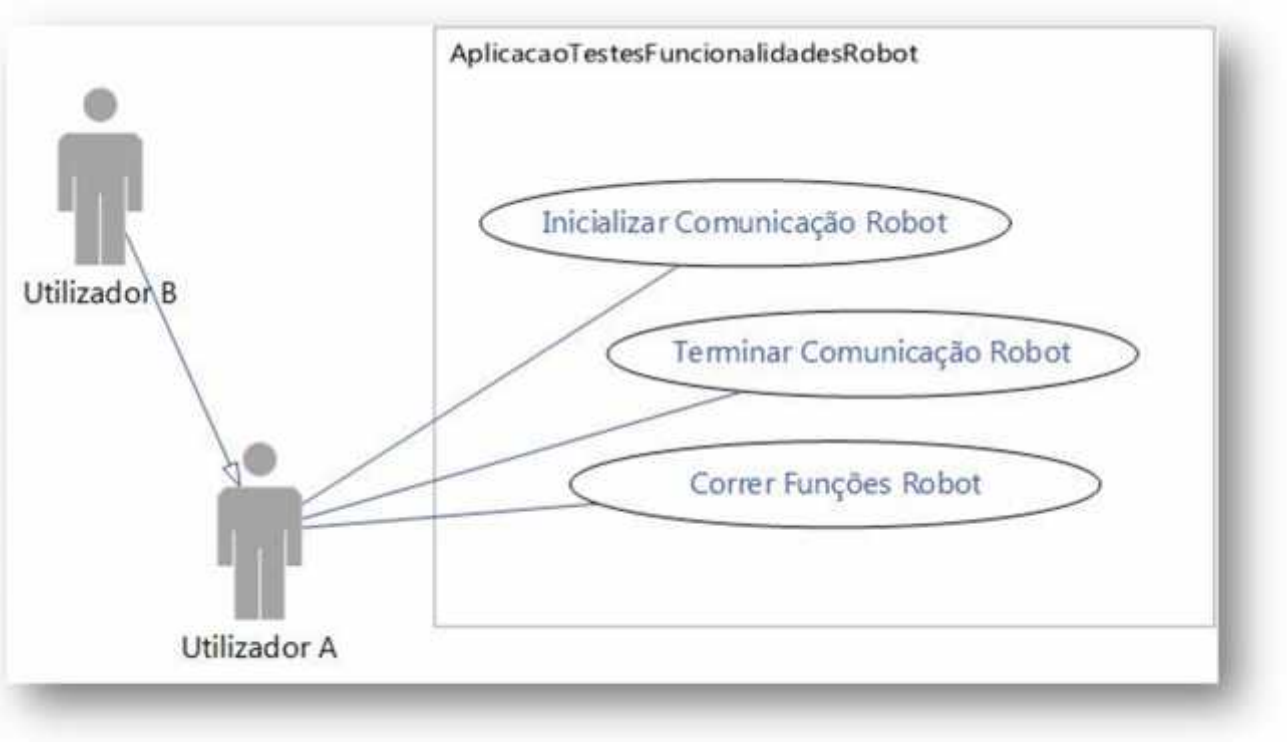


Figura 3.2-2 Diagrama de Use Cases para Aplicação de Testes de Funcionalidades do Robot

Tabela 3.2.3-2 Descrição do Caso de Utilização: “Inicializar Comunicação”

Caso de Utilização	InicializarComunicação
ID	CU2.
Descrição	Permite ao ator inicializar a comunicação entre o computador e o robot.
Atores	Utilizador A Utilizador B
Pré-condições	O Bluetooth do computador deve estar emparelhado com o Bluetooth do robot.
Fluxo Principal	1. O ator clica em “Inicializar Comunicação”. 2. A aplicação imprime no ecrã todos os passos da inicialização da comunicação.
Pós-condições	A comunicação entre o computador e o robot é criada com sucesso.

Tabela 3.2.3-3 Descrição do Caso de Utilização: “Terminar Comunicação”

Caso de Utilização	TerminarComunicação
ID	CU3.
Descrição	Permite ao ator terminar a comunicação entre o computador e o robot.
Atores	Utilizador A Utilizador B
Pré-condições	1. O Bluetooth do computador deve estar emparelhado com o Bluetooth do robot. 2. A comunicação deve estar inicializada.
Fluxo Principal	1. O ator clica em “Terminar Comunicação”. 2. A aplicação imprime no ecrã todos os passos da conclusão da comunicação.
Pós-condições	A comunicação entre o computador e o robot é concluída com sucesso.

Tabela 3.2.3-4 Descrição do Caso de Utilização: “Correr Funções Robot”

Caso de Utilização	CorrerFuncoesRobot
ID	CU4.
Descrição	Permite ao ator correr as funções do robot.
Atores	Utilizador A Utilizador B
Pré-condições	1. O Bluetooth do computador deve estar emparelhado com o Bluetooth do robot. 2. A comunicação deve estar inicializada.
Fluxo Principal	1. O ator escolhe a função que pretende correr. 2. O ator preenche o valor das variáveis que a função utiliza. 3. O ator clica no botão da função pretendida. 4. O robot recebe a instrução e executa-a, no fim devolve uma resposta para o computador. 5. A aplicação imprime no ecrã ta resposta do robot.
Pós-condições	A funcionalidade é executada no robot com sucesso.

CONFIGURADOR DO ROBOT

Nesta aplicação é possível visualizar e alterar todos as variáveis guardadas na EEPROM do robot. Estas variáveis correspondem aos valores de default das funções do robot, assim como os valores das posições das ligações de todos os componentes físicos do robot.

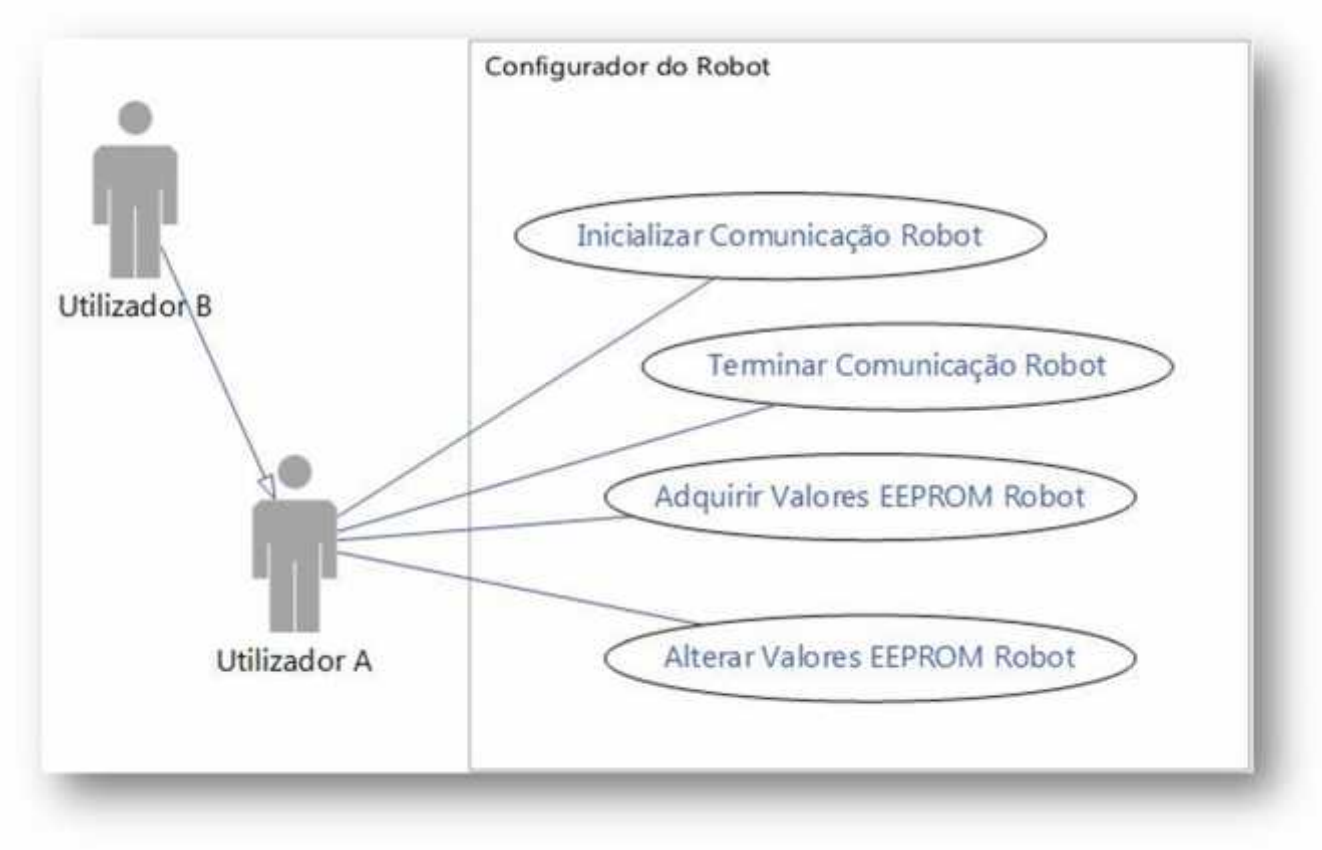


Figura 3.2-3 Diagrama de Use Cases para a aplicação Configurador do Robot

Tabela 3.2.3-5 Descrição do Caso de Utilização: “Adquirir Valores EEPROM Robot”

Caso de Utilização	AdquirirValoresEEPROMRobot
ID	CU5.
Descrição	Permite ao ator visualizar os valores guardados na EEPROM.
Atores	Utilizador A
Pré-condições	1. O Bluetooth do computador deve estar emparelhado com o Bluetooth do robot. 2. A comunicação deve estar inicializada.
Fluxo Principal	1. O ator clica em “Adquirir Valores”. 2. O robot recebe os pedidos e devolve os valores pedidos. 3. A aplicação inicializa as variáveis pretendidas.
Pós-condições	As variáveis no computador são inicializadas com sucesso.

Tabela 3.2.3-6 Descrição do Caso de Utilização: “Alterar Valores EEPROM Robot”

Caso de Utilização	AlterarValoresEEPROMRobot
ID	CU6.
Descrição	Permite ao ator alterar os valores guardados na EEPROM.
Atores	Utilizador A
Pré-condições	1. O Bluetooth do computador deve estar emparelhado com o Bluetooth do robot. 2. A comunicação deve estar inicializada. 3. Os valores devem estar inicializados
Fluxo Principal	1. O ator clica em “Guardar Valores”. 2. O robot recebe os pedidos e guarda os valores recebidos.
Pós-condições	As variáveis são guardadas na EEPROM do robot com sucesso.

3.2.4. *Requisitos Não-Funcionais*

- ✓ Este tipo de requisitos, não são definidos em termos das funcionalidades da aplicação a desenvolver.
- ✓ Definem restrições ou atributos de qualidade a um *software* e/ou ao processo de desenvolvimento desse sistema.

Tabela 3.2.4-1 Requisitos não Funcionais

Requisitos Não Funcionais		
ID	**Descrição**	**Prioridade**
RNF1	O sistema deverá ser implementado na linguagem de programação Java.	Must Have
RNF2	O sistema deverá fazer a abstração de um robot compatível com a plataforma Arduíno.	
RNF3	O sistema deverá possuir as aplicações de testes de funcionalidades e o configurador de robot desenvolvidos em Java.	Must Have
RNF4	O sistema não deverá permitir o acesso as funcionalidades responsáveis pelo estabelecimento da comunicação entre o robot e o computador aos utilizadores da biblioteca final. Esse processamento deve ser transparente para o utilizador.	Must Have
RNF5	O sistema deverá ser completamente transparente relativamente a todo o código processado pelo robot para o utilizador final da biblioteca.	Must Have

3.2.5. *Requisitos Ambientas*

A especificação de requisitos ambientais para o planeamento de um sistema de *software* é extremamente importante porque tem como objetivo guiar os programadores ao longo do ciclo de vida do software.

É nesta fase que se definem as linguagens de programação necessárias para a implementação do *software* em causa. A escolha das linguagens de programação é feita com base no tipo de aplicação a desenvolver de modo a maximizar o desempenho do sistema e a satisfação do cliente. Em paralelo a esta escolha, é feita uma análise ao tipo de *software* necessária para desenvolver a aplicação bem como aos vários requisitos mínimos a nível de hardware que o computador, que irá suportar o sistema, deverá conter.

Na secção de requisitos ambientais ficam definidos os requisitos de *hardware* do utilizador do sistema, linguagens de programação, tecnologias utilizadas e normas a serem seguidas.

Utilizador

Qualquer computador com capacidade de correr um sistema operativo, um ambiente de desenvolvimento java (recomenda-se Netbeans IDE ou eclipse IDE).

Hardware

Tabela 3.2.5-1 Requisitos mínimos do Hardware do Utilizador Final

Componente	Requisito
Processador	**Tipo de processador:** • Mínimo: Processador x86 (32-bit ou 64-bit) **Velocidade do processador:** • Mínimo: 300MHz • Recomendável: 700MHz
Memória RAM	• Mínimo: 128MB • Recomendável: 512MB ou mais • Máximo: o limite do sistema operacional
Disco Rígido	• Mínimo: 40GB (para sistema operativo, IDE java)
Vídeo	• VGA (640 × 480) ou superior
Comunicação	• Placa Bluetooth
Outro	• Teclado e rato • Drive de CD-ROM (ou superior)

Software

Tabela 3.2.5-2 Requisitos mínimos do Software do Utilizador Final

Software	
Sistema operativo:	➢ Windows (x86, x64) ➢ MacOS X (x86 and ppc) ➢ Linux (apenas x86, x86_64, ia64) ➢ Solaris (apenas sparc)
IDE Java	➢ Dado que a biblioteca é desenvolvida em java, a mesma funciona em qualquer editor de java.

Linguagens de Programação e tecnologia

Tecnologia

Linguagens de Programação

- Java
- Processing, baseado na linguagem C/C++.

Ambientes de Execução

- Arduino IDE
- NetBeans IDE (*Integrated Development Environment*)
    - Requisitos de sistema para correr o NetBeans IDE:
        - http://netbeans.org/community/releases/69/relnotes.html#system_requirements

Normas

- camelCasing para variáveis e funções
- PascalCasing para classes
- Outras boas práticas de programação e modelação de sistemas

3.2.6. *Requisitos de Qualidade*

Nesta secção ficam definidos os requisitos de qualidade a atingir pelo sistema, quando colocado em produção. São identificados o caso ideal, o pior caso, a escala de medida de qualidade e os testes a realizar para a sua medição.

A qualidade afeta diretamente a satisfação do utilizador e envolvidos com o sistema. Por isso requisitos não funcionais são importantes. A ideia é explorar essa questão para ter um cliente mais satisfeito no final do projeto.

Os requisitos de qualidade descrevem as qualidades do sistema (como o sistema é) ao invés das

suas funcionalidades.

A qualidade do software pode ser avaliada consoante as seguintes características:

- ✓ **Identificação do requisito** – identificador único do requisito de qualidade em questão
- ✓ **Característica a testar** – Qual a característica do sistema a avaliar
- ✓ **Escala** – A escala utilizada no teste
- ✓ **Teste** – O teste utilizado para analisar esta propriedade do sistema
- ✓ **Pior caso** – O pior caso para que o sistema seja considerado aceitável
- ✓ **Planeado/Desejado** – O caso ideal para que o sistema seja um sucesso

Desempenho

Esta categoria dos requisitos de qualidade engloba três pontos essenciais: **capacidade de processamento**, **capacidade de armazenamento** e **capacidade de resposta**.

Capacidade de Processamento

Consiste na capacidade que o Sistema tem para processar informação. Como teste deste ponto, iremos correr código que possua um conjunto de funções da biblioteca desenvolvida.

Tabela 3.2.6-1 Requisito de Qualidade de Desempenho – Capacidade de Processamento

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado/Desejado
RQ1	**Capacidade de processamento**	Percentagem de tempo do processador	Desempenho do processador a correr código que utilize funções da biblioteca desenvolvida.	Processador a trabalhar 100%	Processador no máximo a trabalhar a 50%

Capacidade de Armazenamento

Diz respeito à capacidade que o Sistema dispõe para guardar toda a informação pretendida. Para este teste baseámo-nos na capacidade de armazenamento na EEPROM do robot de variáveis de default das funções do robot.

Tabela 3.2.6-2 Requisito de Qualidade de Desempenho – Capacidade de Armazenamento

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado/Desejado
RQ2	**Capacidade de armazenamento**	Número de variáveis default.	A capacidade útil de armazenamento de dados na EEPROM do robot.	Falta de espaço para satisfazer as necessidades do sistema (o espaço disponível na EEPROM do robot é insuficiente para guardar todas as variáveis necessárias).	Espaço adequado para as necessidades do sistema (todas as variáveis são guardadas com sucesso)

Capacidade de resposta

Consiste na capacidade que o sistema tem para responder aos pedidos dos seus utilizadores, mais concretamente, o tempo entre a solicitação do utilizador e a apresentação do conteúdo pretendido. Para tal faz-se a medição do tempo de resposta de várias funções disponíveis na biblioteca, ou seja, o tempo entre a execução de uma instrução no IDE e a ação do robot.

Tabela 3.2.6-3 Requisito de Qualidade de Desempenho – Capacidade de Resposta

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado/Desejado
RQ3	**Capacidade de resposta**	Segundos	Tempo necessário para o robot executar uma determinada função. (tempo entre a compilação do código no IDE e a ação concreta do robot)	O sistema demora mais do que 10 segundos a iniciar uma operação	O sistema demora no máximo 1 segundo a iniciar uma operação

Disponibilidade

Este atributo de qualidade permite determinar a utilidade do Sistema no desempenho de tarefas para as quais foi concebido, e é constituído por três pontos fundamentais: **fiabilidade**, **manutibilidade** e **integridade**.

Fiabilidade

Consiste na probabilidade que a aplicação funcionar dentro dos parâmetros de qualidade definidos durante um determinado período de tempo, sob condições de funcionamento definidas anteriormente. Para este teste, iremos correr diversos blocos de código com funções do robot e iremos monitorizar o sistema durante uma semana e calcular o seu *uptime* com base nisso.

Tabela 3.2.6-4 Requisito de Qualidade de Disponibilidade – Fiabilidade

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ4	**Fiabilidade**	Número de instruções corridas com sucesso.	Utilização e monitorização da biblioteca e das suas funções durante uma semana	O sistema apresenta uma disponibilidade inferior a 99%	O sistema funcionou sem falhas com uma disponibilidade de 99.9%

Manutibilidade

Permite obter o tempo médio, entre reparações aquando da ocorrência de falhas no sistema. Introduzimos alguns erros na Biblioteca de forma a podermos medir qual o tempo aproximado que o programador demorará a corrigi-los. Se a aplicação estiver bem estruturada, irá permitir uma melhor manutibilidade, isto é, o programador demorará menos tempo a resolver os problemas.

Tabela 3.2.6-5 Requisito de Qualidade de Disponibilidade – Manutibilidade

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ5	**Manutibilidade**	Horas de trabalho para se colocar a biblioteca em funcionamento	Introduzir um conjunto de erros no código da biblioteca e medir o tempo que demora a colocá-lo de novo em perfeito funcionamento	No máximo deverá demorar-se 8 horas a colocar o sistema em funcionamento após uma falha.	No máximo deverá demorar-se 60 minutos a colocar o sistema em funcionamento após uma falha.

Integridade

A integridade mede a capacidade do sistema manter os seus dados armazenados e disponíveis de forma integra no decorrer das operações do mesmo. É importante que as funcionalidades da biblioteca sejam guardadas de forma persistente e que a utilização de funcionalidades não corrompa o seu bom funcionamento. Iremos portando medir o número de funções afetadas por erros de integridade enquanto monitorizamos o sistema.

Tabela 3.2.6-6 Requisito de Qualidade de Disponibilidade – Integridade

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ6	**Integridade**	Percentagem de funcionalidades afetadas por erros de integridade	Verificar se o sistema executa as suas operações ao longo do tempo sem que sejam corrompidas	A percentagem de funções afetadas por erros não ultrapassa 1%	Não existem funções afetadas por erros

Adaptabilidade

Este tipo de requisito de qualidade consiste numa medida que permite definir a capacidade que o Sistema tem para se adaptar a novas situações, sem perder a sua eficiência.

Extensibilidade

Consiste em medir a facilidade em adicionar novas funcionalidades ao sistema de forma fácil, e ainda aumentar a quantidade de variáveis de *default* armazenadas, tudo isto sem prejudicar as propriedades iniciais do mesmo. Para este teste iremos alterar funcionalidades do robot e avaliar o impacto geral destas alterações no sistema, nomeadamente a nível de desempenho.

Tabela 3.2.6-7 Requisito de Qualidade de Adaptabilidade – Extensibilidade

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ7	**Extensibilidade**	Percentagem de perda de performance	Verificar a resposta do sistema quando se alteram funcionalidades e novas funcionalidades são inseridas.	As novas funcionalidades funcionam perfeitamente e a performance do sistema sofre uma degradação superior a 50%	As novas funcionalidades funcionam perfeitamente e a performance do sistema sofre uma degradação inferior a 50%

Portabilidade

Capacidade do sistema correr noutras plataformas de software para além de naquela em que foi desenvolvido. Para este teste iremos correr a nossa biblioteca em vários sistemas operativos.

Tabela 3.2.6-8 Requisito de Qualidade de Adaptabilidade – Portabilidade

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ8	**Portabilidade**	N.º Sistemas Operativos suportadas	Verificar se a aplicação executa corretamente em vários Sistemas Operativos.	O sistema apenas pode ser executado apenas em Windows 7.	O sistema pode ser executado em plataforma Windows, Linux e Macintosh.

Acessibilidade

A acessibilidade está relacionada com a capacidade do produto estar disponível para o maior número de pessoas possíveis. Para tal convém assegurar que o sistema pode ser visualizado no maior número possível de IDE's. Para tal é importante testar o sistema num certo número de IDE's.

Tabela 3.2.6-9 Requisito de Qualidade de Adaptabilidade – Acessibilidade

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ9	**Acessibilidade**	N.º de IDE's suportados	Verificar se o sistema é apresentado corretamente na Netbeans, Eclipse, Gel, JEdit e OptimalJ.	O sistema só funciona corretamente nas IDE Netbeans e Eclipse.	O sistema funciona perfeitamente nos 5 IDE's testados

Usabilidade

Diz respeito à extensão na qual um produto pode ser usado por utilizadores específicos para alcançar objetivos específicos com efetividade, eficiência e satisfação num contexto de uso específico. A usabilidade divide-se em quatro sub-grupos: **facilidade de aprendizagem**, **satisfação**, **eficiência na utilização** e **resistência a erros**.

Facilidade de Aprendizagem

Este subgrupo mede a capacidade que o utilizador tem para aprender a trabalhar com o Sistema. De modo a obtermos um teste coerente, deveremos selecionar algumas pessoas de diferentes áreas, colocá-las a interagir com a biblioteca, para assim podermos ter uma visão global sobre este aspeto. Medir-se-á o tempo que os utilizadores levam até se familiarizar com todas as funcionalidades da biblioteca.

Tabela 3.2.6-10 Requisito de Qualidade de Usabilidade – Facilidade de Aprendizagem

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ10	**Facilidade de aprendizagem**	Tempo médio (em minutos) que os utilizadores demoram para aprender a utilizar a biblioteca	Medição do tempo que os utilizadores demoram a dominar as funcionalidades da biblioteca ao interagirem com o robot	Em média os utilizadores demoram 90 minutos a familiarizarem-se com a biblioteca	Em média os utilizadores demoram 30 minutos a familiarizarem-se com a biblioteca

Eficiência na utilização

Consiste em medir a eficácia do utilizador para realizar um conjunto de tarefas, para posteriormente, verificar se os resultados obtidos correspondem com os esperados. Este teste consiste em executar um conjunto de tarefas pré-determinadas e medindo o número de tentativas necessárias para tal.

Tabela 3.2.6-11 Requisito de Qualidade de Usabilidade – Eficiência na Utilização

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ11	**Eficiência na utilização**	Número de tentativas necessárias para completar as tarefas designadas	Utilizador executa um conjunto de tarefas pré-determinadas no sistema.	O utilizador executa o conjunto de ações em 20 tentativas	O utilizador executa o conjunto de ações em um máximo de 5 tentativas

Resistência a erros

Diz respeito ao número de vezes que podem ocorrer erros no Sistema. Para tal, deve-se recorrer a um grupo de pessoas e colocá-las a trabalhar com a aplicação e verificar se ocorrem alguns erros. O objetivo da aplicação será sempre não ter qualquer tipo de erro.

Tabela 3.2.6-12 Requisito de Qualidade de Usabilidade – Resistência a erros

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ12	**Resistência a erros**	Número de erros	Colocar um conjunto de pessoas a interagir com o sistema e monitorizar os erros que surjam.	O sistema apresenta 10 erros no decorrer das experiências	O sistema comporta-se como esperado e não ocorre qualquer erro

Satisfação

Consiste em medir o grau de satisfação do utilizador em relação ao Sistema. Este grau de satisfação é medido após os utilizadores utilizarem o Sistema durante um determinado período de tempo, e pode ser feito através de um questionário ou opiniões.

Tabela 3.2.6-13 Requisito de Qualidade de Usabilidade – Satisfação

ID	Req. Qualidade	Escala	Teste	Pior Caso	Planeado
RQ13	**Satisfação**	Grau de satisfação do utilizador (classificação de 1 a 5)	O utilizador interage com produto e/ou tenta obter resposta às suas questões tentando perceber se o produto o satisfaz dando seguidamente uma classificação.	O utilizador não se encontra nada satisfeito com o sistema, dando-lhe a classificação 1	O utilizador encontra-se totalmente satisfeito com o sistema, dando-lhe a classificação 5

Capítulo 4
Apresentação da Solução Proposta

Com base no enquadramento efetuado e em resultado da análise dos diferentes micro-robots efetuada na revisão da literatura, optámos por utilizar o robot farrusco. A principal característica que nos levou a optar por este robot foi o facto de este ser compatível com a plataforma Arduíno pois pelo facto de ser Arduíno, o projeto não se encontra limitado apenas ao robot em que foi desenvolvido, com possibilidade de funcionar com qualquer robot que tenha os mesmos componentes. Relativamente à linguagem utilizada no desenvolvimento da biblioteca, como já referido anteriormente, a escolha recaiu sobre a linguagem java.

Neste capítulo descrevemos em detalhe todos os constituintes da biblioteca, apresentando as soluções de implementação na construção da mesma. São detalhadas as características funcionais do robot como os controlos da sua locomoção básica, estruturas móveis do robot e o controlo dos seus sensores, assim como é especificado o processo de obtenção de informação sensorial dos seus sensores. Apresentamos as aplicações implementadas assim como uma biblioteca desenvolvida para fins de teste da biblioteca de abstração.

4.1. Soluções de Implementação

Para a implementação da biblioteca, após termos o robot e respetiva plataforma definidos, foi essencial optar pela melhor opção de linguagem de programação. Havia vários caminhos possíveis a seguir resultando em diferentes formas de se implementar as bibliotecas.

Como tal, poderíamos optar por escolher uma implementação em uma linguagem procedimental ou uma linguagem Orientada objetos. Dado que as linguagens Orientadas a Objetos têm ganho nos últimos tempos uma importância crescente optamos por uma linguagem orientada a objetos, pois permite-nos um maior conjunto de conceitos possíveis de se trabalhar. Dentro das linguagens Orientadas a objetos, optamos pela linguagem java, que para alem de ser uma linguagem *open source* e o java ser uma das linguagens mais utilizadas atualmente[21].

4.1.1. *O Robot (Farrusco)*

O Farrusco foi o robot escolhido para a concretização do projeto. Este robot é baseado em Arduíno, que era um dos principais requisitos do projeto, e possui um variado conjunto de sensores e

[21] ***Fonte***: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

atuadores que nos permitem controlar e manipular, ideal para o desafio da construção da biblioteca.

Este robot é constituído por dois motores de rotação continua (DC), um motor servo, um led RGB, e três sensores (Infravermelhos, Luz, Som), tem também um buzzer e dois bumpers de colisão, permite a comunicação sem fios através da tecnologia Bluetooth e tem uma bateria para se mover livremente.

4.1.2. *Linguagem de programação*

Nesta secção vamos descrever a linguagem de programação Orientadas a objetos Java, que resultante da RL foi a linguagem de programação escolhida para a implementação deste projeto.

A LINGUAGEM JAVA

Figura 4.1-1 Logotipo Java

Java [22] é uma tecnologia utilizada no desenvolvimento de diversas aplicações Web e outras.

Java é uma linguagem de programação de computadores. Permite que os programadores escrevam instruções de computador ao utilizar comandos baseados em inglês, em substituição de códigos numéricos. É conhecida por ser uma linguagem de "alto nível", pois pode ser lido e escrito facilmente por humanos. Assim como o Inglês, Java tem um conjunto de regras que determinam como as instruções são escritas. Essas regras são conhecidas como "sintaxe". Após um programa ser escrito e executado, as instruções de alto nível são traduzidos em códigos numéricos que os computadores conseguem entender e executar.

ORIGEM E EVOLUÇÃO

Java é uma linguagem de programação e uma plataforma de computação lançada pela primeira vez pela Sun Microsystems em 1995. É a tecnologia que capacita muitos programas da mais alta qualidade, como aplicações pessoais, jogos e aplicações para empresas, entre muitos outros. O Java é executado em mais de 850 milhões de computadores pessoais e em bilhões de dispositivos em todo o mundo, incluindo telemóveis e dispositivos de televisões.

No início dos anos noventa[23], Java foi criado por uma equipe liderada por James Gosling na Sun Microsystems. Foi originalmente desenvolvido para a utilização em dispositivos digitais móveis, como

---

[22] ***Fontes***: http://java.com/pt_BR/download/whatis_java.jsp & http://java.com/pt_BR/download/faq/whatis_java.xml & http://java.about.com/od/gettingstarted/a/whatisjava.htm

[23] Fonte: http://java.about.com/od/gettingstarted/a/whatisjava.htm

telemóveis. No entanto, quando o Java 1.0 foi disponibilizado ao público em 1996, mudaram o seu principal alvo para a utilização na Internet. Foi disponibilizada uma interatividade maior com os utilizadores, ao fornecer aos programadores uma maneira de desenvolver páginas web animadas. Atualmente de acordo com o site oficial é utilizada por mais de 6.5 milhões de programadores em todo o mundo.

UTILIZAÇÃO DE JAVA NA INTERFACE COM SISTEMAS EMBEBIDOS (DISPOSITIVOS / ROBOTS)

Um dos casos de exemplo em que a linguagem java é utilizada na abstração de um dispositivo robótico, é no robot finch.

Segundo o site oficial do robot Finch, este robot foi inicialmente projetado para o ensino de conceitos de programação em linguagem java. A biblioteca desenvolvida para além de dar total suporte ao controlo do robot java, recorre ao uso de bibliotecas de terceiros para controlar funcionalidades nativas do computador, nomeadamente ferramentas de fala, tratamento de vídeos obtidos através de webcam, recolha de dados da internet e reprodução de ficheiros de música através das colunas do computador. Juntamente com o robot finch e através da utilização dos seus sensores e atuadores, a biblioteca desenvolvida permite aos alunos criarem programas bastante completos e interativos.

4.2. Interação do utilizador com a Biblioteca

Na utilização da biblioteca desenvolvida o utilizador interage com um computador que tenha um IDE java disponível e um módulo Bluetooth, que por sua vez faz a ligação ao robot para este executar as suas funções. Demonstramos graficamente esta interação na Figura 4.2-1.

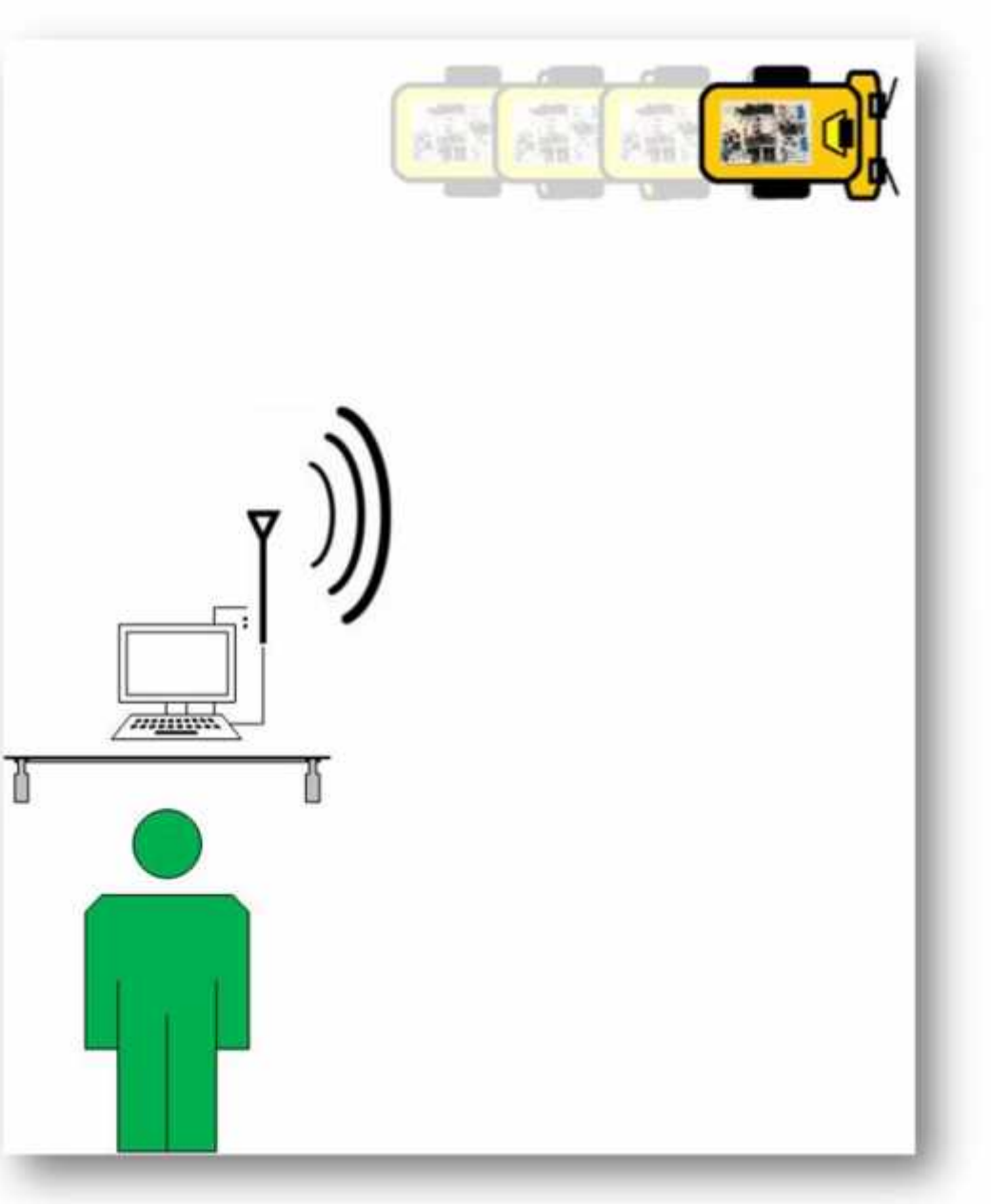


Figura 4.2-1 Demonstração da utilização da biblioteca por parte do utilizador

Como podemos visualizar na Figura 4.2-1, o utilizador apenas tem de construir código java e utilizar as funções disponíveis na biblioteca, e ao executar o seu código, tudo o resto é realizado automaticamente e totalmente transparente para o utilizador. É enviada uma mensagem do computador para o robot, o robot interpreta a mensagem recebida e age consoante a função que lhe foi enviada.

4.3. **Arquitetura da Framework**

Para o desenvolvimento das bibliotecas, foi definida uma arquitetura tal como definida na figura seguinte. Neste diagrama poderemos ver os principais componentes que constituem as Bibliotecas Desenvolvidas.

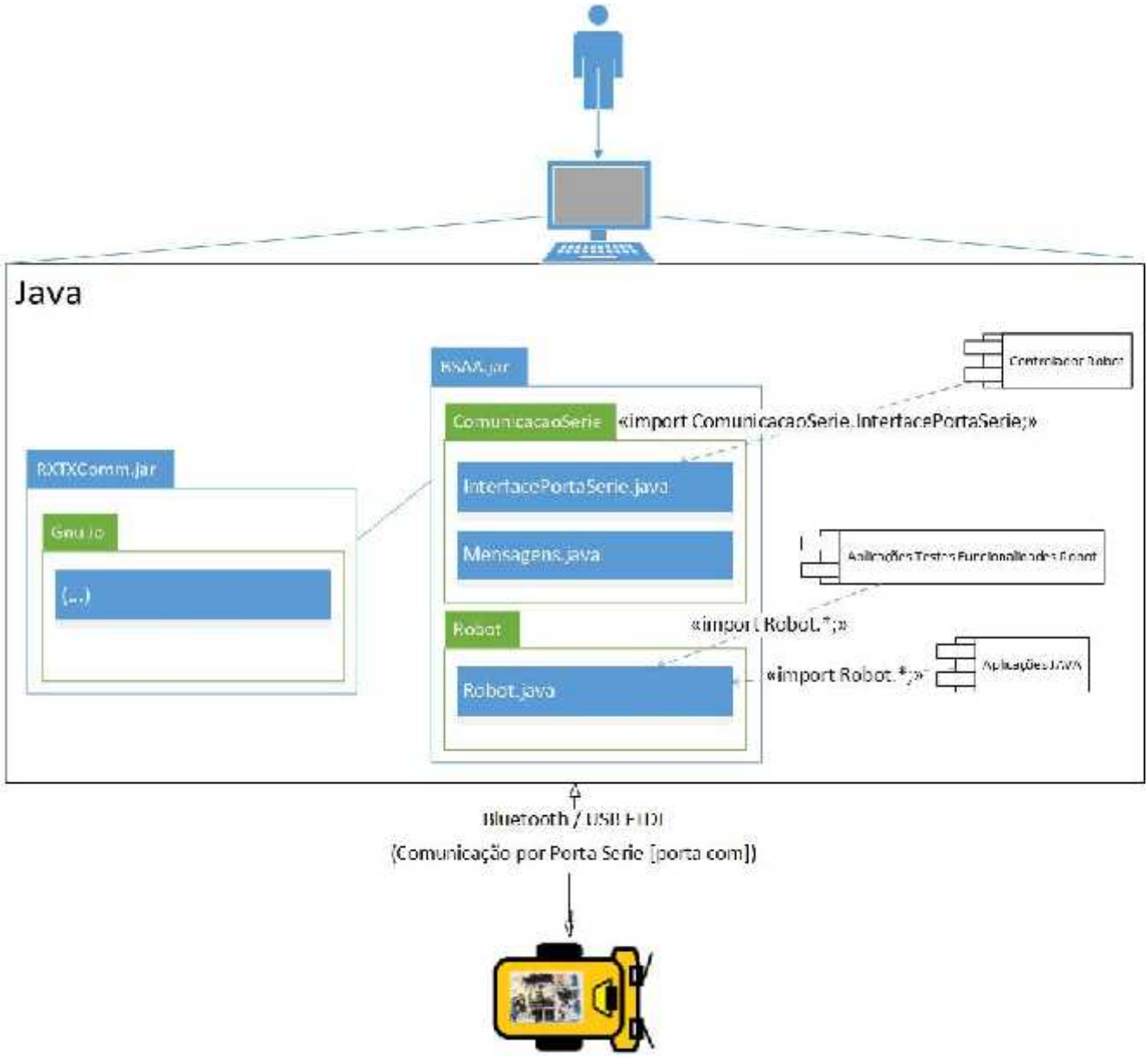


Figura 4.3-1 Arquitetura da framework

Através da arquitetura apresentada na Figura 4.3-1, podemos verificar que a biblioteca principal *BSAA.jar* é constituída por dois packages principais, *ComunicacaoSerie* e *Robot*.

O package *ComunicacaoSerie* contém as classes *InterfacePortaSerie.java* e *Mensagens.java*, que são responsáveis por toda a comunicação entre o computador e o robot, desde o estabelecimento da comunicação entre ambos ao envio/receção de mensagens. O processamento da comunicação entre

o IDE java e o robot encontra-se explicado na secção 4.4.

O package *Robot* é constituído por uma classe *Robot.java*, que contém todos os métodos das funções do robot. Todas as funções de controlo de sensores e atuadores do robot encontram-se detalhadamente explicada na secção 4.5.

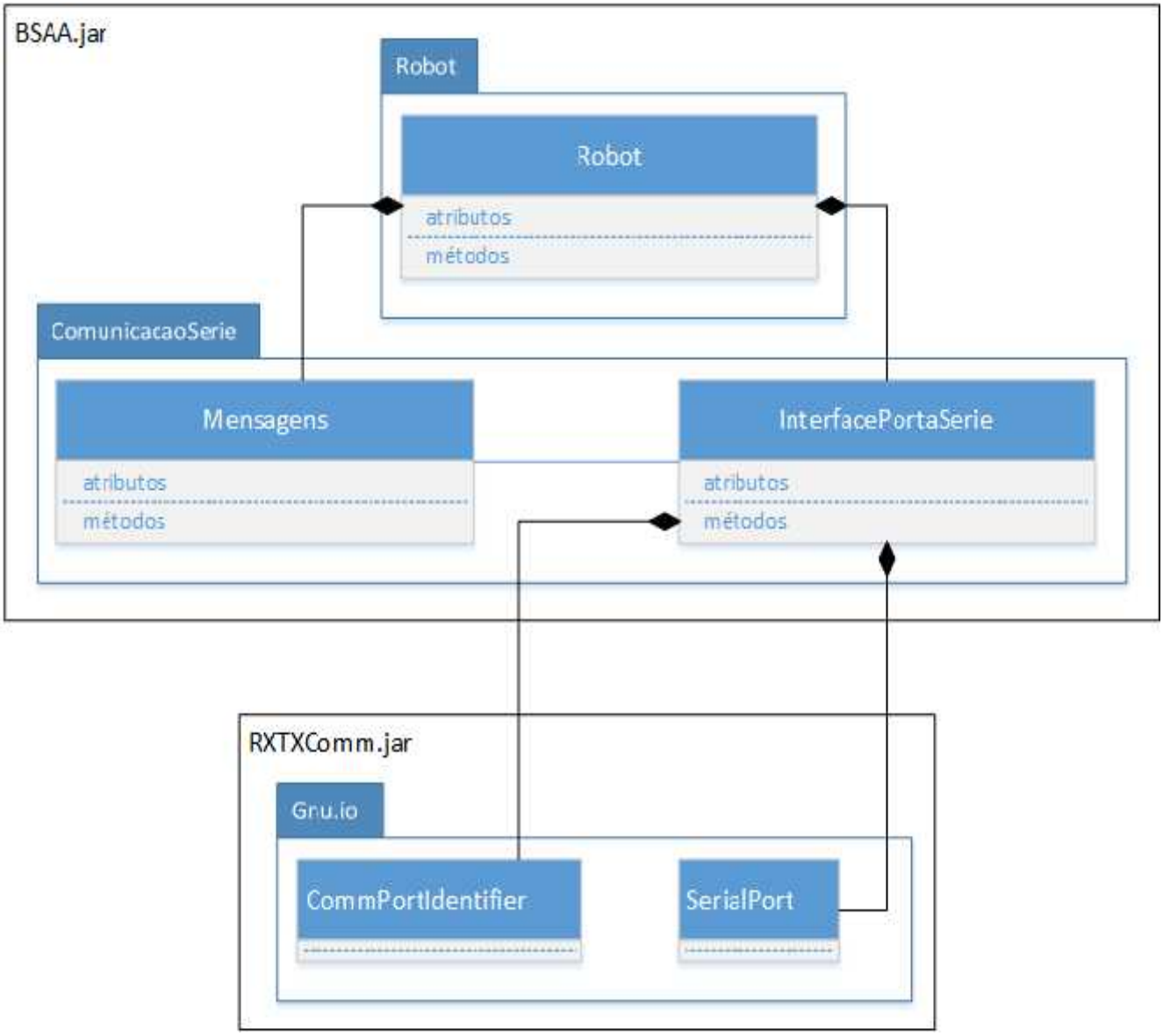


Figura 4.3-2 Diagrama de classes da biblioteca *BSAA.jar.*

Como podemos verificar no diagrama da Figura 4.3-1, a comunicação entre o IDE java e o robot é efetuada através de porta serie. Recorremos a utilização de métodos e classes da biblioteca *RXTXComm.jar*[24], (que tem de ser incluído na instalação do IDE utilizado pelo utilizador), para

[24] **RXTXComm.jar –** é uma biblioteca nativa que fornece comunicação paralela e em serie (RS232) para o Java Development Toolkit (JDK). Fonte: http://www.jcontrol.org/download/rxtx_en.html

possibilitar essa comunicação.

A biblioteca resultante pode ser utilizada na construção dos mais variados projetos que tenham como objetivo o controlo de um robot em plataforma arduíno. Juntamente com a biblioteca, foram também desenvolvidas duas aplicações java de interação com o robot. Foi desenvolvida uma "aplicação de testes de funcionalidades do robot", e um "configurador", que se encontram descritos na secção 4.7.

Como demonstrado na Figura 4.3-1, as aplicações "*Aplicação de testes de funcionalidades do robot*" assim como todas as aplicações criadas pelos utilizadores da biblioteca têm obrigatoriamente de incluir a biblioteca *BSAA.jar* e proceder ao *import* da classe *Robot* do package *Robot*. No caso da aplicação "*Configurador de Robot*" por ser uma aplicação que necessita de aceder a funções restritas ao utilizador final, exclusivamente esta aplicação faz o *import* da classe *InterfacePortaSerie.java* do package *ComunicacaoSerie*.

4.4. Processamento da comunicação entre o ambiente de execução java e o Robot

Um dos requisitos principais do projeto era que toda a comunicação entre o Computador e o Robot fosse tao transparente quanto possível para o utilizador da biblioteca. A comunicação entre os dois seria feita inicialmente através de um cabo USB FTDI (*Future Technology Devices International*) [25], e posteriormente através de Bluetooth. Foi então definido que a comunicação se processaria através de Porta serie. No caso da ligação por Bluetooth, quando os aparelhos são emparelhados é-lhes atribuído uma porta COM (por norma o processo é automático mas em alguns casos é manual), e assim a comunicação será feita a partir dessa porta COM. No caso da ligação por cabo USB FTDI, a porta COM é atribuída automaticamente.

Como um computador pode possuir várias portas COM, e as portas COM num computador podem ser todas diferentes das de outro computador, não é possível definir a qual das portas COM o robot seria conectado. Era então necessário que quando o código fosse compilado, fosse feita a ligação a porta COM correta, ou seja, a porta serie onde o robot se encontrava ligado, automaticamente e sem a intervenção do utilizador.

Para a concretização desse requisito recorreu-se a utilização de métodos e classes da biblioteca *RXTXComm.jar*. Foi definido o package ComunicacaoSerie onde estão definidas as classes que tratam da comunicação. Nesse package foram criadas duas classes, *InterfacePortaSerie* e *Mensagens*, que podemos visualizar num diagrama Uml (*Unified Modeling Language*) na Figura 4.4-1.

[25] **USB FTDI -** http://www.ftdichip.com/Products/Cables/USBTTLSerial.htm

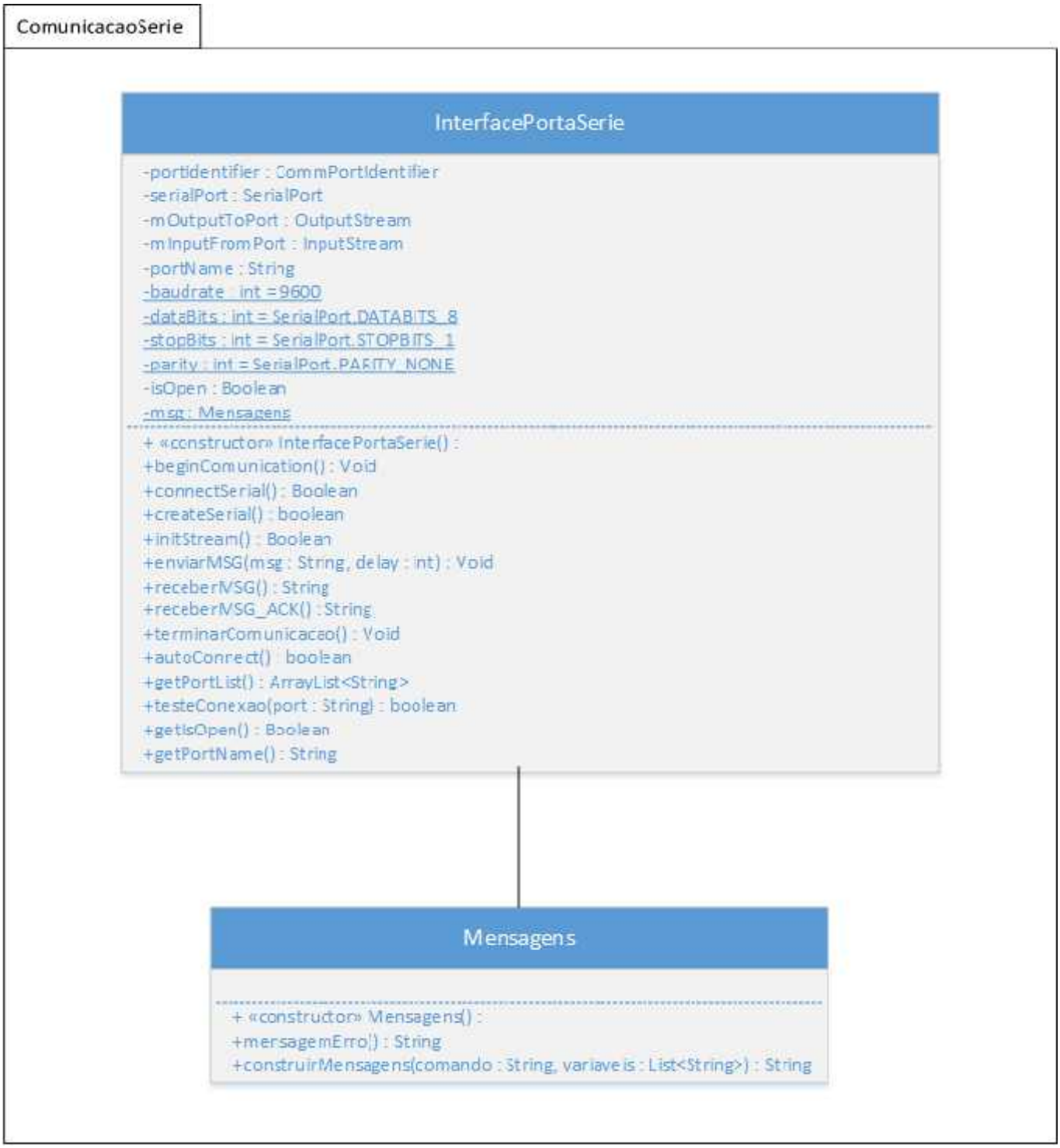


Figura 4.4-1 Diagrama Uml das classes contidas no package *ComunicacaoSerie*.

4.4.1. *Estabelecimento da comunicação*

A comunicação é estabelecida através da utilização das funções contidas na classe InterfacePortaSerie. O processo é iniciado pelo método *beginComunication, que pode ser consultado na Figura 4.4-2*

```
private void beginComunication() {
    ArrayList<String> lista = getPortList(); // recebe a lista das portas COM existentes

    System.out.println("\t|----------------------------------------------------------------------");
    System.out.print("\t|\tPortas Existentes:");

    for (String port : lista) { // imprime no ecrã a lista das postas COM
        System.out.print("\t[" + port + "]");
    }
    System.out.println();

    // tenta estabelecer automáticamente a coneção com o robot através do metodo autoConnect();
    try {
        autoConnect();
    } catch (InterruptedException ex) {
        System.out.println("\t|\tERRO[autoConnect()]: " + ex.getMessage());
    }
}
```

Figura 4.4-2 Método *beginComunication*

Este método recorre ao método *getListPort* para obter o identificador de todas as portas COM existentes no computador. Procede-se então a uma listagem de todas as portas COM existentes com o método *getPortList*, que se se pode visualizar na Figura 4.4-3.

```
private ArrayList<String> getPortList() {
    // Enumeration para guardar todos os identificadores das portas com existentes
    Enumeration pListID = CommPortIdentifier.getPortIdentifiers();

    // cria um array list
    ArrayList<String> al = new ArrayList();

    // percorre a enumeracao para aceder a todos os elementos
    while (pListID.hasMoreElements()) {
        //guarda o indentificador da porta COM
        CommPortIdentifier cpi = (CommPortIdentifier) pListID.nextElement();

        // verifica se o identificador corresponde a uma porta serie e se nao se encontra em uso
        if ((cpi.getPortType() == CommPortIdentifier.PORT_SERIAL) && !cpi.isCurrentlyOwned()) {
            // adiciona o nome da porta com ao array list
            al.add(cpi.getName());
        }
    }
    return al; // devolve o array list com o identificador de todas as portas com existentes no pc
}
```

Figura 4.4-3 Método *getPortList*

O próximo passo passa por se determinar a qual das portas se encontra conectado o Robot. Passa-se então a tentar estabelecer a comunicação com cada uma das portas COM existentes, através do método *autoConnect*, que pode ser consultado na Figura 4.4-4.

```
private boolean autoConnect() throws InterruptedException {
    // Iterator para aceder a cada um dos identificadores obtidos no metodo getPortList
    Iterator<String> it = getPortList().iterator();

    // testa cada uma das portas existentes
    while (it.hasNext()) {
        portName = it.next(); // guarda o nome da proxima porta COM
        if (testeConexao(portName)) { // testa a conexao
            System.out.println("\t|\n\t|\t\t*** Conexão efectuada com a porta: " + portName + "!! ***\n\t|\n\t|----------
            return true;
        } else {
            System.out.println("\t|  Porta pesquisada: " + portName + ", Nao foi possivel estabelecer a conexão...\n\t|----
        }
    }
    return false;
}
```

Figura 4.4-4 Método *autoConnect*.

Este método vai pegar em cada um dos identificadores obtidos no método *getPortList* e tenta estabelecer a comunicação com cada porto através do método *testeConexao*, que pode ser consultado na Figura 4.4-5.

```
private boolean testeConexao(String port) throws InterruptedException {
    System.out.println("\t|-----------------------------------------------------------------------------------------------\n"
            + "\t|  A Pesquisar pela porta: " + port);

    try {
        if (connectSerial()) {  // tenta conectar a porta serial
            System.out.println("\t|\tConectado com o PORT " + port + "!!");
            isOpen = true;
        } else {
            isOpen = false;
            System.out.println("\t|\tNão foi possivel conectar ao PORT " + port + "!!");
        }
    } catch (Exception ex) {
        System.out.println(ex.toString());
    }

    if (isOpen) { // se a conexao foi efetuada com sucesso
        enviarMSG("IEEIE", 500); // é enviado um comando de teste
        System.out.println("\t|\tEnviada Mensagem de testes para o port para estabelecer a comunicação...\n\t|\tA receber mensagem de confirmação do robot...");

        if (receberMSG_ACK().equals("ACK")) { // se receber a mensagem "ACK" a comunicaao esta entao estabelecida com o robot
            System.out.println("\t|\tMensagem recebida com sucesso!!");
            return true;
        } else { // se nao receber uma mensagem, ou esta foi incorreta é terminada a comunicacao com o porto atual
            System.out.print("\t|\tTIMEOUT! (Não obteve resposta do port)\n\t|\t");
            terminarComunicacao();
            return false;
        }
    } else {
        System.out.println("\t|\tPorta fechada por algum motivo!");
        return false;
    }
}
```

Figura 4.4-5 Método *testeConexao*.

Neste método tenta-se estabelecer uma conexão serial com a porta COM atual através do método *connectSerial*. Caso seja possível estabelecer-se a conexão *com sucesso*, ou seja, se não ocorrer nenhum erro e a porta não se encontrar já em uso, este método devolve *"true"*, e é enviado um comando de teste para essa porta. Após o envio do comando de teste existe um delay (de 500 milisegundos no *enviarMSG* e 2000 milisegundos no *receberMSG*_ACK) para a resposta do robot. Caso haja uma resposta do robot antes do tempo limite, esta é tratada para confirmar a sua origem, se a mensagem estiver correta a comunicação é estabelecida com o robot, caso seja incorreta ou o delay chegue ao fim, é emitida uma mensagem de *Timeout*, a comunicação com esta porta serial é terminada e passa-se para a próxima porta COM regressando ao método *autoconnect.* Ao voltar ao método *autoconnect* é reiniciando todo este processo ate que se encontre a porta COM onde se encontra conectado o robot ou se esgotem as portas COM existentes.

Este teste é feito em todas as portas COM até que se identifique a qual das portas o robot se encontra ligado. O diagrama de sequência simples do estabelecimento da comunicação entre o computador e o robot encontra-se definido na figura 4.4-6.

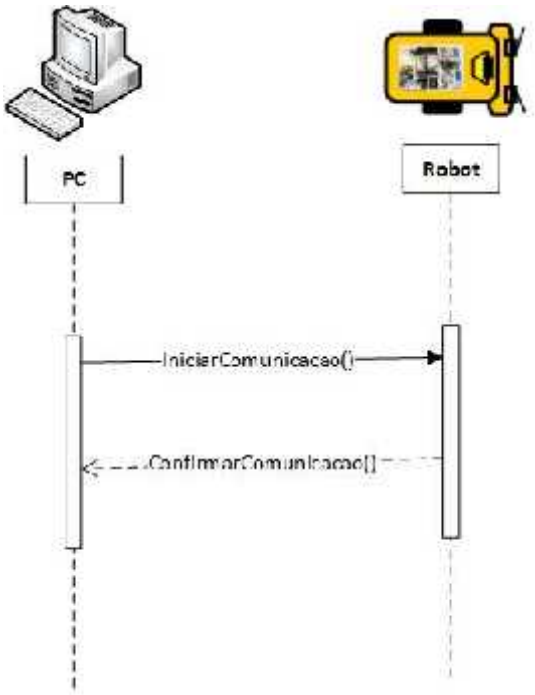


Figura 4.4-6 Diagrama de Sequencia Simples de Estabelecimento da comunicação

Todo este processo precisa de ser feito de cada vez que o código do utilizador é compilado no IDE, sendo então estritamente necessário que o utilizador inicialize a comunicação no início do seu código e que a termine no fim.

Para estabelecer a comunicação é necessário inicializar-se o objeto da classe *Robot "Robot meuRobot = new Robot();",* e para terminar a comunicação é necessário introduzir-se uma chamada do método *terminarComunicacao,* também pertencente a classe *Robot*.

Caso a comunicação não seja terminada com o método correspondente para o efeito, quando se voltar a tentar utilizar o canal de transmissão este ainda se encontrará ativo e assim impossível de se estabelecer a comunicação novamente.

4.4.2. *Envio e receção de mensagens*

Após a comunicação estar estabelecida com a porta COM correta, todas as condições para que exista comunicação entre o robot e o computador estão estabelecidas. As mensagens são enviadas através do método *enviarMSG* (Figura 4.4-7) e recebidas através do método *receberMSG* (Figura 4.4-8) da classe *InterfacePortaSerie*.

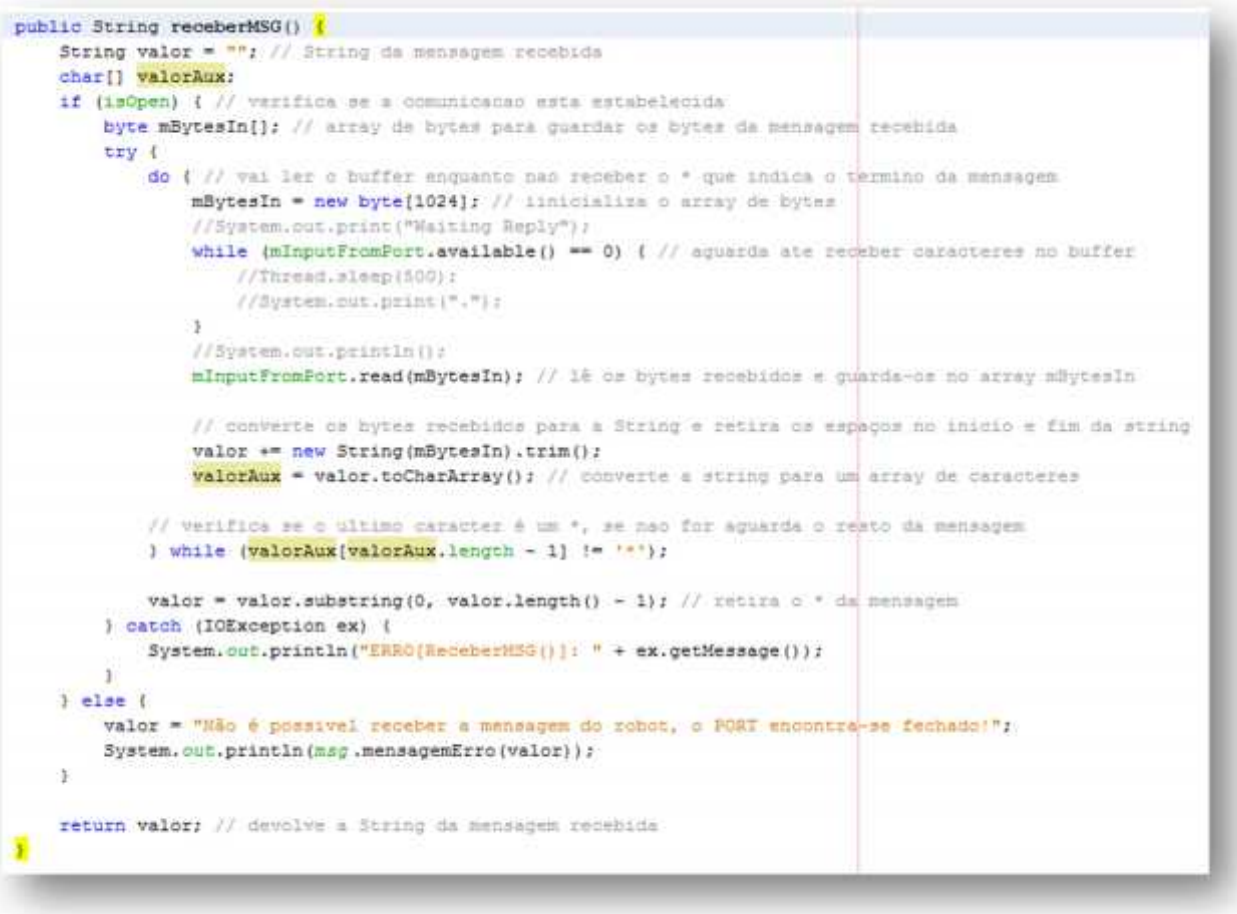
```
public void enviarMSG(String msg, int delay) {
    if (isOpen) { // verifica se a comunicacao esta estabelecida
        try {
            mOutputToPort.write(msg.getBytes()); // envia os bytes da mensagem
            mOutputToPort.flush(); // limpa o canal de transmição
            Thread.sleep(delay); // delay recebido como parâmetro
        } catch (IOException | InterruptedException ex) {
            System.out.println("ERRO[EnviarMSG()]: " + ex.getMessage());
        }
    } else {
        System.out.println(InterfacePortaSerie.msg.mensagemErro("Não é possivel enviar a mensagem para o robot, o PORT encontra-se fechado!\n"));
    }
}
```

Figura 4.4-7 Método *enviarMSG*

Neste método, a mensagem a enviar é recebida como parâmetro. É feita uma verificação para garantir que a comunicação está estabelecida e os bytes da mensagem são enviados.

O método disponível na biblioteca *RXTXComm.jar* para a leitura dos bytes da mensagem recebida não garante que a mensagem está toda completa. Então, para solucionar esse problema foi definido que cada mensagem que é enviada do robot para o IDE tem de obrigatoriamente conter um asterisco (*) no fim da mensagem. Esse asterisco serve para indicar que a mensagem chegou ao fim.

No método criado para a leitura da mensagem recebida, descrito na Figura 4.4.8. Inicialmente, assim como no método *enviarMSG*, é verificado se a comunicação esta estabelecida e caso não haja nenhum problema procede-se então a leitura dos bytes recebidos. Aguarda-se até que existam bytes no buffer para ser lidos, e no fim verifica-se se o asterisco já se encontra na última posição da mensagem recebida. Caso o asterisco ainda não faça parte da mensagem, o programa continua a aguardar que cheguem mais caracteres. A mensagem apenas é considerada completa após a chegada do “asterisco” na mensagem.



```
public String receberMSG() {
    String valor = ""; // String da mensagem recebida
    char[] valorAux;
    if (isOpen) { // verifica se a comunicacao esta estabelecida
        byte mBytesIn[]; // array de bytes para guardar os bytes da mensagem recebida
        try {
            do { // vai ler o buffer enquanto nao receber o * que indica o termino da mensagem
                mBytesIn = new byte[1024]; // iinicializa o array de bytes
                //System.out.print("Waiting Reply");
                while (mInputFromPort.available() == 0) { // aguarda ate receber caracteres no buffer
                    //Thread.sleep(500);
                    //System.out.print(".");
                }
                //System.out.println();
                mInputFromPort.read(mBytesIn); // lê os bytes recebidos e guarda-os no array mBytesIn

                // converte os bytes recebidos para a String e retira os espaços no inicio e fim da string
                valor += new String(mBytesIn).trim();
                valorAux = valor.toCharArray(); // converte a string para um array de caracteres

            // verifica se o ultimo caracter é um *, se nao for aguarda o resto da mensagem
            } while (valorAux[valorAux.length - 1] != '*');

            valor = valor.substring(0, valor.length() - 1); // retira o * da mensagem
        } catch (IOException ex) {
            System.out.println("ERRO[ReceberMSG()]: " + ex.getMessage());
        }
    } else {
        valor = "Não é possivel receber a mensagem do robot, o PORT encontra-se fechado!";
        System.out.println(msg.mensagemErro(valor));
    }

    return valor; // devolve a String da mensagem recebida
}
```

Figura 4.4-8 Método *receberMSG*

Para o correto funcionamento da biblioteca era essencial que o código desenvolvido pelos utilizadores fosse síncrono com as ações do robot. Para a implementação de algoritmos que tivessem uma representação em tempo real no robot era necessário que as suas ações fossem simultâneas com as ordens do código no IDE. Como o processamento do código no IDE é mais rápido que no Arduíno, enviando varias funções seguidas para o robot, este apenas executava uma ignorando todas as outras. Havia também a necessidade da existência de funções que abdicavam de uma resposta do robot para o computador, como é o caso das funções no configurador de robots.

Com isto em mente, na implementação das funções da biblioteca, um dos primeiros desafios, foi manter o sincronismo entre a execução do código no IDE e a execução das funções no robot, dada a diferença de velocidades de ambos. Para solucionar este requisito, foram criados dois tipos de funções: **funções bloqueantes** e **funções não bloqueantes**.

FUNÇÕES BLOQUEANTES

As **funções bloqueantes** foram criadas com o principal objetivo de manter a simultaneidade e sincronismo entre a execução do código no ambiente de desenvolvimento e a ação executada pelo robot.

O seu funcionamento traduz-se da seguinte forma:

- Começa por ser enviado um comando correspondente a uma determinada funcionalidade para o Robot.
- O código no ambiente de desenvolvimento fica bloqueado a aguardar uma resposta do robot
- O robot interpreta o comando recebido, e executa a função correspondente.
- Após a conclusão da ação do robot, este envia uma mensagem de resposta para o computador.
- A mensagem enviada pelo robot é recebida e interpretada, e prossegue a execução do restante código.

Podemos visualizar um esquema do envio de uma mensagem do computador para o robot na Figura 4.4-9 em que existe uma resposta do robot, e demonstramos um exemplo de um pedido de leitura do sensor de infravermelhos na Figura 4.4-10.

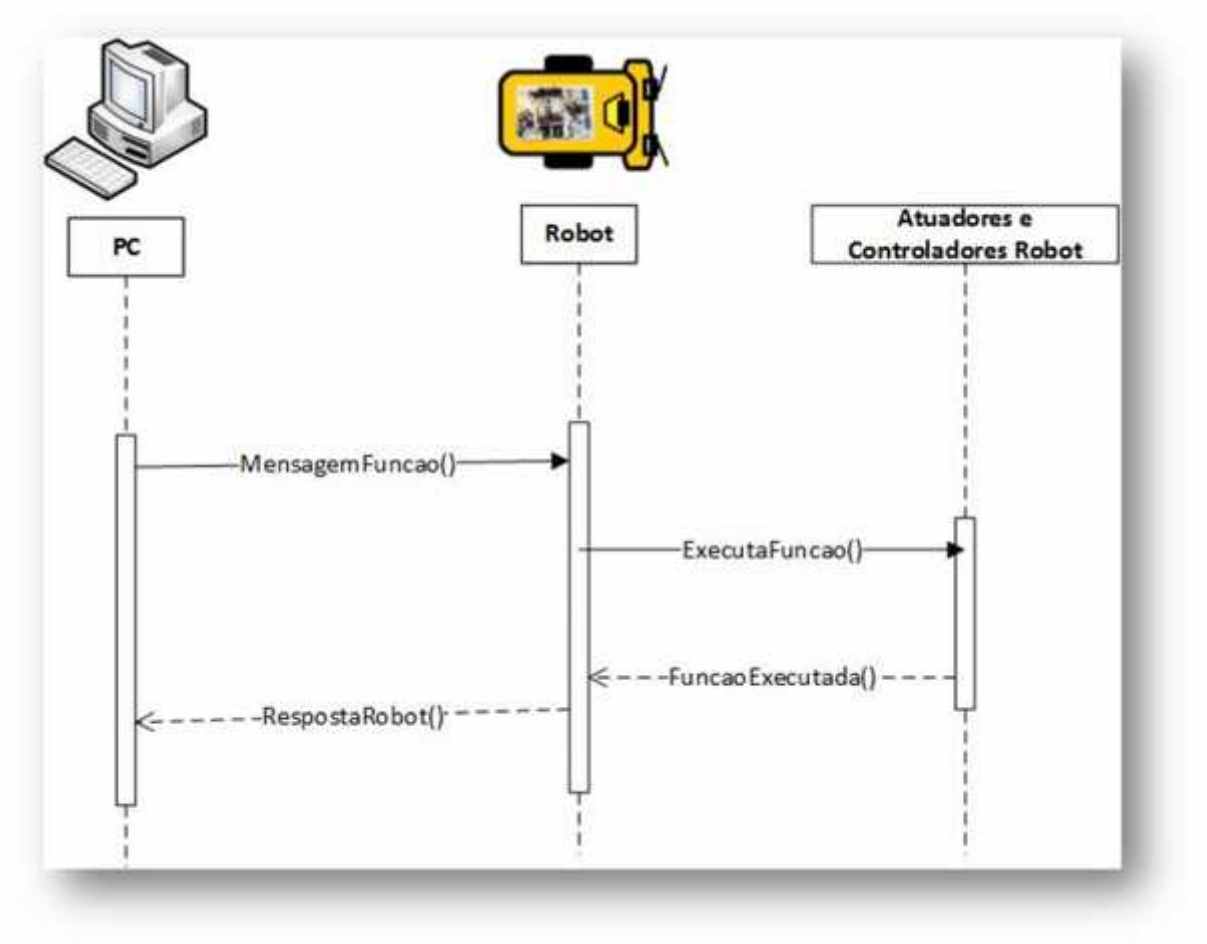


Figura 4.4-9 Diagrama de Sequencia Genérico da comunicação entre o computador e robot

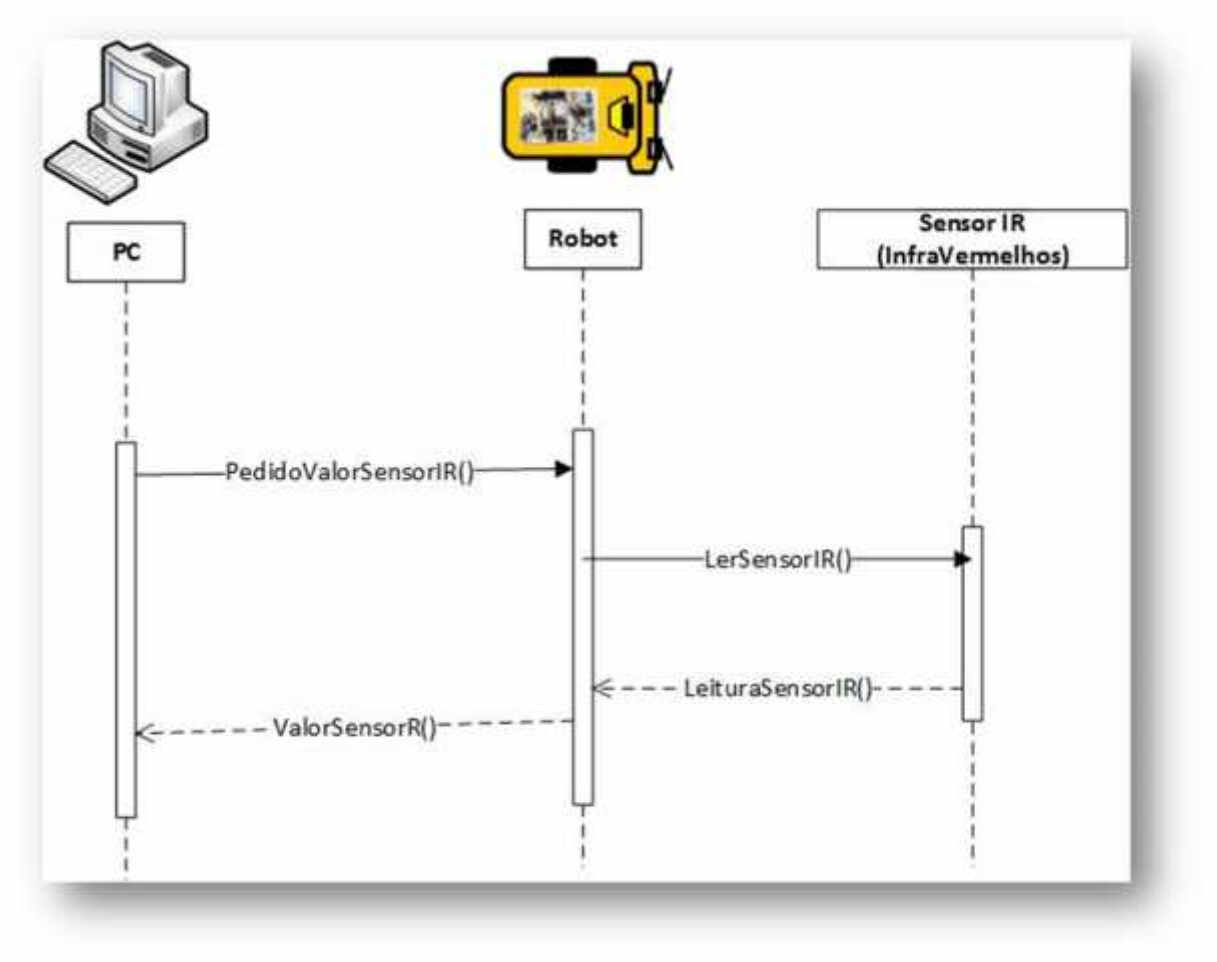


Figura 4.4-10 Diagrama de Sequencia do pedido de leitura do sensor de infra vermelhos

FUNÇÕES NÃO BLOQUEANTES

Foram também criadas funções não bloqueantes, que ao contrário das funções bloqueantes, não necessitam de uma resposta do robot para continuar a execução do restante código no ambiente de desenvolvimento.

O seu funcionamento traduz-se da seguinte forma:

- É enviado um comando correspondente a uma determinada funcionalidade para o Robot.
- O código no ambiente de desenvolvimento prossegue sem que seja necessária uma resposta do robot.
- O robot interpreta o comando recebido, e executa a função correspondente.

Demonstramos um exemplo do funcionamento de uma função não bloqueante na Figura 4.4-11.

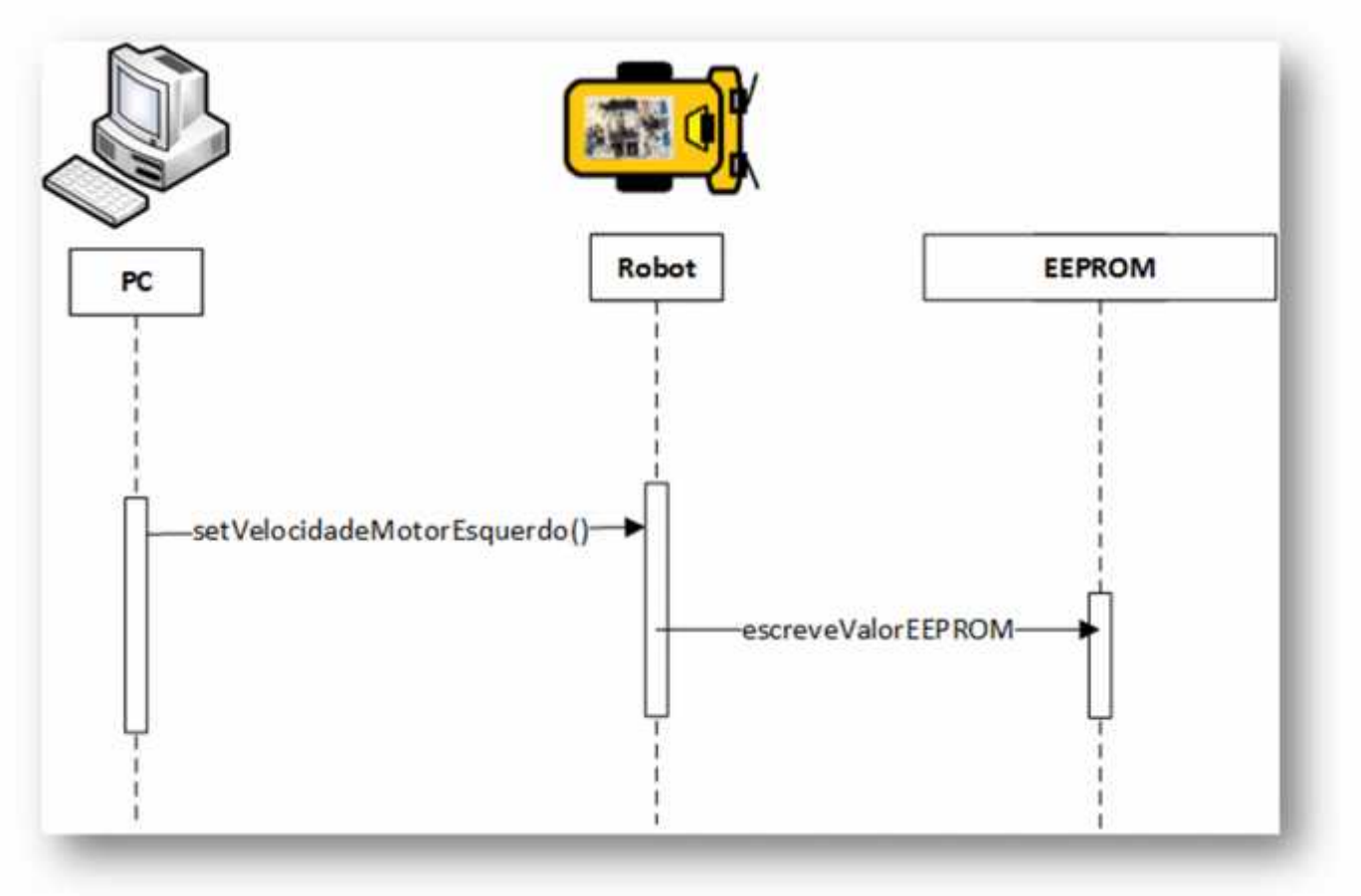


Figura 4.4-11 Diagrama de Sequência da alteração da velocidade do motor esquerdo na EEPROM do robot

4.4.3. *Construção e Constituição das mensagens*

Para cada função é enviado uma mensagem para o robot. Todas as mensagens são constituídas por um comando específico único para cada função. Existem mensagens em que apenas é enviado o comando correspondente a função e mensagens em que é necessário enviar-se parâmetros para a execução da função.

Essas mensagens são construídas pelo método *contruirMensagens* da classe *Mensagens.* A mensagem é constituída por um comando correspondente a função desejada, e no caso de ser necessário especificar os parâmetros dessa função, são enviados também os valores desses parâmetros separados por um cardinal “#”.

```
public String construirMensagens(String comando, List<String> variaveis) {
    String msg = comando;

    try {
        //Percorre todas as variaveis da lista e concactena a string separando-as com um #
        for (int i = 0; i < variaveis.size(); i++) {
            msg += "#" + variaveis.get(i);
        }
    } catch (Exception ex) {
        //Caso ocorra algum erro, é lançada uma mensagem de erro para o ecrã
        msg = mensagemErro("Erro na construcao da string de mensagem:\n" + ex.getMessage());
    }
    return msg;
}
```

Figura 4.4-12 Método construirMensagens

O método *construirMensagens* identificado na Figura 4.4-12 é o método responsável por concatenar o comando da função desejada com os parâmetros necessários para a execução da função. As mensagens ficam então com seguinte formato:

comando#variavel1#variavel2#...#variavelN.

No caso das funções em que existe resposta por parte do robot, as funções que controlam a locomoção do robot devolve dados relacionados com as velocidades dos motores e *delays* das funções. Essas mensagens são enviadas num formato idêntico:

Para as funções de movimentos pré-estabelecidos:

cmd#velocidadeMotorEsquerdo#velocidadeMotorDireito#delayDurante#delayApos

Para as funções de movimentos contínuos:

cmd#velocidadeMotorEsquerdoFrente#velocidadeMotorEsquerdoTras#velocidadeMOtorDireito Frente#velocidadeMotorDireitoTras

Foram criados alguns métodos na Classe *Robot* especificamente para tratar as mensagens enviadas pelo robot para o IDE. Encontra-se demonstrado o diagrama da classe *Robot* na Figura 4.5.2-1.

O método demonstrado na Figura 4.4-13 foi criado para tratar as mensagens recebidas do robot.

```
private String receiveMSG() {
    input = comunicacao.receberMSG(); // recebe a mensagem do robot e coloca-a na variavel input

    String cmd = extractCmdVar(); // extrai um comando da mensagem
    String funcao = "";

    // verifica qual o comando recebido do robot e constroi uma mensagem intuitiva
    if (cmd.equals("AF") || cmd.equals("AFVars")) {
        funcao = "andou em frente";
    } else if (cmd.equals("AT") || cmd.equals("ATVars")) {
        funcao = "andou para tras";
    } else if (cmd.equals("GE") || cmd.equals("GEVars")) {
        funcao = "girou para a esquerda";
    } else if (cmd.equals("GD") || cmd.equals("GDVars")) {
        funcao = "girou para a direita";
    } else if (cmd.equals("VEF") || cmd.equals("VEFVars")) {
        funcao = "virou para a esquerda avançando para a frente";
    } else if (cmd.equals("VET") || cmd.equals("VETVars")) {
        funcao = "virou para a esquerda recuando para tras";
    } else if (cmd.equals("VDF") || cmd.equals("VDFVars")) {
        funcao = "virou para a direita avançando para a frente";
    } else if (cmd.equals("VDT") || cmd.equals("VDTVars")) {
        funcao = "virou para a direita recuando para tras";
    } else if (cmd.equals("LF")
            || cmd.equals("LT")
            || cmd.equals("L")) {
        return funcoesLoop(cmd);
    }

    // constroi uma mensagem intuitiva para o utilizador
    String rsp = "O Robot " + funcao
            + " com Velocidades nos Motores(Esquerdo:" + extractCmdVar() + " Direito:" + extractCmdVar()
            + ") Delays(Durante:" + extractCmdVar() + " Após:" + extractCmdVar() + ")";

    return rsp; // devolve a resposta para o utilizador
}
```

Figura 4.4-13 Método *receiveMSG*

No método *receiveMSG* inicialmente extrai-se a primeira *substring* através do método *extractCmdVar*, demonstrado na Figura 4.4.15, que corresponde ao comando pertencente à função executada. Após termos indicação de qual função foi executada é contruída uma mensagem intuitiva para o utilizador, extraindo posteriormente todas as variáveis recebidas na mensagem e concatenando-as na mensagem final. Caso o comando recebido for um comando correspondente a uma função continua, a mensagem final é contruída pela função *funcoesLoop*, demonstrada na Figura 4.4-14.

```
private String funcoesLoop(String cmd) {
    String rsp = "";

    int valEsqF = Integer.parseInt(extractCmdVar());
    int valEsqT = Integer.parseInt(extractCmdVar());
    int valDirF = Integer.parseInt(extractCmdVar());
    int valDirT = Integer.parseInt(extractCmdVar());

    switch (cmd) {
        case "LF":
            rsp = "O Robot move-se para a frente com as seguintes Velocidades nos Motores(Esquerdo:" +
                    valEsqF + " Direito:" + valDirF + ")";
            break;
        case "LT":
            rsp = "O Robot move-se para tras com as seguintes Velocidades nos Motores(Esquerdo:" +
                    valEsqT + " Direito:" + valDirT + ")";
            break;
        case "L":
            rsp = "O Robot move-se com as seguintes Velocidades nos Motores(EsqFrente:" +
                    valEsqF + " EsqTras:" + valEsqT + " DirFrente:" + valDirF + " DirTras:" + valDirT + ")";
            break;
    }
    return rsp; // devolve a resposta para o utilizador
}
```

Figura 4.4-14 Método *funcoesLoop*

Foi criado um método auxiliar *extractCMDVar*, demonstrada na Figura 4.4-15, que lê a mensagem recebida, analisa-a caracter a caracter e faz uma extração de uma *string* a cada separador #.

```
private String extractCmdVar() {
    // Nota: a mensagem quando recebida do robot, vem no formato:
    // cmd#velocidadeMotorEsquerdo#velocidadeMotorDireito#delayDurante#delayApos
    // ou
    //cmd#velocidadeMotorEsquerdoFrente#velocidadeMotorEsquerdoTras#velocidadeMOtorDireitoFrente#velocidadeMotorDireitoTras

    String comando = ""; // variavel auxiliar para guardar o comando retirado da mensagem do robot
    int a = 0; // variavel auxiliar para controlar o indice na mensagem

    if (input.length() > 0) { // verifica se a mensagem recebida no input tem caracteres para analisar
        for (a = 0; a < input.length(); a++) { // percorre todos os caracteres do input
            if (input.charAt(a) != '#') { // verifica se encontra um caracter de separação #
                comando += input.charAt(a); // ate encontrar um caracter de separação, guarda os caracteres
            } else { // quando encontrar um caracter de separação, incrementa o indice e sai do ciclo for
                a++;
                break;
            }
        }
    }

    String inputAux = ""; // variavel auxiliar para guardar o input sem o comando encontrado no código acima
    if (a < input.length()) { // verifica se a mensagem não chegou ao fim
        for (int i = a; i < input.length(); i++) { // percorre a mensagem do ponto onde parou ate ao fim da string
            inputAux += input.charAt(i); // guarda o resto da mensagem na variavel auxiliar
        }
    }

    input = inputAux; // actualiza a variavel input para o valor da variavel auxiliar
    return comando; // devolve o comando extraido da mensagem input
}
```

Figura 4.4-15 Método *extractCmdVar*

4.4.4. *Receção das mensagens no robot*

Para que seja possível o robot executar as funções inscritas no código criado pelo utilizador, para cada função é enviado um código específico para o robot. Esse código é depois interpretado pelo robot que executa a função correspondente.

Semelhantemente ao processo utilizado no java para a leitura das mensagens enviadas pelo robot, no arduíno temos o método *getVal,* que utiliza o método *extractCmdVar* (que é usado para extrair o comando e parâmetros da função) e converte a *string* recebida num número inteiro.

Figura 4.4-16 Representação das classes *CONTROLADOR*_ROBOT e *InterfacePortaSerie* do Robot

Demonstramos na Figura 4.4-17 o método *tratarComunicacaoComm*, que é o método que gere a comunicação entre o robot e o computador, da parte do Arduíno.

```
void tratarComunicacaoComm()
{
  input = "";
  input = lerSerial();

  String cmd = "";
  cmd = extrairCmdVar();

  if(cmd.length() > 0)
  {
    //------------------------------------------------------ Mensagem para estabelecer a comunicacao
    if(cmd.equals("TESTE")){ Serial.print(ACK); }

    //------------------------------------------------------ Controlo Motores
    else if(cmd.equals("Parar")){
      cmdParar(0);
      Serial.print("parados*");
    }
    else if(cmd.equals("AF")){ cmdAndarFrente(getVelocidadeDireito(),getVelocidadeEsquerdo(),getDelayAndarFrente(),0); }
    else if(cmd.equals("AFVars")){ cmdAndarFrente(getVal(),getVal(),getVal(),getVal()); }

    (...) Etc.

    //------------------------------------------------------ Comando Invalido
    else{ Serial.print("Comando Invalido!*"); }
  }
  Serial.flush();
}
```

Figura 4.4-17 Método *tratarComunicacaoComm*

Neste método temos inicialmente uma chamada ao método *lerSerial*, que se encontra demonstrado na Figura 4.4-18. O método *lerSerial* é o método responsável pela leitura da mensagem recebida. No caso do arduíno contrariamente ao java, não é necessário ser enviado um caracter que indique o término da mensagem, pois a leitura do *buffer* apenas acaba quando a mensagem chega toda.

```
String lerSerial(){
  String option = "";
  while (Serial.available()) // nao termina enquanto nao receber a mensagem toda
  {
    delay(5);  //delay para permitir o buffer encher
    if (Serial.available() > 0)
    {
      char c = Serial.read();  //le um byte do serial buffer
      option += c; //guarda o cacarter obtido na mensagem a retornar
    }
  }
  return option; // devolve a mensagem completa
}
```

Figura 4.4-18 Método lerSerial

4.5. Controlo de Sensores e Atuadores do Robot

Um robot autônomo necessita de sensores e atuadores para interagir com o meio ambiente.

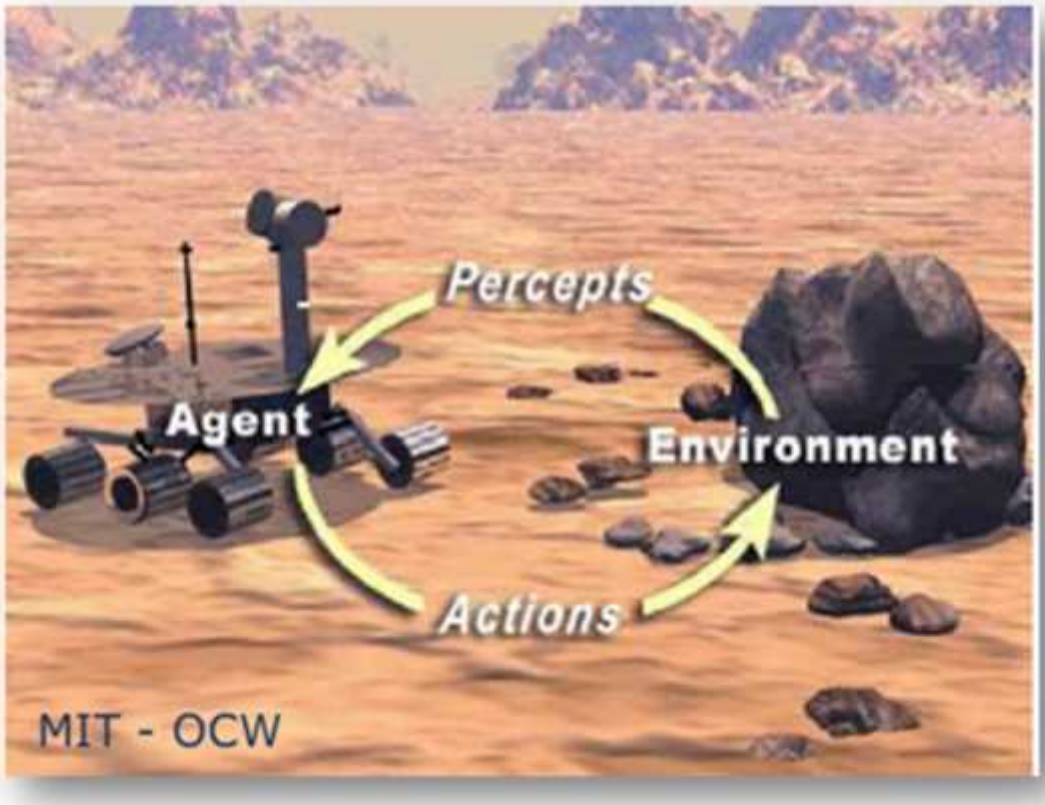


Figura 4.5-1 Interação de um robot com o meio ambiente através de sensores[26]

Baseado nos sensores e atuadores o robot deve interpretar os sinais provenientes dos sensores nas suas decisões.

Sensores são transdutores, ou seja, conversores de grandezas físicas em sinais elétricos correspondentes. Um robot é equipado com sensores para monitorar a velocidade com que se move, a posição em que se encontra, a localização de uma peça a ser manipulada, as dimensões da peça, a aproximação de um ser humano, e o impacto com um obstáculo (MORAES, 2003).

Como citado por Coelho, (Coelho, 2005), um **sensor** é caracterizado como sendo um dispositivo que, quando submetido a ação de uma quantidade física não elétrica, apresenta uma característica de natureza elétrica (ex. tensão, corrente ou impedância).

---

26 ***Fonte***: *http://bazzim.mit.edu/oeit/OcwWeb/Electrical-Engineering-and-Computer-Science/6-825Techniques-in-Artificial-IntelligenceFall2002/CourseHome/index.htm*

Atuadores são componentes que realizam a conversão da energia elétrica, hidráulica, pneumática em energia mecânica. A potência mecânica gerada pelos atuadores é enviada aos elos através dos sistemas de transmissão para que os mesmos se movimentem. Coelho, (Coelho, 2005) afirma que, assim como um sensor, um **atuador** é também ele um dispositivo conversor de energia. Contudo, e ao contrario de um sensor, um atuador converte energia elétrica em não elétrica, isto é, **um atuador executa a operação inversa que um sensor**.

No caso do robot farrusco, este utiliza como sensores, sensores de Infravermelhos, sensores de som, sensores de pressão e sensores de luz. Como atuadores, o farrusco possui dois motores DC (motores de rotação continua) para possibilitar a locomoção do robot e um motor servo que contém uma pequena plataforma com todos sensores. Possui também um led RGB e um buzzer.

4.5.1. *Níveis dos sensores e velocidades dos motores*

A existência de níveis em algumas funções do robot era parte dos requisitos do projeto. Pretendia-se que nas funções de movimentos contínuos fosse enviado um nível (entre 0 e 5) correspondente a velocidade do motor em alternativa a velocidade (entre 0 e 255) usualmente utilizada. Era também requisito do projeto a mesma lógica para as leituras dos sensores.

Para o cumprimento deste requisito foram criadas e guardadas variáveis no robot correspondente a cada nível de cada sensor e cada velocidade de cada motor. Sendo 5 níveis para cada, 3 sensores e 2 motores, foram criadas 25 variáveis. Estas variáveis encontram-se alocadas na EEPROM do robot, cujo funcionamento se encontra explicado na secção 4.7.1.

Na biblioteca foram criadas funções para os movimentos contínuos que recebiam como parâmetro os níveis das velocidades. No caso os sensores foram criadas funções que ao invés de devolverem o valor do sensor, devolviam o nível correspondente a esse valor.

Era também requisito que todos esses níveis fossem configurados numa aplicação de configuração de robots, essa aplicação encontra-se detalhada na secção 4.9. Nessa aplicação seria possível definir para cada nível um intervalo de valores correspondente a esse nível.

No caso dos níveis das velocidades dos motores, estes teriam um valor fixo e seriam criados segundo a seguinte regra: Nível 0 = 0 < Nível1 < Nível2 < Nível3 < Nível4 < Nível5 <= 255.

No caso dos sensores, estes corresponderiam a uma determinada escala [x : y], e o valor guardado seria o Y para cada sensor, seguindo as seguintes regras:

[0 : **Nivel1**], 0 >= **Nível1** < X do Nível2

[(Nivel1+1) : **Nivel2**], Nível1 > **Nível2** < X do Nível3

[(Nivel2+1) : **Nivel3**], Nível2 > **Nível3** < X do Nível4

[(Nivel3+1) : **Nivel4**], Nível3 > **Nível4** < X do Nível5

[(Nivel4+1) : **Nivel5**], Nível4 > **Nível5** <= (1023, a exceção do sensor de som que aqui é <= 10000)

Foram então guardados os 25 valores na *EEPROM* e foram criadas as funções *get* e set para cada um desses valores.

Como já referido, para o robot executar uma ação, é enviado um comando para o robot que é interpretado e a função correspondente é executada. Este é o processo utilizado para todas as funcionalidades desde uma função de deslocar o robot em frente a um *get* de uma variável. Logo, para possibilitar a existência dos níveis foram adicionados a função que faz o tratamento dessas mensagens, as instruções "*else if*" para os *gets* e *sets* das 25 variáveis.

No entanto, após se adicionar todas essas instruções, começaram a surgir algumas anormalidades no robot. Após varias tentativas de resolução do problema, este continuava a persistir. Chegamos a conclusão que o problema persistia devido a limitações de memória do Arduíno, e também devido ao excesso de instruções "*else if*" encadeadas existentes. A solução passou então por comentar todo o código relacionado com os níveis das velocidades dos motores, no arduíno, na biblioteca, na aplicação de testes de funcionalidades e na aplicação do configurador, fazendo desaparecer essas funções do robot.

Decidimos então manter todo o código relacionado com os níveis das velocidades dos motores e respetivas funções na biblioteca e aplicações, mas comentado, pois devido a limitações de memória neste robot, não é possível termos estas funções funcionais mas num robot que possua mas memória já será possível.

4.5.2. *Package e Classe Robot no Java*

Como demonstrado nas Figuras 4.3-1 e 4.3-2, todas as funções de controlo do robot encontram-se na classe *Robot* do package *Robot*. Demonstramos em detalhe uma representação Uml da classe Robot na Figura 4.5-2. Descrevemos nas secções 4.5.2, 4.5.3, 4.5.4, 4.5.5 e 4.5.6 todos os métodos correspondestes as funções do robot que foram implementados na biblioteca.

Figura 4.5-2 Representação Uml da Classe Robot

4.5.3. *Controlo da locomoção do robot*

Para o controlo da locomoção do robot, procedeu-se ao desenvolvimento de vários métodos que permitem manipular o movimento do robot. Passaremos então a enumerar os diversos métodos, assim como a descrever pormenorizadamente o comportamento e especificações de cada um deles.

Inicialmente foram desenvolvidos métodos que executam um determinado movimento com princípio e fim, estes métodos executam um determinado movimento consoante um conjunto de variáveis, que podem ser definidas pelo utilizador ou corresponderem a valores de default.

MÉTODO SLEEP(DELAY : INT) : VOID

Tabela 4.5.3-1 Detalhes do método goFront

+ sleep(delay : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
sleep	Envia uma mensagem para o robot que o coloca em delay.	Nada.

Este método tem como parâmetros:

Tabela 4.5.3-2 Parâmetros do método goFront

Nome do parâmetro	Tipo	Descrição
delay	Int	Tempo em que o robot ira ficar em modo sleep. Tem de estar entre 0 e 10000 milisegundos.

Este método envia para o robot o valor do delay que o robot ira permanecer inativo. Durante este tempo o robot ficará bloqueado. Se o robot estivesse a executar uma função com continuidade, como um mover continuamente ou buzinar continuamente, este iria continuar a executar essa mesma instrução, apenas não voltara a receber instruções ate que o tempo de delay termine.

MÉTODO GOFRONT() : STRING

Tabela 4.5.3-3 Detalhes do método goFront

+ goFront() : String		
Nome do Método	**Descrição do método**	**O que retorna?**
goFront	Neste método é enviada uma mensagem para o robot para andar em frente. Este ira se mover com velocidades e delays de default.	A resposta editada enviada pelo robot.

Este método não tem parâmetros.

Para a implementação desta funcionalidade no robot, foi necessário estimar uma combinação da velocidade dos motores e delay (duração do movimento) para o robot se deslocar em frente aproximadamente 5cm.

Possuindo o robot dois motores DC para tornar possível a locomoção do robot, para a concretização deste movimento, ambos os motores são colocados em rotação com sentido horário, a uma determinada velocidade durante um determinado delay, resultando no movimento representado na Figura 4.5-3.

Figura 4.5-3 Representação do movimento goFront.

MÉTODO GOFRONT(VELOCIDADEMOTORDIREITO : INT, VELOCIDADEMOTORESQUERDO : INT, DELAY : INT, DELAYSTOP : INT) : STRING

Tabela 4.5.3-4 Detalhes do método goFront

+ goFront(velocidadeMotorDireito : int, velocidadeMotorEsquerdo : int, delay : int, delayStop : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
goFront	É enviada uma mensagem para o robot para andar em frente. Este ira se mover com velocidades e delays definidos pelo utilizador.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.5.3-5 Parâmetros do método goFront

Nome do parâmetro	Tipo	Descrição
velocidadeMotorDireito	int	Velocidade que se ira mover o motor direito. Tem de estar entre 0 e 255.
velocidadeMotorEsquerdo	Int	Velocidade que se ira mover o motor esquerdo. Tem de estar entre 0 e 255.
delay	Int	Tempo em que o robot se vai mover. Tem de estar entre 0 e 10000 milisegundos.
delayStop	int	Tempo que o robot fica parado após concluir o movimento. Tem de estar entre 0 e 10000 milisegundos.

Para a implementação desta funcionalidade no robot, ambos os motores são colocados em rotação com sentido horário, com a velocidade enviada no método pelo utilizador. Este movimento ira se prolongar durante o delay definido pelo utilizador e após a conclusão do movimento, o robot ficará imóvel durante o período de tempo definido em delayStop. O movimento executado pelo robot é semelhante ao representado na Figura 4.5-3.

MÉTODO GOBACK() : STRING

Tabela 4.5.3-6 Detalhes do método goBack

+ goBack() : String		
Nome do Método	**Descrição do método**	**O que retorna?**
goBack	Neste método é enviada uma mensagem para o robot para andar para trás. Este ira se mover com velocidades e delays de default.	A resposta editada enviada pelo robot.

Este método não tem parâmetros.

Para a implementação desta funcionalidade no robot, assim como na função goFront, também nesta função foi necessário estimar uma combinação da velocidade dos motores e delay (duração do movimento), neste caso para o robot se deslocar para trás, aproximadamente 5cm.

Na concretização deste movimento, ambos os motores são colocados em rotação com sentido anti-horário, com uma determinada velocidade durante um determinado delay, resultando no movimento representado na Figura 4.5-4.

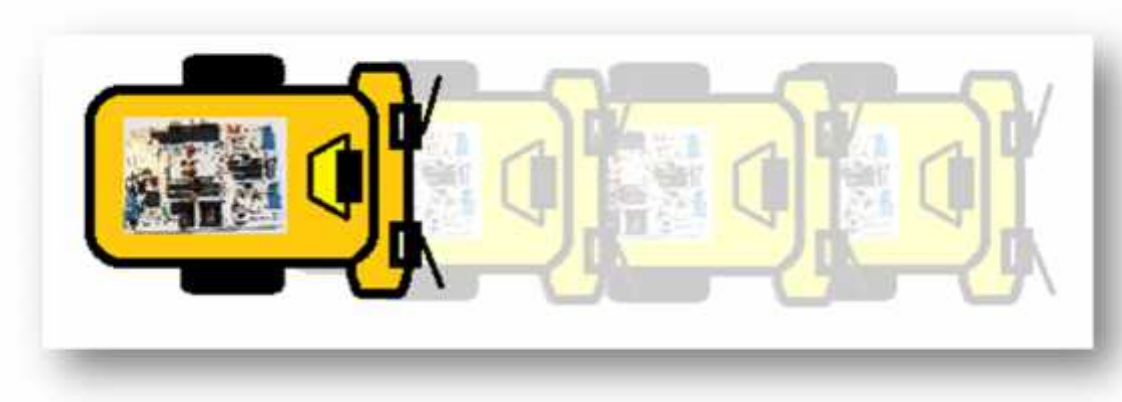


Figura 4.5-4 Representação do movimento goBack

MÉTODO GOBACK(VELOCIDADEMOTORDIREITO : INT, VELOCIDADEMOTORESQUERDO : INT, DELAY : INT, DELAYSTOP : INT) : STRING

Tabela 4.5.3-7 Detalhes do método goBack

+ goBack(velocidadeMotorDireito : int, velocidadeMotorEsquerdo : int, delay : int, delayStop : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
goBack	É enviada uma mensagem para o robot para andar para trás. Este ira se mover com velocidades e delays definidos pelo utilizador.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.5.3-8 Parâmetros do método goBack

Nome do parâmetro	Tipo	Descrição
velocidadeMotorDireito	int	Velocidade que se ira mover o motor direito. Tem de estar entre 0 e 255.
velocidadeMotorEsquerdo	Int	Velocidade que se ira mover o motor esquerdo. Tem de estar entre 0 e 255.
delay	Int	Tempo em que o robot se vai mover. Tem de estar entre 0 e 10000 milisegundos.
delayStop	int	Tempo que o robot fica parado após concluir o movimento. Tem de estar entre 0 e 10000 milisegundos.

Para a implementação desta funcionalidade no robot, ambos os motores são colocados em rotação com sentido anti-horário, com a velocidade enviada no método pelo utilizador. Este movimento ira se prolongar durante o delay definido pelo utilizador e após a conclusão do movimento, o robot ficará imóvel durante o período de tempo definido em delayStop. O movimento executado pelo robot é semelhante ao representado na Figura 4.5-4.

MÉTODO TURNRIGHT() : STRING

Tabela 4.5.3-9 Detalhes do método turnRight

+ turnRight () : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnRight	É enviada uma mensagem para o robot para girar para a direita. Este ira se mover com velocidades e delays de default.	A resposta editada enviada pelo robot.

Este método não tem parâmetros.

Para a implementação desta funcionalidade no robot foi necessário estimar uma combinação da velocidade dos motores e delay (duração do movimento) para o robot efetuar uma rotação correspondente a aproximadamente 5 graus.

Neste movimento, ambos os motores são colocados em rotação, o motor esquerdo é colocado em rotação com sentido horário e o motor direito é colocado em rotação com sentido anti-horário, os motores terão uma determinada velocidade e a rotação irá prolongar-se durante um determinado delay. O movimento executado pelo robot para uma rotação de 90º a direita é semelhante ao representado na Figura 4.5-5.

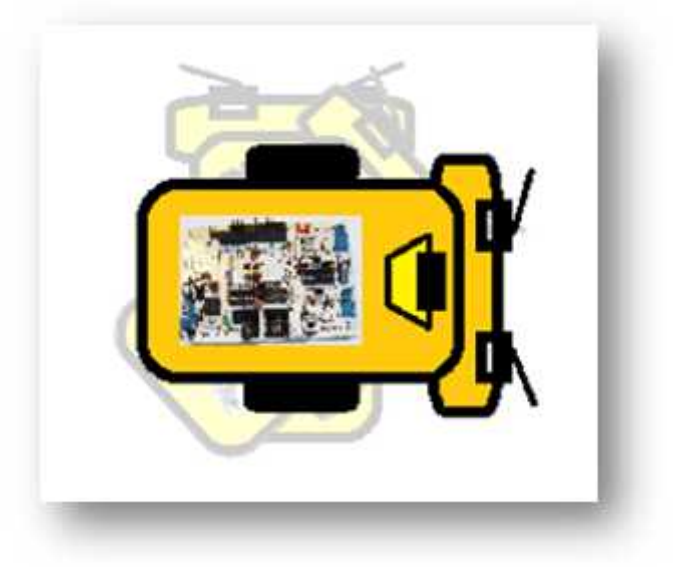


Figura 4.5-5 Representação do movimento turnRight

MÉTODO TURNRIGHT(VELOCIDADEMOTORDIREITO : INT, VELOCIDADEMOTORESQUERDO : INT, DELAY : INT, DELAYSTOP : INT) : STRING

Tabela 4.5.3-10 Detalhes do método turnRight

+ turnRight (velocidadeMotorDireito : int, velocidadeMotorEsquerdo : int, delay : int, delayStop : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnRight	É enviada uma mensagem para o robot para girar para a direita. Este ira se mover com velocidades e delays definidos pelo utilizador.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.5.3-11 Parâmetros do método turnRight

Nome do parâmetro	Tipo	Descrição
velocidadeMotorDireito	int	Velocidade que se ira mover o motor direito. Tem de estar entre 0 e 255.
velocidadeMotorEsquerdo	Int	Velocidade que se ira mover o motor esquerdo. Tem de estar entre 0 e 255.
delay	Int	Tempo em que o robot se vai mover. Tem de estar entre 0 e 10000 milisegundos.
delayStop	int	Tempo que o robot fica parado após concluir o movimento. Tem de estar entre 0 e 10000 milisegundos.

Para a implementação desta funcionalidade no robot, ambos os motores são colocados em rotação, o motor esquerdo é colocado em rotação com sentido horário e o motor direito é colocado em rotação com sentido anti-horário, os motores terão uma velocidade e delay definidos pelo utilizador. Após a conclusão do movimento, o robot ficará imóvel durante o período de tempo definido em delayStop. O movimento executado pelo robot para uma rotação de 90º a direita é semelhante ao representado na Figura 4.5-5.

MÉTODO TURNLEFT() : STRING

Tabela 4.5.3-12 Detalhes do método turnLeft

+ turnLeft () : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnLeft	É enviada uma mensagem para o robot para girar para a esquerda. Este ira se mover com velocidades e delays de default.	A resposta editada enviada pelo robot.

Este método não tem parâmetros.

Para a implementação desta funcionalidade no robot foi necessário estimar uma combinação da velocidade dos motores e delay (duração do movimento) para o robot efetuar uma rotação correspondente a aproximadamente 5 graus.

Na concretização deste movimento, ambos os motores são colocados em rotação, o motor esquerdo é colocado em rotação com sentido anti-horário e o motor direito é colocado em rotação com sentido horário, os motores terão uma determinada velocidade e a rotação irá prolongar-se durante um determinado delay. O movimento executado pelo robot para uma rotação de 90º à esquerda é semelhante ao representado na Figura 4.5-6.

Figura 4.5-6 Representação do movimento turnLeft

MÉTODO TURNLEFT(VELOCIDADEMOTORDIREITO : INT, VELOCIDADEMOTORESQUERDO : INT, DELAY : INT, DELAYSTOP : INT) : STRING

Tabela 4.5.3-13 Detalhes do método turnLeft

+ turnLeft (velocidadeMotorDireito : int, velocidadeMotorEsquerdo : int, delay : int, delayStop : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnLeft	É enviada uma mensagem para o robot para girar para a esquerda. Este ira se mover com velocidades e delays definidos pelo utilizador.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.5.3-14 Parâmetros do método turnLeft

Nome do parâmetro	Tipo	Descrição
velocidadeMotorDireito	int	Velocidade que se ira mover o motor direito. Tem de estar entre 0 e 255.
velocidadeMotorEsquerdo	Int	Velocidade que se ira mover o motor esquerdo. Tem de estar entre 0 e 255.
delay	Int	Tempo em que o robot se vai mover. Tem de estar entre 0 e 10000 milisegundos.
delayStop	int	Tempo que o robot fica parado após concluir o movimento. Tem de estar entre 0 e 10000 milisegundos.

Para a implementação desta funcionalidade no robot, ambos os motores são colocados em rotação, o motor esquerdo é colocado em rotação com sentido anti-horário e o motor direito é colocado em rotação com sentido horário, os motores terão uma velocidade e delay definidos pelo utilizador. Após a conclusão do movimento, o robot ficará imóvel durante o período de tempo definido em delayStop. O movimento executado pelo robot para uma rotação de 90º a esquerda é semelhante ao representado na Figura 4.5-6.

MÉTODO TURNLEFTMOVINGFRONT() : STRING

Tabela 4.5.3-15 Detalhes do método turnLeftMovingFront

+ turnLeftMovingFront () : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnLeftMovingFront	É enviada uma mensagem para o robot para virar para a esquerda virando o robot para a frente. Este ira se mover com velocidades e delays de default.	A resposta editada enviada pelo robot.

Este método não tem parâmetros.

Na concretização deste movimento, apenas um dos motores é colocado em rotação, o motor esquerdo mantem-se imóvel e o motor direito é colocado em rotação com sentido horário, o motor direito terá uma determinada velocidade e a rotação irá prolongar-se durante um determinado delay. O movimento executado pelo robot para uma rotação de 90º a esquerda movendo o robot em frente é semelhante ao representado na Figura 4.5-7.

Figura 4.5-7 Representação do movimento turnLeftMovingFront

MÉTODO TURNLEFTMOVINGFRONT(VELOCIDADEMOTORDIREITO : INT, DELAY : INT, DELAYSTOP : INT) : STRING

Tabela 4.5.3-16 Detalhes do método turnLeftMovingFront

+ turnLeftMovingFront (velocidadeMotorDireito : int, delay : int, delayStop : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnLeftMovingFront	É enviada uma mensagem para o robot para virar para a esquerda movendo o motor direito para a frente. Este ira se mover com velocidades e delays definidos pelo utilizador.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.5.3-17 Parâmetros do metodo turnLeftMovingFront

Nome do parâmetro	Tipo	Descrição
velocidadeMotorDireito	int	Velocidade que se ira mover o motor direito. Tem de estar entre 0 e 255.
delay	Int	Tempo em que o robot se vai mover. Tem de estar entre 0 e 10000 milisegundos.
delayStop	int	Tempo que o robot fica parado após concluir o movimento. Tem de estar entre 0 e 10000 milisegundos.

Para a implementação desta funcionalidade no robot, apenas um dos motores é colocado em rotação, o motor esquerdo mantem-se imóvel e o motor direito é colocado em rotação com sentido horário, o motor direito terá uma velocidade e delay definidos pelo utilizador. Após a conclusão do movimento, o robot ficará imóvel durante o período de tempo definido em delayStop. O movimento executado pelo robot para uma rotação de 90º a esquerda movendo o robot em frente é semelhante ao representado na Figura 4.5-7.

MÉTODO TURNRIGHTMOVINGFRONT() : STRING

Tabela 4.5.3-18 Detalhes do método turnRightMovingFront

+ turnRightMovingFront () : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnRightMovingFront	É enviada uma mensagem para o robot para virar para a direita movendo o motor esquerdo para a frente. Este ira se mover com velocidades e delays de default.	A resposta editada enviada pelo robot.

Este método não tem parâmetros.

Na concretização deste movimento, apenas um dos motores é colocado em rotação, o motor esquerdo é colocado em rotação com sentido horário e o motor direito mantem-se imóvel, o motor esquerdo terá uma determinada velocidade e a rotação irá prolongar-se durante um determinado delay. O movimento executado pelo robot para uma rotação de 90º a direita movendo o robot em frente é semelhante ao representado na Figura 4.5-8.

Figura 4.5-8 Representação do movimento turnRightMovingFront

MÉTODO TURNRIGHTMOVINGFRONT(VELOCIDADEMOTORESQUERDO : INT, DELAY : INT, DELAYSTOP : INT) : STRING

Tabela 4.5.3-19 Detalhes do método turnRightMovingFront

+ turnRightMovingFront (velocidadeMotorEsquerdo : int, delay : int, delayStop : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnRightMovingFront	É enviada uma mensagem para o robot para virar para a direita movendo o motor esquerdo para a frente. Este ira se mover com velocidades e delays definidos pelo utilizador.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.5.3-20 Parâmetros do método turnRightMovingFront

Nome do parâmetro	Tipo	Descrição
velocidadeMotoEsquerdo	int	Velocidade que se ira mover o motor esquerdo. Tem de estar entre 0 e 255.
delay	Int	Tempo em que o robot se vai mover. Tem de estar entre 0 e 10000 milisegundos.
delayStop	int	Tempo que o robot fica parado após concluir o movimento. Tem de estar entre 0 e 10000 milisegundos.

Para a implementação desta funcionalidade no robot, apenas um dos motores é colocado em rotação, o motor esquerdo é colocado em rotação com sentido horário e o motor direito mantém-se imóvel, o motor esquerdo terá uma velocidade e delay definidos pelo utilizador. Após a conclusão do movimento, o robot ficará imóvel durante o período de tempo definido em delayStop. O movimento executado pelo robot para uma rotação de 90º a esquerda movendo o robot em frente é semelhante ao representado na Figura 4.5-8.

MÉTODO TURNRIGTHMOVINGBACK() : STRING

Tabela 4.5.3-21 Detalhes do método turnRigthMovingBack

+ turnRigthMovingBack () : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnRigthMovingBack	É enviada uma mensagem para o robot para virar para a direita movendo o motor direito para trás. Este ira se mover com velocidades e delays de default.	A resposta editada enviada pelo robot.

Este método não tem parâmetros.

Na concretização deste movimento, apenas um dos motores é colocado em rotação, o motor esquerdo mantem-se imóvel e o motor direito é colocado em rotação com sentido anti-horário, o motor direito terá uma determinada velocidade e a rotação irá prolongar-se durante um determinado delay. O movimento executado pelo robot para uma rotação de 90º a esquerda movendo o robot em frente é semelhante ao representado na Figura 4.5-9.

Figura 4.5-9 Representação do movimento turnRigthMovingBack

MÉTODO TURNRIGTHMOVINGBACK(VELOCIDADEMOTORDIREITO : INT, DELAY : INT, DELAYSTOP : INT) : STRING

Tabela 4.5.3-22 Detalhes do método turnRigthMovingBack

+ turnRigthMovingBack (velocidadeMotorDireito : int, sleep : int, delayStop : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnRigthMovingBack	É enviada uma mensagem para o robot para virar para a direita movendo o motor direito para trás. Este ira se mover com velocidades e delays definidos pelo utilizador.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.5.3-23 Parâmetros do método turnRigthMovingBack

Nome do parâmetro	Tipo	Descrição
velocidadeMotorDireito	int	Velocidade que se ira mover o motor direito. Tem de estar entre 0 e 255.
sleep	Int	Tempo em que o robot se vai mover. Tem de estar entre 0 e 10000 milisegundos.
delayStop	int	Tempo que o robot fica parado após concluir o movimento. Tem de estar entre 0 e 10000 milisegundos.

Para a implementação desta funcionalidade no robot, apenas um dos motores é colocado em rotação, o motor esquerdo mantem-se imóvel e o motor direito é colocado em rotação com sentido anti-horário, o motor direito terá uma velocidade e delay definidos pelo utilizador. Após a conclusão do movimento, o robot ficará imóvel durante o período de tempo definido em *delayStop*. O movimento executado pelo robot para uma rotação de 90º a esquerda movendo o robot em frente é semelhante ao representado na Figura 4.5-9.

MÉTODO TURNLEFTMOVINGBACK() : STRING

Tabela 4.5.3-24 Detalhes do método turnLeftMovingBack

+ turnLeftMovingBack () : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnLeftMovingBack	É enviada uma mensagem para o robot para virar para a esquerda movendo o motor esquerdo para trás. Este ira se mover com velocidades e delays de default.	A resposta editada enviada pelo robot.

Este método não tem parâmetros.

Na concretização deste movimento, apenas um dos motores é colocado em rotação, o motor esquerdo é colocado em rotação com sentido anti-horário e o motor direito mantém-se imóvel, o motor esquerdo terá uma determinada velocidade e a rotação irá prolongar-se durante um determinado delay. O movimento executado pelo robot para uma rotação de 90º a direita movendo o robot em frente é semelhante ao representado na Figura 4.5-10.

Figura 4.5-10 Representação do movimento turnLeftMovingBack

MÉTODO TURNLEFTMOVINGBACK(VELOCIDADEMOTORESQUERDO : INT, DELAY : INT, DELAYSTOP : INT) : STRING

Tabela 4.5.3-25 Detalhes do método turnLeftMovingBack

+ turnLeftMovingBack (velocidadeMotorEsquerdo : int, sleep : int, delayStop : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnLeftMovingBack	É enviada uma mensagem para o robot para virar para a esquerda movendo o motor esquerdo para trás. Este ira se mover com velocidades e delays definidos pelo utilizador.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.5.3-26 Parâmetros do método turnLeftMovingBack

Nome do parâmetro	Tipo	Descrição
velocidadeMotoEsquerdo	int	Velocidade que se ira mover o motor esquerdo. Tem de estar entre 0 e 255.
sleep	Int	Tempo em que o robot se vai mover. Tem de estar entre 0 e 10000 milisegundos.
delayStop	int	Tempo que o robot fica parado após concluir o movimento. Tem de estar entre 0 e 10000 milisegundos.

Para a implementação desta funcionalidade no robot, apenas um dos motores é colocado em rotação, o motor esquerdo é colocado em rotação com sentido anti-horário e o motor direito mantem-se imóvel, o motor esquerdo terá uma velocidade e delay definidos pelo utilizador. Após a conclusão do movimento, o robot ficará imóvel durante o período de tempo definido em delayStop. O movimento executado pelo robot para uma rotação de 90º a esquerda movendo o robot em frente é semelhante ao representado na Figura 4.5-10.

Para além de métodos de movimentos com início e fim preestabelecidos, existem também métodos para movimentos contínuos. Estes movimentos apenas têm definidas as velocidades para cada motor, e para terminar esse movimento foi criado um método específico para o efeito.

MÉTODO MOVEFRONT() : VOID

Tabela 4.5.3-27 Detalhes do método moveFront

+ moveFront () : void		
Nome do Método	**Descrição do método**	**O que retorna?**
moveFront	É enviada uma mensagem para o robot se mover em frente sem fim definido. Este irá se mover com velocidades default. Este movimento ira se prolongar ate que seja enviada uma mensagem em ordem do contrário.	Nada.

Este método não tem parâmetros.

Para a concretização deste movimento, ambos os motores são colocados em rotação com sentido horário, a uma determinada velocidade, resultando no movimento contínuo do movimento representado na Figura 4.5-3.

MÉTODO MOVEBACK() : VOID

Tabela 4.5.3-28 Detalhes do método moveBack

+ moveBack () : void		
Nome do Método	**Descrição do método**	**O que retorna?**
moveBack	É enviada uma mensagem para o robot se mover para trás sem fim definido. Este irá se mover com velocidades default. Este movimento ira se prolongar ate que seja enviada uma mensagem em ordem do contrário.	Nada.

Este método não tem parâmetros.

Para a concretização deste movimento, ambos os motores são colocados em rotação com sentido anti-horário, a uma determinada velocidade, resultando no movimento contínuo do movimento representado na Figura 4.5-4.

MÉTODO STOP() : STRING

Tabela 4.5.3-29 Detalhes do método stop

+ stop () : String		
Nome do Método	**Descrição do método**	**O que retorna?**
stop	É enviada uma mensagem para o robot para parar todos os motores.	A resposta editada enviada pelo robot.

Este método não tem parâmetros.

Nesta funcionalidade todos os motores são parados, colocando os seus valores a zero.

MÉTODO MOVEFRONT(VELOCIDADEMOTORESQUERDO : INT, VELOCIDADEMOTORDIREITO : INT) : VOID

Tabela 4.5.3-30 Detalhes do método moveFront

+ moveFront (velocidadeMotorEsquerdo : int, velocidadeMotorDireito : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
moveFront	É enviada uma mensagem para o robot se mover em frente sem fim definido. Este irá se mover com velocidades definidas pelo utilizador. Este movimento ira se prolongar ate que seja enviada uma mensagem em ordem do contrário.	Nada.

Este método tem como parâmetros:

Tabela 4.5.3-31 Parâmetros do método moveFront

Nome do parâmetro	Tipo	Descrição
velocidadeMotorEsquerdo	Int	Velocidade que se ira mover o motor esquerdo. Tem de estar entre 0 e 255.
velocidadeMotorDireito	int	Velocidade que se ira mover o motor direito. Tem de estar entre 0 e 255.

Para a implementação desta funcionalidade no robot, ambos os motores são colocados em rotação com sentido horário, com a velocidade enviada no método pelo utilizador. O movimento executado pelo robot é semelhante ao representado na Figura 4.5-3.

Método moveFrontLevel(levelMotorEsquerdo : int, levelMotorDireito : int) : void

Tabela 4.5.3-32 Detalhes do método moveFront

+ moveFrontLevel(velocidadeMotorEsquerdo : int, velocidadeMotorDireito : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
moveFrontLevel	É enviada uma mensagem para o robot se mover em frente sem parar. Este ira se mover com velocidades correspondentes aos níveis definidos pelo utilizador. Este movimento ira se prolongar ate que seja enviada uma mensagem em ordem do contrário.	Nada.

Este método tem como parâmetros:

Tabela 4.5.3-33 Parâmetros do método moveFront

Nome do parâmetro	Tipo	Descrição
levelMotorEsquerdo	int	Nível que se ira mover o motor esquerdo. Tem de estar entre 0 e 5.
levelMotorDireito	int	Nível que se ira mover o motor esquerdo. Tem de estar entre 0 e 5.

Para a implementação desta funcionalidade no robot, ambos os motores são colocados em rotação com sentido horário, com a velocidade correspondente aos níveis enviados no método pelo utilizador. O movimento executado pelo robot é semelhante ao representado na Figura 4.5-3.

MÉTODO MOVEBACK(VELOCIDADEMOTORESQUERDO: INT, VELOCIDADEMOTORDIREITO : INT) : VOID

Tabela 4.5.3-34 Detalhes do método moveBack

+ moveBack (velocidadeMotorEsquerdo : int, velocidadeMotorDireito : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
moveBack	É enviada uma mensagem para o robot se mover para trás sem fim definido. Este irá se mover com velocidades definidas pelo utilizador. Este movimento ira se prolongar ate que seja enviada uma mensagem em ordem do contrário.	Nada.

Este método tem como parâmetros:

Tabela 4.5.3-35 Parâmetros do método moveBack

Nome do parâmetro	Tipo	Descrição
velocidadeMotorEsquerdo	Int	Velocidade que se ira mover o motor esquerdo. Tem de estar entre 0 e 255.
velocidadeMotorDireito	int	Velocidade que se ira mover o motor direito. Tem de estar entre 0 e 255.

Para a implementação desta funcionalidade no robot, ambos os motores são colocados em rotação com sentido anti-horário, com a velocidade enviada no método pelo utilizador. O movimento executado pelo robot é semelhante ao representado na Figura 4.5-4.

MÉTODO MOVEBACKLEVEL(LEVELMOTORESQUERDO : INT, LEVELMOTORDIREITO : INT) : VOID

Tabela 4.5.3-36 Detalhes do método moveFront

+ moveBackLevel(velocidadeMotorEsquerdo : int, velocidadeMotorDireito : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
moveBackLevel	É enviada uma mensagem para o robot se mover para trás sem parar. Este ira se mover com velocidades correspondentes aos níveis definidos pelo utilizador. Este movimento ira se prolongar ate que seja enviada uma mensagem em ordem do contrário.	Nada.

Este método tem como parâmetros:

Tabela 4.5.3-37 Parâmetros do método moveFront

Nome do parâmetro	Tipo	Descrição
levelMotorEsquerdo	int	Nível que se ira mover o motor esquerdo. Tem de estar entre 0 e 5.
levelMotorDireito	int	Nível que se ira mover o motor esquerdo. Tem de estar entre 0 e 5.

Para a implementação desta funcionalidade no robot, ambos os motores são colocados em rotação com sentido anti-horário, com a velocidade correspondente aos níveis enviados no método pelo utilizador. O movimento executado pelo robot é semelhante ao representado na Figura 4.5-4.

MÉTODO MOVE(VELOCIDADEMOTORESQUERDOFRENTE: INT, VELOCIDADEMOTORESQUERDOTRAS: INT, VELOCIDADEMOTORDIREITOFRENTE: INT, VELOCIDADEMOTORDIREITOTRAS: INT) : VOID

Tabela 4.5.3-38 Detalhes do método move

+ move (velocidadeMotorEsquerdoFrente: int, velocidadeMotorEsquerdoTras: int, velocidadeMotorDireitoFrente: int, velocidadeMotorDireitoTras: int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
move	É enviada uma mensagem para o robot se mover em uma direção definida pelo utilizador e sem fim definido. Apenas uma das velocidades de cada motor pode ser positiva, isto é, se a velocidadeMotorEsquerdoFrente for positiva a velocidadeMotorEsquerdoTras tem de ser nula, e vice-versa. Pelo menos um dos lados dos motores tem de ser positivo. O robot irá se mover com velocidades definidas pelo utilizador. Este movimento irá prolongar-se ate que seja enviada uma mensagem em ordem do contrário.	Nada.

Este método tem como parâmetros:

Tabela 4.5.3-39 Parâmetros do método move

Nome do parâmetro	Tipo	Descrição
velocidadeMotorEsquerdoFrente	int	Velocidade que se ira mover o motor esquerdo para a frente. Tem de estar entre 0 e 255.
velocidadeMotorEsquerdoTras	Int	Velocidade que se ira mover o motor esquerdo para trás. Tem de estar entre 0 e 255.
velocidadeMotorDireitoFrente	Int	Velocidade que se ira mover o motor direito para a frente. Tem de estar entre 0 e 255.
velocidadeMotorDireitoTras	int	Velocidade que se ira mover o motor direito para trás. Tem de estar entre 0 e 255.

Para a implementação desta funcionalidade, os motores são colocados em movimento consoante os valores enviados pelo utilizador, podendo o movimento resultante do robot variar de caso para caso.

4.5.4. *Controlo das estruturas móveis do robot*

Para aumentar a interatividade do robot com o meio ambiente, este possui um motor servo que suporta uma pequena plataforma com todos os sensores do robot. Para poder movimentar essa plataforma foram implementados métodos que manipulam a posição do motor servo, que se encontra mencionado em baixo:

MÉTODO LOOKTOANGLE (ANGULO: INT, DELAY: INT) : STRING

Tabela 4.5.4-1 Detalhes do método lookToAngle

+ lookToAngle (angulo: int, delay: int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
lookToAngle	Move o servo para um determinado ângulo durante um determinado período de tempo.	Resposta do robot editada, que informa sobre o novo ângulo do servo.

Este método tem como parâmetros:

Tabela 4.5.4-2 Parâmetros do método lookToAngle

Nome do parâmetro	Tipo	Descrição
ângulo	int	Destino do servo. Tem de estar entre 0 e 255.
delay	Int	Tempo que o servo demora a chegar ao destino. Tem de estar entre 0 e 10000 milisegundos.

Esta função no robot coloca o motor servo num angulo definido pelo utilizador, e é passado também um valor de *delay* para aguardar que o motor alcance o ângulo definido.

MÉTODO GETSERVOANGLE() : INT

Tabela 4.5.4-3 Detalhes do método getServoAngle

Y		
Nome do Método	**Descrição do método**	**O que retorna?**
getServoAngle	Devolve o angulo atual do motor servo.	Angulo atual do motor servo.

Este método não tem parâmetros.

Nesta função é lido o valor do angulo do servo, sendo posteriormente enviado o seu valor do robot para o computador.

Para a realização de algoritmos mais complexos, surgiu a necessidade de criar uma funcionalidade que fizesse o servo entrar em “estado de alerta” e movimentar-se continuamente de um angulo inicial para um angulo final e inverter esse movimento, até que seja enviada uma mensagem para terminar o movimento.

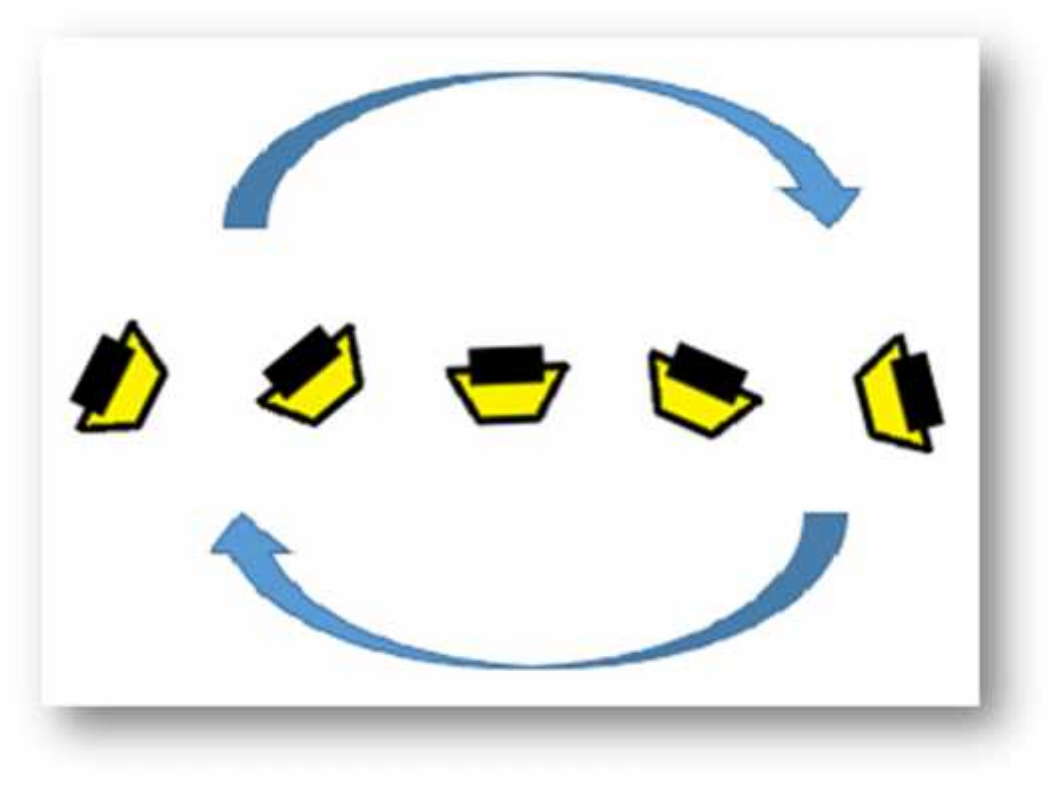


Figura 4.5-11 Representação do movimento servoRange

MÉTODO SERVORANGEON () : VOID

Tabela 4.5.4-4 Detalhes do método servoRangeOn

Y		
Nome do Método	**Descrição do método**	**O que retorna?**
servoRangeOn	Envia uma mensagem para o robot que aciona a função servoRange(). Esta função permite que o motor servo se mova entre um angulo mínimo e um angulo máximo, com um determinado incremento durante um determinado delay. Todos estes valores são valores definidos por default.	Nada.

Este método não tem parâmetros.

O movimento executado pelo servo é semelhante ao representado na figura 4.5-11.

MÉTODO SERVORANGEON (ANGULOMINIMO: INT, ANGULOMAXIMO: INT, DELAY : INT, INCREMENTO : INT) : VOID

Tabela 4.5.4-5 Detalhes do método servoRangeOn

+ servoRangeOn (anguloMinimo: int, anguloMaximo: int, delay : int, incremento : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
servoRangeOn	Envia uma mensagem para o robot que aciona a função servoRange(). Esta função permite que o motor servo se mova entre um angulo mínimo e um angulo máximo, com um determinado incremento durante um determinado delay. Todos estes valores são valores definidos pelo utilizador.	Nada.

Este método tem como parâmetros:

Tabela 4.5.4-6 Parâmetros do método servoRangeOn

Nome do parâmetro	Tipo	Descrição
anguloMinimo	int	Angulo mínimo do movimento do motor servo. Tem de estar entre 2 e 178 graus. Tem de ser menor que o anguloMaximo.
anguloMaximo	Int	Angulo máximo do movimento do motor servo. Tem de estar entre 2 e 178 graus. Tem de ser maior que o anguloMinimo.
delay	Int	Tempo de cada um dos incrementos do motor servo. Tem de estar entre 0 e 255 milisegundos.
incremento	int	Incremento de cada um dos passos do motor servo. Tem de ser positivo e ser menor ou igual a diferença entre o anguloMinimo e anguloMaximo.

O movimento executado pelo servo é semelhante ao representado na figura 4.5-11.

MÉTODO SERVORANGEOFF () : VOID

Tabela 4.5.4-7 Detalhes do método servoRangeOff

+ servoRangeOff () : void		
Nome do Método	**Descrição do método**	**O que retorna?**
servoRangeOff	Envia uma mensagem para o robot que termina o movimento da função servoRange().	Nada.

Este método não tem parâmetros.

Nesta funcionalidade o movimento do servo é terminado.

MÉTODO GETDELAYSERVORANGE () : INT

Tabela 4.5.4-8 Detalhes do método getDelayServoRange

+ getDelayServoRange () : int		
Nome do Método	**Descrição do método**	**O que retorna?**
getDelayServoRange	Devolve o delay atual da função servoRange().	Devolve o delay atual da função servoRange().

Este método não tem parâmetros.

Nesta função é lido o valor do delay da função servo range, sendo posteriormente enviado o seu valor do robot para o computador.

MÉTODO SETDELAYSERVORANGE (NOVODELAY: INT) : VOID

Tabela 4.5.4-9 Detalhes do método setDelayServoRange

+ setDelayServoRange (novoDelay: int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
setDelayServoRange	Altera o delay da função servoRange().	Nada.

Este método tem como parâmetros:

Tabela 4.5.4-10 Parâmetros do método setDelayServoRange

Nome do parâmetro	Tipo	Descrição
novoDelay	int	Novo delay da função servoRange(). Tem de estar entre 0 e 255 milisegundos.

Nesta funcionalidade o valor do delay da função servoRange é atualizado para o valor enviado.

4.5.5. *Controlo de outros atuadores*

Foi também incorporado um led RGB no robot, e sucessivamente criados métodos para manipular o led livremente. O estado de um Led on/off possui no arduíno o valor High e Low.

MÉTODO LEDREDON () : VOID

Tabela 4.5.5-1 Detalhes do método ledRedOn

+ ledRedOn () : void		
Nome do Método	**Descrição do método**	**O que retorna?**
ledRedOn	Envia uma mensagem para o robot que acende o led Vermelho.	Nada.

Este método não tem parâmetros.

MÉTODO LEDGREENON() : VOID

Tabela 4.5.5-2 Detalhes do método ledGreenOn

+ ledGreenOn () : void		
Nome do Método	**Descrição do método**	**O que retorna?**
ledGreenOn	Envia uma mensagem para o robot que acende o led Verde.	Nada.

Este método não tem parâmetros.

MÉTODO LEDBLUEON () : VOID

Tabela 4.5.5-3 Detalhes do método ledBlueOn

+ ledBlueOn () : void		
Nome do Método	**Descrição do método**	**O que retorna?**
ledBlueOn	Envia uma mensagem para o robot que acende o led Azul.	Nada.

Este método não tem parâmetros.

MÉTODO LEDREDOFF () : VOID

Tabela 4.5.5-4 Detalhes do método ledRedOff

+ ledRedOff () : void		
Nome do Método	**Descrição do método**	**O que retorna?**
ledRedOff	Envia uma mensagem para o robot que apaga o led Vermelho.	Nada.

Este método não tem parâmetros.

Método ledGreenOff() : void

Tabela 4.5.5-5 Detalhes do método ledGreenOff

+ ledGreenOff () : void		
Nome do Método	**Descrição do método**	**O que retorna?**
ledGreenOff	Envia uma mensagem para o robot que apaga o led Verde.	Nada.

Este método não tem parâmetros.

Método ledBlueOff() : void

Tabela 4.5.5-6 Detalhes do método ledBlueOff

+ ledBlueOff() : void		
Nome do Método	**Descrição do método**	**O que retorna?**
ledBlueOff	Envia uma mensagem para o robot que apaga o led azul.	Nada.

Este método não tem parâmetros.

Método ledRedOn (delay : int) : void

Tabela 4.5.5-7 Detalhes do método ledRedOn

+ ledRedOn (delay : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
ledRedOn	Envia uma mensagem para o robot que acende o led Vermelho, durante um período de tempo determinado pelo utilizador.	Nada.

Este método tem como parâmetros:

Tabela 4.5.5-8 Parâmetros do método ledRedOn

Nome do parâmetro	Tipo	Descrição
delay	int	Tempo que o led permanece ligado. Tem de estar entre 0 e 10000 milisegundos.

MÉTODO LEDGREENON(DELAY : INT) : VOID

Tabela 4.5.5-9 Detalhes do método ledGreenOn

+ ledGreenOn (delay : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
ledGreenOn	Envia uma mensagem para o robot que acende o led Verde, durante um período de tempo determinado pelo utilizador.	Nada.

Este método tem como parâmetros:

Tabela 4.5.5-10 Parâmetros do método ledBlueOn

Nome do parâmetro	Tipo	Descrição
delay	int	Tempo que o led permanece ligado. Tem de estar entre 0 e 10000 milisegundos.

MÉTODO LEDBLUEON (DELAY : INT) : VOID

Tabela 4.5.5-11 Detalhes do método ledBlueOn

+ ledBlueOn(delay : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
ledBlueOn	Envia uma mensagem para o robot que acende o led Azul, durante um período de tempo determinado pelo utilizador.	Nada.

Este método tem como parâmetros:

Tabela 4.5.5-12 Parâmetros do método ledBlueOn

Nome do parâmetro	Tipo	Descrição
delay	int	Tempo que o led permanece ligado. Tem de estar entre 0 e 10000.

Assim, como um led RGB, também foi adicionado ao robot um buzzer para que este pudesse comunicar com o mundo exterior. Este *buzzer* permite se emitir uma frequência entre 0 e 65535 . Para manipular o *buzzer* foram criados 3 métodos, um para emitir uma frequência durante um determinado período de tempo definido pelo utilizador, outro para emitir uma frequência continuamente, e por fim um método para silenciar o buzzer.

Método buzz(frequência : int, delay : int) : void

Tabela 4.5.5-13 Detalhes do método buzz

+ buzz (frequência : int, delay : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
buzz	Emite uma frequência no buzzer durante um período de tempo, definidos pelo utilizador.	Nada.

Este método tem como parâmetros:

Tabela 4.5.5-14 Parâmetros do método buzz

Nome do parâmetro	Tipo	Descrição
frequência	int	Valor de referência que o buzzer emite. Tem de estar entre 0 e 65535 [27]hz.
delay	int	Tempo de emissão da frequência. Tem de estar entre 0 e 10000 milisegundos.

Método buzzOn(frequência : int) : void

Tabela 4.5.5-15 Detalhes do método buzzOn

+ buzzOn(frequência : int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
buzzOn	Emite uma frequência no buzzer sem limite de tempo definido.	Nada.

Este método tem como parâmetros:

Tabela 4.5.5-16 Parâmetros do método buzzOn

Nome do parâmetro	Tipo	Descrição
frequência	int	Valor de referência que o buzzer emite. Tem de estar entre 0 e 65535 hz.

Método buzzOff() : void

Tabela 4.5.5-17 Detalhes do método buzzOff

+ buzzOff() : void		
Nome do Método	**Descrição do método**	**O que retorna?**
buzzOff	Coloca o buzzer em silêncio.	Nada.

Este método não tem parâmetros.

[27] **Fonte**: https://code.google.com/p/rogue-code/wiki/ToneLibraryDocumentation#Ugly_Details

4.5.6. *Obtenção de informação sensorial*

Para obter informação sensorial sobre o que rodeia o robot, são utilizados diversos sensores. No caso do robot virtualizado, este possui dois sensores de pressão (um esquerdo e outro direito), um sensor Infravermelhos, um sensor de som e um sensor de luz.

Os dois bumpers constituem em duas “antenas” de contacto, que quando em contacto com um obstáculo possui no arduíno o valor *High*, sendo o seu estado de origem, isto é, sem obstáculo o valor *Low*.

Demonstramos um exemplo de deteção de um obstáculo pelo *bumper* esquerdo na Figura 4.5-12.

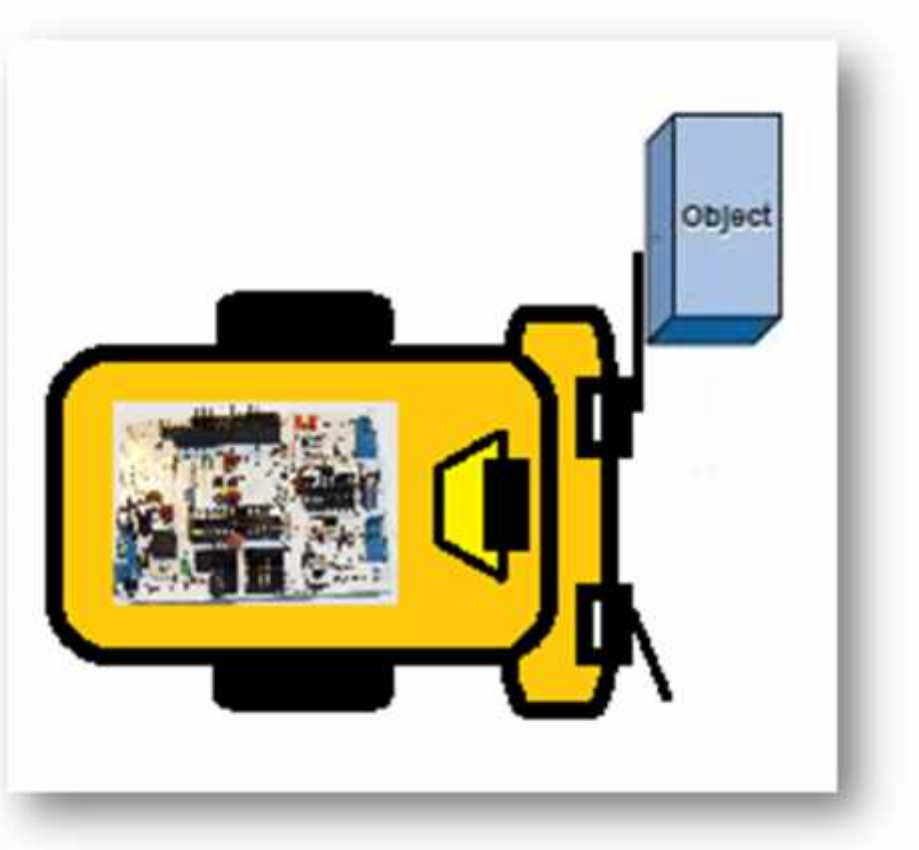


Figura 4.5-12 Representação gráfica de deteção de um obstáculo pelo *bumper* esquerdo

Para a leitura dos bumpers do robot foram criados os métodos que passamos a descrever:

MÉTODO READBUMPERSAFTERPRESSED() : STRING

Tabela 4.5.6-1 Detalhes do método readBumpersAfterPressed

+ readBumpersAfterPressed() : String		
Nome do Método	**Descrição do método**	**O que retorna?**
readBumpersAfterPressed	Retorna o valor dos bumpers direito e esquerdo após um deles ser pressionado, no formato de uma *string*.	A leitura dos bumpers, em formato de uma *string*.

Este método não tem parâmetros.

Para a implementação desta funcionalidade os valores dos bumpers são lidos a cada iteração do processador do arduíno, no entanto, os valores dos *bumpers* apenas é enviado ao computador após pelo menos um dos *bumpers* se encontrar pressionado.

MÉTODO READBUMPERSAFTERPRESSEDARRAY() : INT[]

Tabela 4.5.6-2 Detalhes do método readBumpersAfterPressedArray

+ readBumpersAfterPressedArray() : int[]		
Nome do Método	**Descrição do método**	**O que retorna?**
readBumpersAfterPressedArray	Retorna o valor dos bumpers direito e esquerdo após um deles ser pressionado, no formato de um array.	A leitura dos bumpers, em formato de um array.

Este método não tem parâmetros.

Esta funcionalidade funciona da mesma forma que no método readBumpersAfterPressed() : *String*, sendo neste caso, o valor dos *bumpers* enviado no formato de um *array* em vez de uma *string*.

MÉTODO READBUMPERS() : STRING

Tabela 4.5.6-3 Detalhes do método readBumpers

+ readBumpers () : String		
Nome do Método	**Descrição do método**	**O que retorna?**
readBumpers	Retorna o valor dos bumpers direito e esquerdo no momento do pedido, no formato de uma string.	O valor dos bumpers, em formato de uma string.

Este método não tem parâmetros.

Nesta funcionalidade os valores dos bumpers é lido e retornado logo após a sua leitura.

MÉTODO READBUMPERSARRAY() : INT[]

Tabela 4.5.6-4 Detalhes do método readBumpersArray

+ readBumpersArray() : int[]		
Nome do Método	**Descrição do método**	**O que retorna?**
readBumpersArray	Retorna o valor dos bumpers direito e esquerdo no momento do pedido, no formato de um array.	O valor dos bumpers, em formato de um array.

Este método não tem parâmetros.

Esta funcionalidade funciona de forma semelhante ao método readBumpers() : *String*, sendo neste caso, o valor dos *bumpers* enviado no formato de um a*rray* em vez de uma *string.*

MÉTODO ISBUMPERLEFTPRESSED() : BOOLEAN

Tabela 4.5.6-5 Detalhes do método isBumperLeftPressed

+ isBumperLeftPressed () : Boolean		
Nome do Método	**Descrição do método**	**O que retorna?**
isBumperLeftPressed	Verifica se o bumper esquerdo está pressionado.	Devolve True ou false, consoante o estado do bumper.

Este método não tem parâmetros.

Nesta funcionalidade os valores dos bumpers é lido e enviado para o computador. Após a leitura dos *bumpers*, é devolvido *true* ou *false* consoante o estado do b*umper* esquerdo, sendo *true* caso o *bumper* esteja *high*, e *false* caso esteja *low*.

MÉTODO ISBUMPERRIGHTPRESSED() : BOOLEAN

Tabela 4.5.6-6 Detalhes do método isBumperRightPressed

+ isBumperRightPressed () : Boolean		
Nome do Método	**Descrição do método**	**O que retorna?**
isBumperRightPressed	Verifica se o bumper direito está pressionado.	Devolve True ou false, consoante o estado do bumper.

Este método não tem parâmetros.

Este método funciona de forma semelhante ao metodo isBumperRightPressed() : *Boolean*. Após a leitura dos *bumpers*, é também devolvido *true* ou *false* consoante o estado do *bumper* direito.

MÉTODO GETBUMPERLEFTVALUE() : BOOLEAN

Tabela 4.5.6-7 Detalhes do método getBumperLeftValue

+ getBumperLeftValue () : Boolean		
Nome do Método	**Descrição do método**	**O que retorna?**
getBumperLeftValue	Devolve o estado do bumper esquerdo. Para uma leitura correta é necessário correr o método readBumpers ou readBumpersAfterPressed.	Devolve True ou false, consoante o estado do bumper.

Este método não tem parâmetros.

Para a implementação desta funcionalidade, foram adicionadas duas variáveis (leftBumperValue e rightBumperValue) para guardar os valores de ambos os bumpers.

Para que o valor dessas variáveis seja atualizado é necessário correr previamente um dos métodos: readBumpersAfterPressed() : String, readBumpersAfterPressedArray() : int[], readBumpers() : String, readBumpersArray() : int[]. Cada um destes métodos atualiza os valores das variáveis leftBumperValue e rightBumperValue.

Quando este método é executado, é devolvido *true* ou *false* consoante o estado da variável, sendo *true* quando a variável leftBumperValue é igual a 1 e *false* quando é igual a 0.

MÉTODO GETBUMPERRIGHTVALUE() : BOOLEAN

Tabela 4.5.6-8 Detalhes do método getBumperRightValue

+ getBumperRightValue () : Boolean		
Nome do Método	**Descrição do método**	**O que retorna?**
getBumperRightValue	Devolve o estado do bumper direito. Para uma leitura correta é necessário correr o método readBumpers ou readBumpersAfterPressed.	Devolve True ou false, consoante o estado do bumper.

Este método não tem parâmetros.

Esta funcionalidade tem um funcionamento igual ao do método getBumperLeftValue() : Boolean, sendo devolvido *true* ou *false*, consoante o valor da variável rightBumperValue.

Como já referido, o robot contém um sensor de Infravermelhos, este sensor é utilizado para deteção de obstáculos a distância. Existem sensores de infravermelho ativos e passivos[28]. Um sensor de infravermelho ativo é composto por um emissor de luz infravermelha e um recetor, que reage a essa luz. Por sua vez, um sensor de infravermelho passivo não emite luz infravermelha, mas apenas capta esse tipo de luz no ambiente. No caso do robot virtualizado, este utiliza um sensor de infravermelhos ativo.

Apresentamos um exemplo de deteção de um obstáculo pelo sensor de IR do robot na Figura 4.5-13.

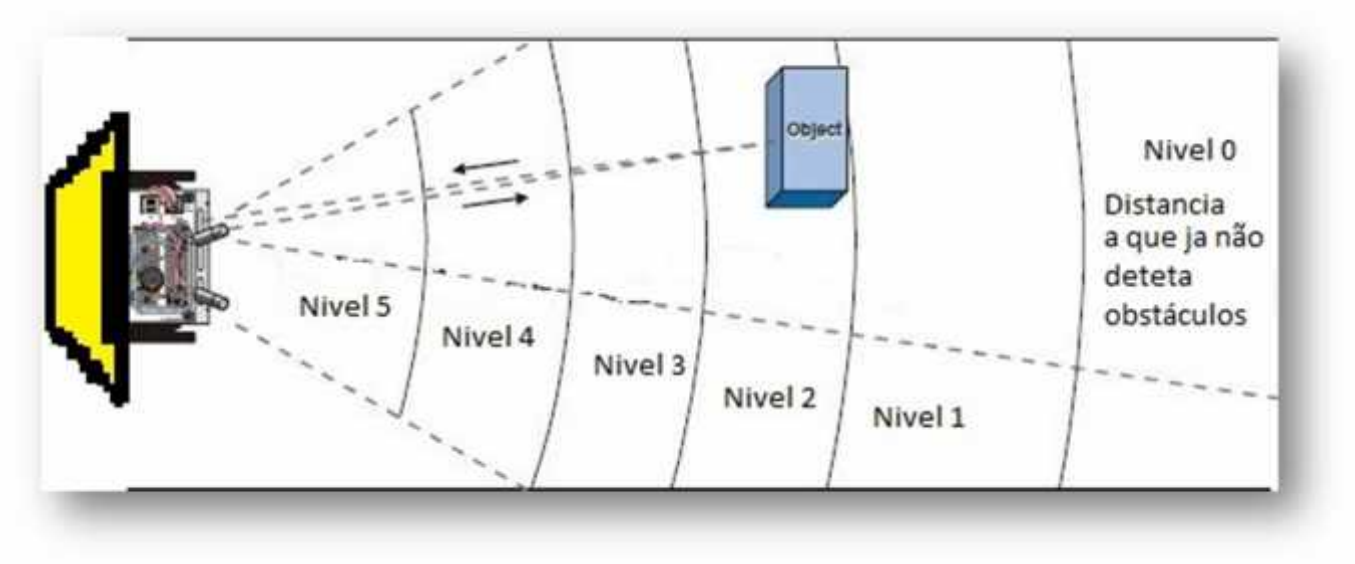


Figura 4.5-13 Representação gráfica do sensor IR.

MÉTODO READINFRARED () : INT

Tabela 4.5.6-9 Detalhes do método readInfraRed

+ readInfraRed () : int		
Nome do Método	**Descrição do método**	**O que retorna?**
readInfraRed	Devolve a leitura do sensor Infravermelhos (IR).	Leitura do sensor Infravermelhos (IR).

Este método não tem parâmetros.

Nesta funcionalidade o valor do sensor de IR é lido e enviado o resultado para o computador.

[28] ***Fontes***: http://www.engineersgarage.com/articles/what-is-passive-infrared-pir-sensor?page=2 e http://robolivre.org/conteudo/sensor-de-infravermelho

MÉTODO INFRAREDLEVEL () : INT

Tabela 4.5.6-10 Detalhes do método infraRedLevel

+ infraRedLevel () : int		
Nome do Método	**Descrição do método**	**O que retorna?**
infraRedLevel	Devolve o nível do sensor Infra-Vermelhos (IR).	Nível do sensor Infravermelhos (IR).

Este método não tem parâmetros.

Nesta funcionalidade o valor do sensor de IR é lido, tratado e devolvido o seu nível correspondente.

MÉTODO ISINFRAREDVALUEOVER (VALOR: INT) : VOID

Tabela 4.5.6-11 Detalhes do método isInfraRedValueOver

+ isInfraRedValueOver (valor: int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
isInfraRedValueOver	Verifica se o valor do sensor de Infravermelhos (IR) é superior ao valor recebido do utilizador como parâmetro.	Devolve True ou false, consoante o resultado da comparação.

Este método tem como parâmetros:

Tabela 4.5.6-12 Parâmetros do método isInfraRedValueOver

Nome do parâmetro	Tipo	Descrição
valor	int	Valor de referência para a comparação.

Nesta funcionalidade, o valor do sensor é lido e comparado com o valor recebido como parâmetro. No fim é devolvido *true* ou *false*, consoante se o valor recebido é superior ou inferior ao valor lido do sensor.

Existe também disponível um sensor de som analógico. Um sensor de som analógico é normalmente utilizado para a detenção de volume no ambiente. Através do arduíno é possível obter o valor do sinal de saída e calcular a intensidade do som[29].

O objetivo da utilização deste sensor era a deteção de som proveniente de palmas, fala ou gritos.

Figura 4.5-14 Robot capta som de palmas através do sensor de som.

Um sensor de som (microfone) produz sinais elétricos muito pequenos. Se conectado diretamente a um pin analógico do arduíno, não se ria obter qualquer alteração significativa. Logo, o sinal precisa de ser amplificado primeiro. O sensor de som utilizado neste projeto em para esse efeito um amplificador opamp 100x, que amplifica o sinal o suficiente para o este ser captado pelo arduíno.

[29] ***Fonte***: *http://www.dfrobot.com/wiki/index.php/Analog_Sound_Sensor_SKU:_DFR0034*

Para uma correta leitura [30]de um sinal de áudio é primeiro necessário efetuar-se alguns cálculos adicionais. Um sinal de áudio é representado por uma onda estando em constante mudança, logo o valor que é retornado pelo *analoRead()* [31] no arduíno vai depender do ponto da onda em que é feita a leitura. Um exemplo de uma onda de um sinal de áudio é representado na Figura 4.5-15. Ao longo do tempo da esquerda para a direita, a voltagem sobe e desce com um padrão regular. Se forem feitas leituras nos três locais marcados, todos em tempos diferentes, iremos ter três valores diferentes. Se usamos estas leituras para tomar decisões, poderemos concluir incorretamente que a intensidade (volume) do sinal é superior no meio (ponto numero 2).

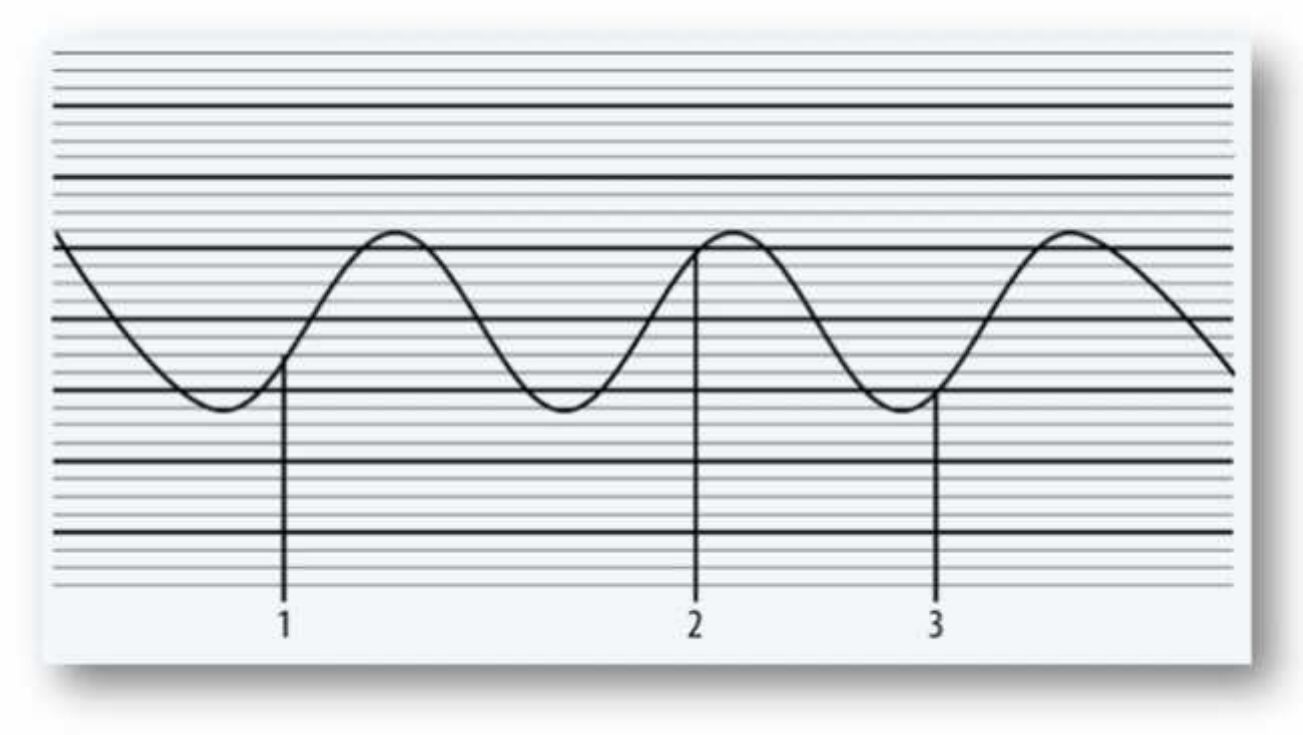


Figura 4.5-15 Sinal de áudio lido em três sítios.

Para uma leitura correta e precisa é necessário fazermos múltiplas leituras o mais próximo possível. Os altos e baixos da onda aumentam a medida que o sinal fica maior. A diferença entre um ponto superior e inferior é designada de amplitude do sinal, e esta aumenta a medida que o sinal fica mais alto, ou seja, aumenta a sua intensidade (volume).

Para se obter a dimensão dos altos e baixos, faz-se a diferença entre o ponto central da contagem e os níveis dos altos e baixos. O ponto central pode ser visualizado na Figura 4.5-16, na forma de uma linha que esta traçada entre o ponto mais alto e o ponto mais baixo da onda. A linha representa o DC *offset* do sinal (*DC offset* – é o valor do sinal sem altos e baixos). Se o valor do *DC offset* for subtraído

[30] ***Fonte***: https://www.inkling.com/read/arduino-cookbook-michael-margolis-2nd/chapter-6/recipe-6-7

[31] analogRead() - Lê um valor especifico de um pin analógico. ***Fonte***: *http://arduino.cc/en/Reference/analogRead*

ao valor obtido da função *analogRead()*, iremos obter o valor correto da leitura da amplitude do sinal.

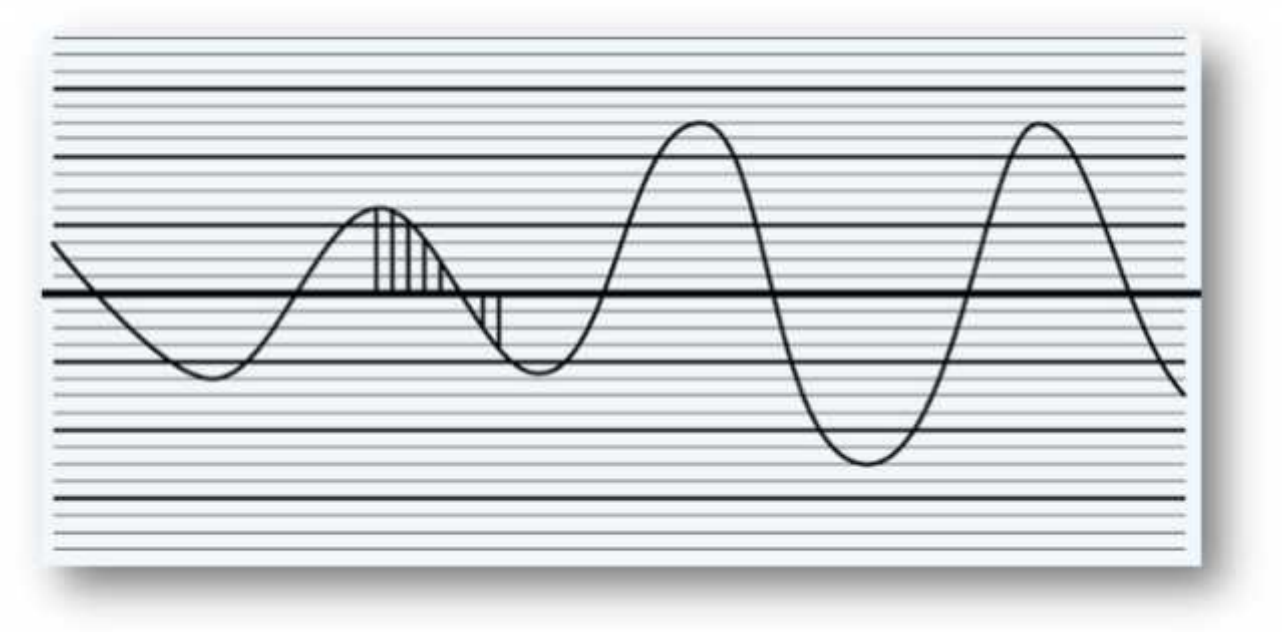


Figura 4.5-16 Ponto central do sinal de áudio

A medida que o sinal aumenta a sua intensidade, a média desses valores também irá aumentar, mas como alguns desses valores são negativos (onde o sinal era inferior ao valor do DC offset), os valores vão se anular e a média será igual a zero. De modo a corrigir isso é feito o quadrado dos valores (valor multiplicado por si próprio). Desta forma teremos todos os valores positivos, e irá aumentar a diferença entre as pequenas alterações do sinal, o que ajuda a analisar melhor o sinal. O valor da média irá aumentar ou diminuir como a amplitude do sinal faz.

Para os cálculos serem corretos é necessário saber-se o valor do DC offset. Para obtermos um sinal limpo, o amplificador do sinal é desenvolvido para ter um DC offset o mais próximo possível ao meio do intervalo da voltagem (sendo o intervalo entre 0 e 1023, assume-se o DC offset a 512), para que o sinal possa ser amplificado o máximo possível sem que haja distorção do sinal.

MÉTODO READSOUND () : INT

Tabela 4.5.6-13 Detalhes do método readSound

+ readSound() : int		
Nome do Método	**Descrição do método**	**O que retorna?**
readSound	Devolve a leitura do sensor de som.	Leitura do sensor de som.

Este método não tem parâmetros.

Nesta funcionalidade o valor do sensor de Som é lido, é feito o tratamento do sinal e todos os cálculos necessários e enviado o resultado para o computador.

MÉTODO SOUNDLEVEL () : INT

Tabela 4.5.6-14 Detalhes do método soundLevel

+ soundLevel() : int		
Nome do Método	**Descrição do método**	**O que retorna?**
soundLevel	Devolve o nível do sensor de som.	Nível do sensor de som.

Este método não tem parâmetros.

Nesta funcionalidade é feita uma leitura do método “readSound () : int” e devolvido o seu nível correspondente.

MÉTODO ISSOUNDVALUEOVER (VALOR: INT) : VOID

Tabela 4.5.6-15 Detalhes do método isSoundValueOver

+ isSoundValueOver (valor: int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
isSoundValueOver	Verifica se o valor do sensor de Som é superior ao valor recebido do utilizador como parâmetro.	Devolve True ou false, consoante o resultado da comparação.

Este método tem como parâmetros:

Tabela 4.5.6-16 Parâmetros do método isSoundValueOver

Nome do parâmetro	Tipo	Descrição
valor	int	Valor de referência para a comparação.

Nesta funcionalidade, o valor do sensor é lido e comparado com o valor recebido como parâmetro. No fim é devolvido true ou false, consoante se o valor recebido é superior ou inferior ao valor lido do sensor.

Foi também adicionado um sensor de Luz [32] ao robot. Desta forma é possível ao robot detetar diferentes níveis de intensidade de luz no ambiente. Um sensor de luz é um dispositivo que converte energia em forma de luz em energia elétrica. Neste sensor não existe a necessidade de se realizarem cálculos como no caso do sensor de som, bastando uma leitura do pin analógico. O sensor utilizado no robot é um sensor fotovoltaico. Estes sensores mudam a sua resistência dependendo da intensidade de luz a que são expostos.

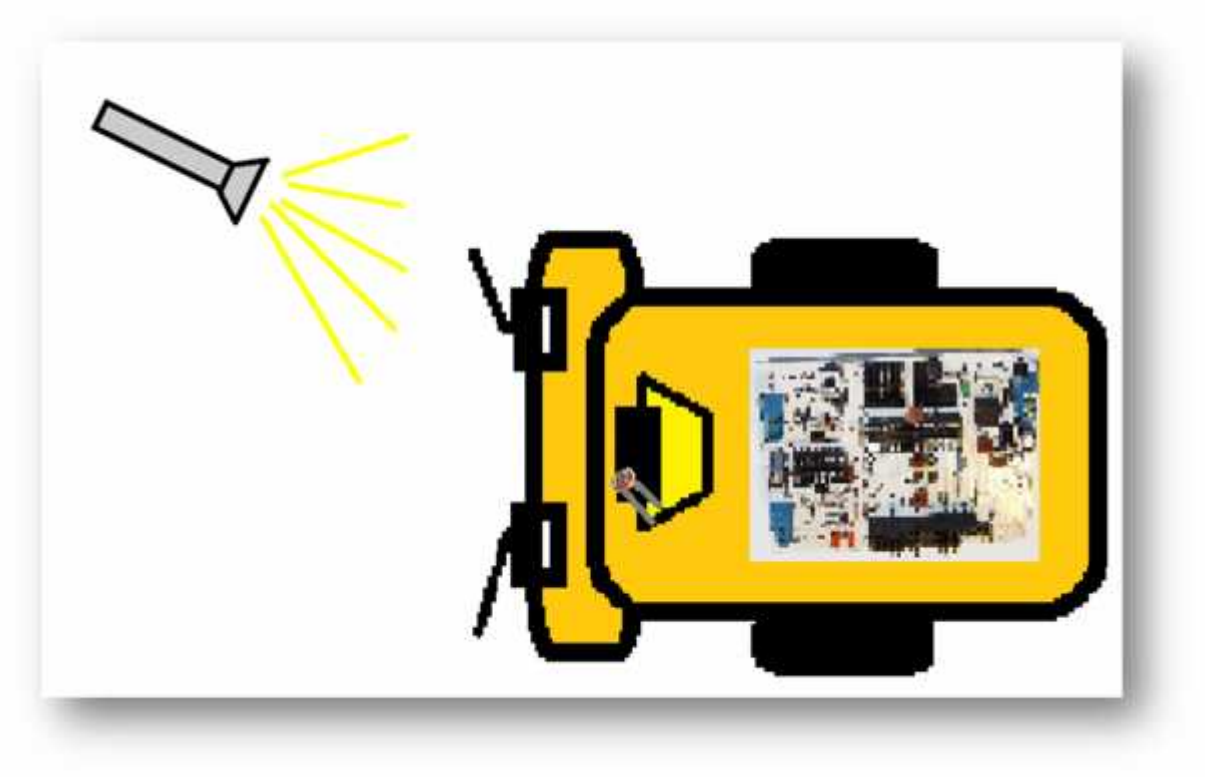


Figura 4.5-17 Robot capta a Luz através do sensor de Luz.

MÉTODO READLIGHT() : INT

Tabela 4.5.6-17 Detalhes do método readLight

+ readLight () : int		
Nome do Método	**Descrição do método**	**O que retorna?**
readLight	Devolve a leitura do sensor de luz.	Leitura do sensor de luz.

Este método não tem parâmetros.

Nesta funcionalidade o valor do sensor de Luz é lido e enviado o resultado para o computador.

[32] ***Fonte***: *https://www.sparkfun.com/products/9088*

MÉTODO LIGHTLEVEL () : INT

Tabela 4.5.6-18 Detalhes do método lightLevel

+ lightLevel() : int		
Nome do Método	**Descrição do método**	**O que retorna?**
lightLevel	Devolve o nível do sensor de luz.	Nível do sensor de luz.

Este método não tem parâmetros.

Nesta funcionalidade o valor do sensor de Luz é lido, tratado e devolvido o seu nível correspondente.

MÉTODO ISLIGHTVALUEOVER (VALOR: INT) : VOID

Tabela 4.5.6-19 Detalhes do método isLightValueOver

+ isLightValueOver (valor: int) : void		
Nome do Método	**Descrição do método**	**O que retorna?**
isLightValueOver	Verifica se o valor do sensor de Luz é superior ao valor recebido do utilizador como parâmetro.	Devolve True ou false, consoante o resultado da comparação.

Este método tem como parâmetros:

Tabela 4.5.6-20 Parâmetros do método isLightValueOver

Nome do parâmetro	**Tipo**	**Descrição**
valor	int	Valor de referência para a comparação.

Nesta funcionalidade, o valor do sensor é lido e comparado com o valor recebido como parâmetro. No fim é devolvido true ou false, consoante se o valor recebido é superior ou inferior ao valor lido do sensor.

Por fim, foram também criados dois métodos que possibilitasse a leitura de todos os valores de todos os sensores e do angulo do servo, numa única leitura.

MÉTODO GETALLVALUES() : INT

Tabela 4.5.6-21 Detalhes do método getAllValues

+ getAllValues () : int		
Nome do Método	**Descrição do método**	**O que retorna?**
getAllValues	Devolve os valores de todos os sensores no formato de uma string. (Valor do Bumper Direito, Valor do Bumper Esquerdo, Valor do Sensor de Infravermelhos (IR), Angulo Servo, Valor do Sensor de Luz, Valor do Sensor de Som)	Os valores de todos os sensores no formato de uma string.

Este método não tem parâmetros.

Nesta funcionalidade, são lidos todos os valores de todos os sensores e o valor do angulo do servo. Esses valores são concatenados numa string e essa string é enviada para o computador. Essa string contém todos os valores separados por um #.

MÉTODO GETALLVALUESARRAY() : INT

Tabela 4.5.6-22 Detalhes do método getAllValuesArray

+ getAllValuesArray () : int		
Nome do Método	**Descrição do método**	**O que retorna?**
getAllValuesArray	Devolve os valores de todos os sensores no formato de um array. (Valor do Bumper Direito, Valor do Bumper Esquerdo, Sensor de Infravermelhos (IR), Angulo Servo, Sensor de Luz, Sensor de Som).	Os valores de todos os sensores no formato de um array

Este método não tem parâmetros.

Esta funcionalidade recebe a mesma string que o método getAllValues () : int, no entanto coloca todos esses valores dentro de um array e devolve esse array.

4.6. Modelo de código da Biblioteca BSAA.jar

Demonstramos na Figura 4.6-1 o modelo de codigo criado para a utilização das funções da biblioteca de abstração BSAA.jar

```
package BSAA;

//import para utilizar as funcoes do robot
import Robot.Robot;

/**
 * Criado por:
 * Data:
 * Este ficheiro é um template inicial para a utilização do robot
 */

public class Template {

    public static void main(String[] args) {
        // Instancializar o objeto do robot
        Robot meuRobot = new Robot();

        // Inserir codigo aqui!!!

        //Terminar sempre o programa com meuRobot.TerminarComunicacao();
        meuRobot.terminarComunicacao();
    }
}
```

Figura 4.6-1 Modelo de codigo Biblioteca *BSAA.jar*

Após se adicionar a biblioteca de abstração ao projeto, para se utilizar as funções definidas na bibliotecas basta usar-se o modelo de código apresentado na Figura 4.6-1. Neste modelo de código encontra-se o *import* da classe Robot que permite a utilização das funções existentes na classe robot. No método *main* é necessário inicializarmos um objeto do tipo Robot e no fim das instruções temos obrigatoriamente de terminar a comunicação com o robot.

4.7. Controlo do Robot no Arduíno

Todas as funções demonstradas na secção 4.5, de modo a executarem alguma ação, foi necessário programa-las todas diretamente no arduíno. Demonstramos o diagrama de classes do código arduíno na Figura 4.7-1.

Figura 4.7-1 Diagrama de classes do código arduíno

4.7.1. *EEPROM*

Como já referido anteriormente, algumas funções possuem valores *default*. Para que esses valores possam ser guardados e calibrados para cada robot, estes foram guardados na *EEPROM* [33] do robot, assim como os valores das velocidades dos motores, e pins de cada um dos componentes do robot. Para tal, foi criado um script especificamente para a criação desses valores na *EEPROM*, e posteriormente criados métodos específicos para ler e escrever esses valores da *EEPROM*.

SCRIPT PARA CRIAR VALORES NA EEPROM

Para a criação deste script é necessário compreender a utilização e funcionamento da *EEPROM*.

Uma *EEPROM* (de *Electrically-Erasable Programmable Read-Only Memory*) é um tipo de memória que pode armazenar valores que serão retidos mesmo quando a energia é desligada e pode ser programada e apagada várias vezes, eletricamente. Pode ser lida um número ilimitado de vezes, mas só pode ser apagada e programada, um número limitado de vezes, que varia normalmente entre 100.000 e 1 milhão.

A quantidade de memória EEPROM presente em um Arduíno varia conforme o microcontrolador instalado na placa: 1024 bytes para o ATmega328, 512 bytes no ATmega168 e ATmega8, e 4 KB (4096 bytes) sobre o ATmega1280 e ATmega2560.

Para usar esta função basta incluir a biblioteca no início do código desta forma: **#include <EEPROM.h>**

Uma vez que a biblioteca é incluída no programa, um objeto EEPROM está disponível para o acesso a memória. A biblioteca fornece comandos para ler e escrever dados na memória. A biblioteca EEPROM requer que seja especificado o endereço de memória que se deseja ler ou escrever. Isto significa que é preciso se manter a par de onde cada valor é escrito de forma que quando se for ler o valor, o acesso seja feito a partir do endereço correto.

Para **escrever** [34] um valor na memória, é utilizado: **EEPROM.write(address, value);**
Onde:

address – posição da memória que será escrito, é um inteiro entre 0 e 1023 (UNO);
value - valor a ser armazenado inteiro entre 0 e 255 (um único byte).

---

[33] ***Fonte***: http://arduino.cc/en/Reference/EEPROM

[34] ***Fonte***: http://arduino.cc/en/Reference/EEPROMWrite

Para **ler** [35] uma determinada posição de memória: **value = EEPROM.read(address);**

Onde:

address – posição da memória que será lido, é um inteiro entre 0 e 1023 (UNO);

value - valor do endereço da EEPROM é um inteiro entre 0 e 255 (um único byte).

No entanto esta formula apenas funciona para valores entre 0 e 255, para guardar valores superiores [36]é necessário fazer-se uma conversão de valores de 16 ou de 32 bits em bytes.

```
loByte = lowByte(val);
hiByte = highByte(val);
```

Onde:

val - qualquer tipo de variável

loByte - byte com a parte mais baixa de val

hiByte - byte com a parte mais alta de val

Para se poder recuperar os valores guardados na EEPROM, é utilizada a função word [37]que converte dois bytes em um inteiro de 16bits.

```
value = word(x)
value = word(h, l)
```

Onde:

value - uma word

x - uma variável de qualquer tipo

h - a parte alta de uma word

l - a parte baixa de uma word

Com isto, passamos a descrever o script da EEPROM.

DECLARAÇÃO DAS VARIÁVEIS

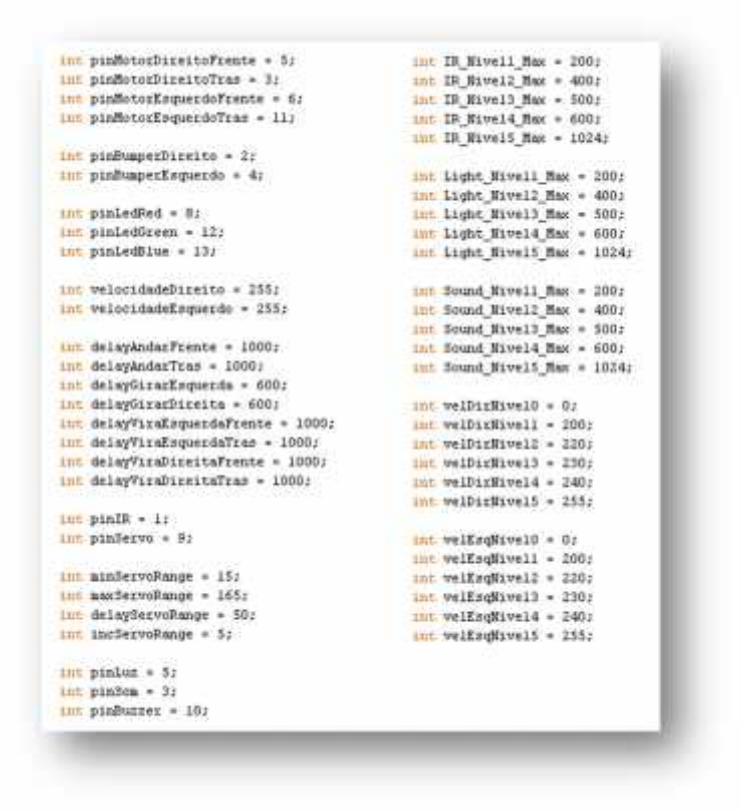
Inicialmente foram declaradas todas as variáveis que vão ser guardadas na EEPROM. Apresentamos essas variáveis na Figura 4.7-2.

---

[35] ***Fonte***: http://arduino.cc/en/Reference/EEPROMRead

[36] ***Fontes***: http://arduino.cc/en/Reference/LowByte e http://arduino.cc/en/Reference/HighByte

[37] ***Fonte***: http://arduino.cc/en/Reference/WordCast



```
int pinMotorDireitoFrente = 5;
int pinMotorDireitoTras = 3;
int pinMotorEsquerdoFrente = 6;
int pinMotorEsquerdoTras = 11;

int pinBumperDireito = 2;
int pinBumperEsquerdo = 4;

int pinLedRed = 8;
int pinLedGreen = 12;
int pinLedBlue = 13;

int velocidadeDireito = 255;
int velocidadeEsquerdo = 255;

int delayAndarFrente = 1000;
int delayAndarTras = 1000;
int delayGirarEsquerda = 600;
int delayGirarDireita = 600;
int delayViraEsquerdaFrente = 1000;
int delayViraEsquerdaTras = 1000;
int delayViraDireitaFrente = 1000;
int delayViraDireitaTras = 1000;

int pinIR = 1;
int pinServo = 9;

int minServoRange = 15;
int maxServoRange = 165;
int delayServoRange = 50;
int incServoRange = 5;

int pinLux = 5;
int pinSom = 3;
int pinBuzzer = 10;

int IR_Nivel1_Max = 200;
int IR_Nivel2_Max = 400;
int IR_Nivel3_Max = 500;
int IR_Nivel4_Max = 600;
int IR_Nivel5_Max = 1024;

int Light_Nivel1_Max = 200;
int Light_Nivel2_Max = 400;
int Light_Nivel3_Max = 500;
int Light_Nivel4_Max = 600;
int Light_Nivel5_Max = 1024;

int Sound_Nivel1_Max = 200;
int Sound_Nivel2_Max = 400;
int Sound_Nivel3_Max = 500;
int Sound_Nivel4_Max = 600;
int Sound_Nivel5_Max = 1024;

int velDirNivel0 = 0;
int velDirNivel1 = 200;
int velDirNivel2 = 220;
int velDirNivel3 = 230;
int velDirNivel4 = 240;
int velDirNivel5 = 255;

int velEsqNivel0 = 0;
int velEsqNivel1 = 200;
int velEsqNivel2 = 220;
int velEsqNivel3 = 230;
int velEsqNivel4 = 240;
int velEsqNivel5 = 255;
```

Figura 4.7-2 Variáveis a guardar na EEPROM.

FUNÇÃO SETUP

Na função *setup* foi colocado o código para inicializar os pins dos motores e dos led de forma a ser possível manter o robot parado e com o led apagado. Seguidamente foi criado um código para limpar a memória da EEPROM, percorrendo todas as posições da EEPROM e colocando os seus valores a zero. Demonstramos o codigo da função setup na Figura 4.7-3.

```
void setup()
{
  pinMode(pinMotorDireitoFrente, OUTPUT);
  pinMode(pinMotorDireitoTras, OUTPUT);
  pinMode(pinMotorEsquerdoFrente, OUTPUT);
  pinMode(pinMotorEsquerdoTras, OUTPUT);

  cmdParar(0);

  pinMode(pinLedRed, OUTPUT);
  pinMode(pinLedGreen, OUTPUT);
  pinMode(pinLedBlue, OUTPUT);

  digitalWrite(pinLedBlue, HIGH);
  //Limpar a memoria EEPROM
  for (int i = 0; i < 512; i++)
    EEPROM.write(i, 0);
  digitalWrite(pinLedBlue, LOW);

  Serial.begin(9600);
}
```

Figura 4.7-3 Função *Setup* na EEPROM

FUNÇÃO LOOP

Na função loop() são atribuídos os valores defult a cada uma das posições da valores *EEPROM*:

```
EEPROM.write(0, pinMotorDireitoFrente);
EEPROM.write(1, pinMotorDireitoTras);
EEPROM.write(2, pinMotorEsquerdoFrente);
EEPROM.write(3, pinMotorEsquerdoTras);

EEPROM.write(4, pinBumperDireito);
EEPROM.write(5, pinBumperEsquerdo);

EEPROM.write(6, pinLedRed);
EEPROM.write(7, pinLedGreen);
EEPROM.write(8, pinLedBlue);

EEPROM.write(9, velocidadeDireito);
EEPROM.write(10, velocidadeEsquerdo);

EEPROM.write(11, highByte(delayAndarFrente));
EEPROM.write(12, lowByte(delayAndarFrente));

EEPROM.write(13, highByte(delayAndarTras));
EEPROM.write(14, lowByte(delayAndarTras));

EEPROM.write(15, highByte(delayGirarEsquerda));
EEPROM.write(16, lowByte(delayGirarEsquerda));

EEPROM.write(17, highByte(delayGirarDireita));
EEPROM.write(18, lowByte(delayGirarDireito));

EEPROM.write(19, highByte(delayViraEsquerdaFrente));
EEPROM.write(20, lowByte(delayViraEsquerdaFrente));

EEPROM.write(21, highByte(delayViraEsquerdaTras));
EEPROM.write(22, lowByte(delayViraEsquerdaTras));

EEPROM.write(23, highByte(delayViraDireitaFrente));
EEPROM.write(24, lowByte(delayViraDireitoFrente));

EEPROM.write(25, highByte(delayViraDireitaTras));
EEPROM.write(26, lowByte(delayViraDireitoTras));

EEPROM.write(27, pinIR);
EEPROM.write(28, pinServo);

EEPROM.write(29, minServoRange);
EEPROM.write(30, maxServoRange);
EEPROM.write(31, delayServoRange);
EEPROM.write(32, incServoRange);

EEPROM.write(33, pinLuz);
EEPROM.write(34, pinSom);
EEPROM.write(35, pinBuzzer);
```

```
//-----

EEPROM.write(35, highByte(IR_Nivel1_Max));
EEPROM.write(37, lowByte(IR_Nivel1_Max));

EEPROM.write(38, highByte(IR_Nivel2_Max));
EEPROM.write(39, lowByte(IR_Nivel2_Max));

EEPROM.write(40, highByte(IR_Nivel3_Max));
EEPROM.write(41, lowByte(IR_Nivel3_Max));

EEPROM.write(42, highByte(IR_Nivel4_Max));
EEPROM.write(43, lowByte(IR_Nivel4_Max));

EEPROM.write(44, highByte(IR_Nivel5_Max));
EEPROM.write(45, lowByte(IR_Nivel5_Max));

//-----

EEPROM.write(55, highByte(Light_Nivel1_Max));
EEPROM.write(57, lowByte(Light_Nivel1_Max));

EEPROM.write(58, highByte(Light_Nivel2_Max));
EEPROM.write(59, lowByte(Light_Nivel2_Max));

EEPROM.write(60, highByte(Light_Nivel3_Max));
EEPROM.write(61, lowByte(Light_Nivel3_Max));

EEPROM.write(62, highByte(Light_Nivel4_Max));
EEPROM.write(63, lowByte(Light_Nivel4_Max));

EEPROM.write(64, highByte(Light_Nivel5_Max));
EEPROM.write(66, lowByte(Light_Nivel5_Max));

//-----

EEPROM.write(75, highByte(Sound_Nivel1_Max));
EEPROM.write(77, lowByte(Sound_Nivel1_Max));

EEPROM.write(78, highByte(Sound_Nivel2_Max));
EEPROM.write(79, lowByte(Sound_Nivel2_Max));

EEPROM.write(80, highByte(Sound_Nivel3_Max));
EEPROM.write(81, lowByte(Sound_Nivel3_Max));

EEPROM.write(82, highByte(Sound_Nivel4_Max));
EEPROM.write(84, lowByte(Sound_Nivel4_Max));

EEPROM.write(84, highByte(Sound_Nivel5_Max));
EEPROM.write(85, lowByte(Sound_Nivel5_Max));

//-----

EEPROM.write(95, velDirNivel0);
EEPROM.write(96, velDirNivel1);
EEPROM.write(97, velDirNivel2);
EEPROM.write(98, velDirNivel3);
EEPROM.write(99, velDirNivel4);
EEPROM.write(100, velDirNivel5);

//

EEPROM.write(105, velEsqNivel0);
EEPROM.write(106, velEsqNivel1);
EEPROM.write(107, velEsqNivel2);
EEPROM.write(108, velEsqNivel3);
EEPROM.write(109, velEsqNivel4);
EEPROM.write(110, velEsqNivel5);
```

Figura 4.7-4 Código para guardar as variáveis na EEPROM na função *loop*

Métodos Get e Set

Após termos os valores registados na *EEPROM*, foram criados métodos get e set para esses valores poderem ser lidos e alterados.

Demonstramos exemplos de métodos get na Figura 4.7-4:

```
int getVelocidadeEsquerdo(){
  return EEPROM.read(10);
}
int getDelayAndarFrente(){
  return word(EEPROM.read(11), EEPROM.read(12));
}
```

Figura 4.7-5 Exemplos de dois métodos get de variaveis da EEPROM

Demonstramos exemplos dos métodos get na Figura 4.7-5.

```
void setVelocidadeEsquerdo(int velocidadeEsquerdo){
  EEPROM.write(10, velocidadeEsquerdo);
}
void setDelayAndarFrente(int delayAndarFrente){
  EEPROM.write(11, highByte(delayAndarFrente));
  EEPROM.write(12, lowByte(delayAndarFrente));
}
```

Figura 4.7-6 Exemplos de dois métodos set de variáveis da EEPROM

4.8. Utilização da Biblioteca na conceção de uma Aplicação de Testes de Funcionalidades do Robot

Como a biblioteca possui uma vasta quantidade de funções, para facilitar o teste de cada uma das funções da biblioteca, foi desenvolvido uma aplicação específica para esse mesmo propósito. Assim torna-se mais fácil e pratico, para além de ser mais rápido. Basta conectar a aplicação ao robot, selecionar qual a função que se deseja correr e observar se o resultado é o desejado. Demonstramos a interface da aplicação na Figura 4.8-1. Para a implementação desta aplicação foram utilizadas as funções disponíveis na Biblioteca *BSAA.jar*. O processo foi o mesmo, como qualquer outra aplicação que use a biblioteca.

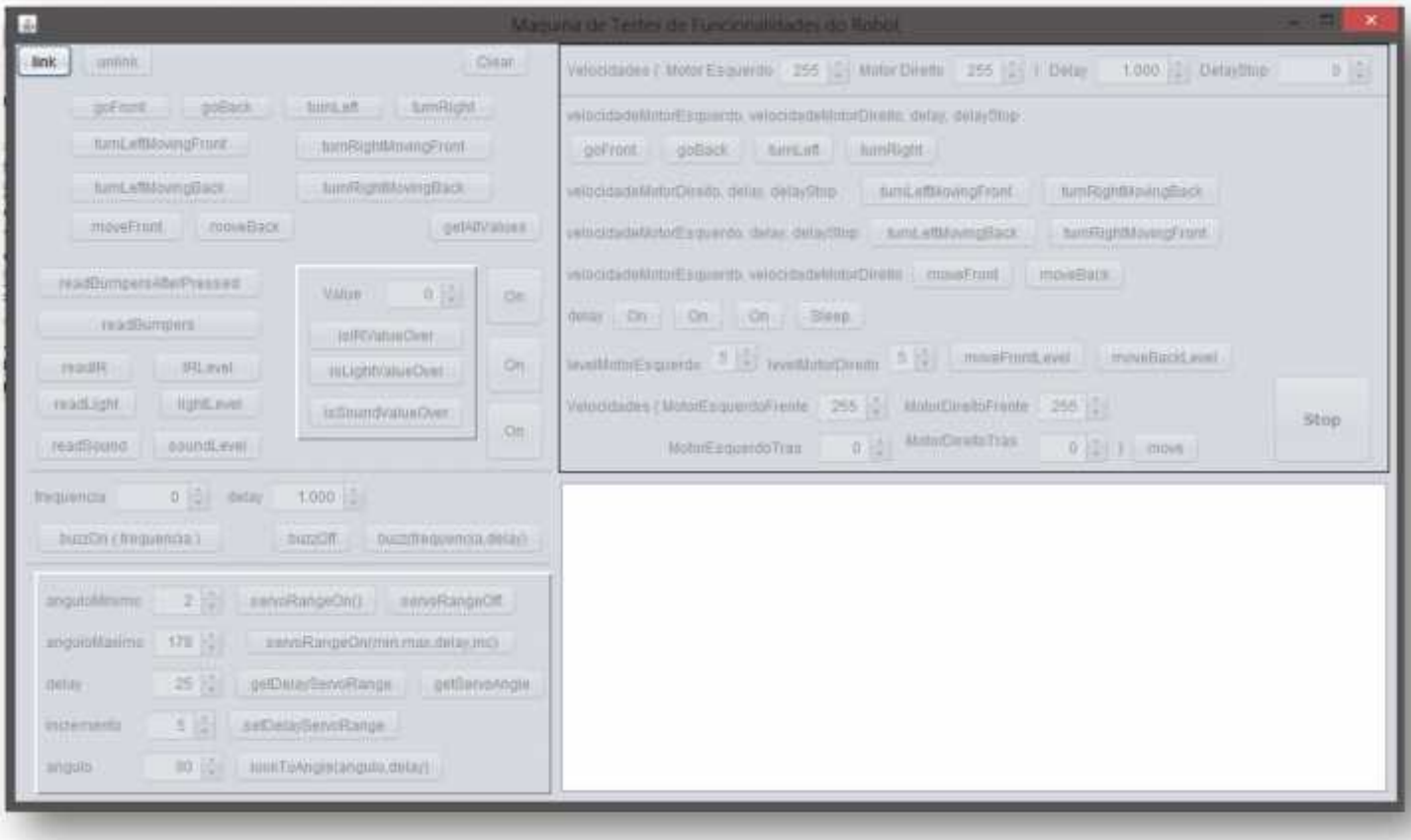


Figura 4.8-1 Interface da Aplicação de Testes de Funcionalidades do Robot sem conexão estabelecida

Antes de se ter acesso as funcionalidades do Robot, tem de primeiro se estabelecer a conexão com o mesmo. Para isso, terá de se clicar no botão "*link*" da aplicação.

Assim que a conexão seja estabelecida com sucesso, todas as funcionalidades serão desbloqueadas, como apresentado na Figura 4.8-2.

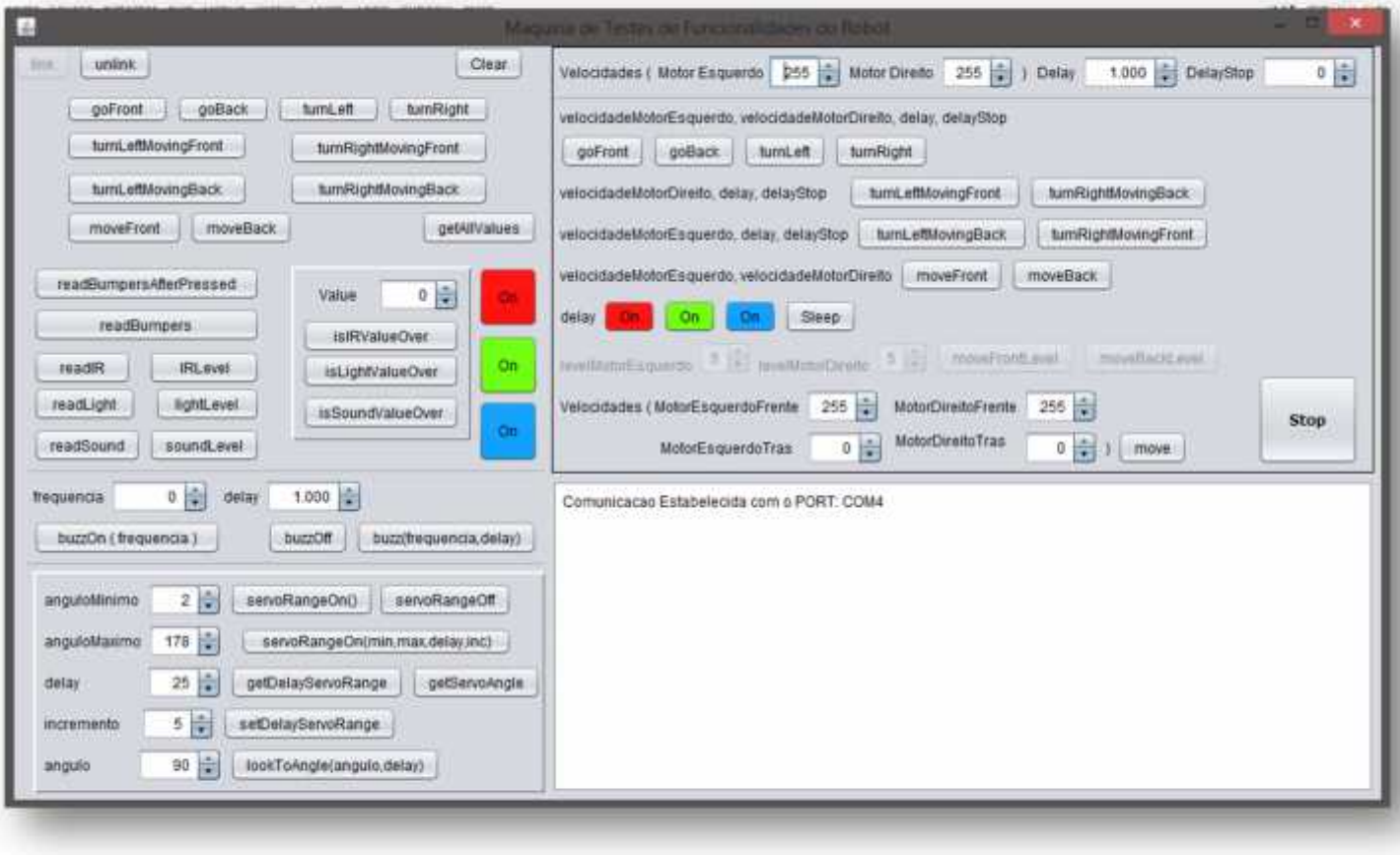


Figura 4.8-2 Interface da Aplicação de Testes de Funcionalidades do Robot com conexão estabelecida

O processo de se colocar todas as funcionalidades do robot numa única aplicação foi um grande desafio. Passamos então a explicar cada uma das áreas da aplicação:

Numa primeira área temos os botões responsáveis por estabelecer (link) e terminar (unlink) a comunicação com o robot e o botão clear, responsável por limpar os registos das funções efetuadas.

Figura 4.8-3 Área 1 da Máquina de Testes de Funcionalidades do Robot

Numa segunda área temos disponíveis os botões das funções responsáveis pela locomoção do robot que correm com valores default e a função *getAllValues* que devolve os valores de todos os sensores.

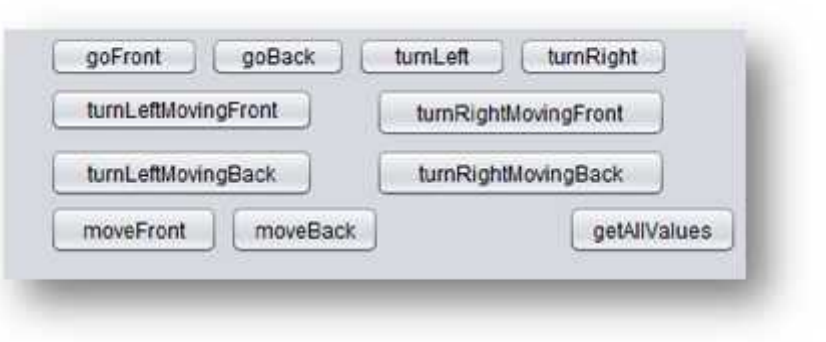


Figura 4.8-4 Área 2 da Aplicação de Testes de Funcionalidades do Robot

Seguidamente temos disponíveis os botões para as funções de interação com os sensores e os leds (as funções em que não é enviado o parâmetro do delay). Temos disponível um campo para indicar o valor a enviar pelas funções *isIRValueOver*, *isLightValueOver* e *isSoundValueOver*. O valor do campo delay esta limitado entre 0 e 10000 milisegundos para garantir que apenas valores validos são enviados para as funções. Nos botões dos leds, após um deles ser acionado, o seu texto muda para *Off*.

Figura 4.8-5 Área 3 da Aplicação de Testes de Funcionalidades do Robot

Temos as funções de interação com o buzzer do robot. Em cada um dos botões encontra-se indicado quais são os parâmetros da função.

Figura 4.8-6 Área 4 da Máquina de Testes de Funcionalidades do Robot

Para a interação com o servo do robot temos as todos os campos dos parâmetros necessários lado a lado com os botões das funções. Também aqui temos indicado no botão quais são os parâmetros enviados para cada uma das funções. Os campos estão implementados de forma a não ser possível

ocorrer erros nas funções. O valor do *anguloMinimo* é sempre inferior ao valor do *anguloMaximo*, da mesma forma que o valor do incremento nunca é superior ao valor da diferença entre o *anguloMáximo* e *anguloMinimo*, o valor do *delay* esta limitado entre 0 e 255 e o valor do *ângulo* esta limitado entre 2 e 178.

Figura 4.8-7 Área 5 da Aplicação de Testes de Funcionalidades do Robot

Relativamente as funções responsáveis pela locomoção do robot que recebem parâmetros, estão apresentadas na Figura 4.8-8. Na parte superior encontra-se todos os parâmetros usados por estas funções. Seguidamente temos os botões organizados da seguinte forma: primeiro a lista com os parâmetros que as funções recebem seguido dos botões correspondentes as funções que os usam.

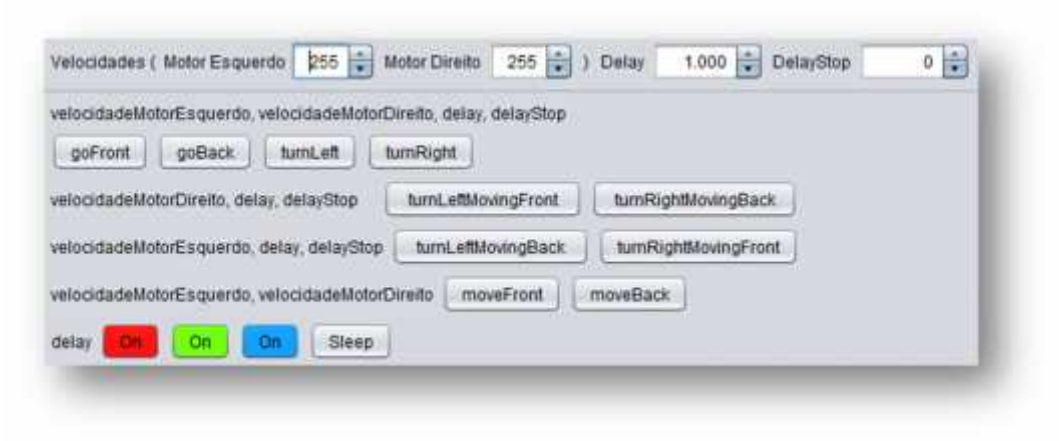


Figura 4.8-9 Área 6 da Aplicação de Testes de Funcionalidades do Robot

Relativamente as funções que enviam os níveis dos motores em alternativa aos valores das velocidades, estes encontram-se desativadas por limitações do *hardware* do robot atual.

Figura 4.8-9 Área 7 da Aplicação de Testes de Funcionalidades do Robot

Seguidamente temos a função move, com todos os seus parâmetros e a função *stop*.

Figura 4.8-10 Área 8 da Aplicação de Testes de Funcionalidades do Robot

Finalmente temos a Área onde é mostrado as respostas do robot.

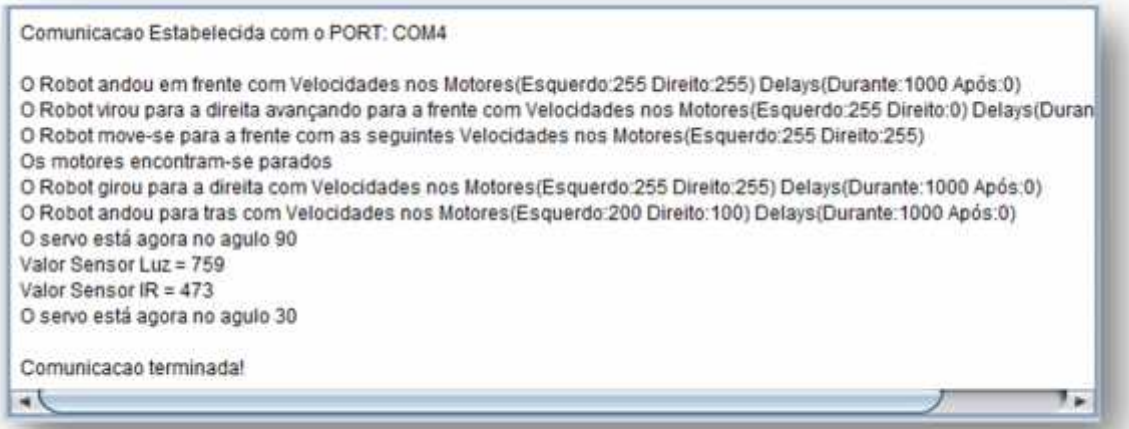


Figura 4.8-11 Área 9 da Aplicação de Testes de Funcionalidades do Robot

4.9. Utilização da Biblioteca na conceção de um Configurador de robots

Dado a existência de funções que utilizam valores predefinidos, foi desenvolvido um configurador de robots para se proceder a calibragem desses valores. O configurador de robots é a aplicação onde é possível definir-se os valores das funções, como as velocidades dos motores, os *delays* os níveis, assim como os valores dos pins onde estão ligados os atuadores e sensores do robot.

O principal propósito para a existência desta aplicação é a possibilidade de os valores num robot diferirem de outros robots para cumprirem os mesmos objetivos. E que com o passar do tempo alguns componentes do robot se possam danificar e ter por exemplo, para ambas as rodas se deslocarem a mesma velocidade, colocar uma roda a rodar mais “rápido” que a outra.

Contrariamente a aplicação de testes de funcionalidades do robot, a aplicação do configurador não interage com a biblioteca da mesma forma que qualquer outra aplicação. Esta aplicação acede diretamente a classe *InterfacePortaSerie*, e tem criadas as suas próprias funções de interação com o robot. O motivo para essa diferença é que esta aplicação não utiliza as funções criadas na biblioteca *BSAA.jar* mas sim funções próprias e restritas para a interação com a *EEPROM* do robot

As funções resumem-se a funções de *gets* e *sets* de todas as variáveis, que não estão diretamente disponíveis ao utilizador da biblioteca.

```
javadoc
public int getPinMotorDireitoFrente() {
    comunicacao.enviarMSG("GPMDF", 0); // é enviada uma mensagem para o robot
    return Integer.parseInt(comunicacao.receberMSG()); // retorna a resposta do robot, passando o valor para o formato int
}

javadoc
public void setPinMotorDireitoFrente(int novoPin) {
    if (novoPin >= 0 && novoPin <= 13) { //verifica se o novoPin se encontra dentro dos valores admitidos
        comunicacao.enviarMSG("SPMDF" + novoPin, delay); // caso esteja correto, é enviada uma mensagem para o robot
    } else { // caso o valor da variável não esteja dentro dos valores admitidos, é emitida uma mensagem de erro para o utilizador
        msg.mensagemErro("Erro: Os pins teem de estar entre 0 e 13!!!");
    }
}
```

Figura 4.9-1 Exemplos de funções *get* e *set* da aplicação do configurador

Demonstramos a interface da aplicação na Figura 4.9-2.

Figura 4.9-2 Interface do Configurador de Robots sem conexão estabelecida

Igualmente a aplicação de testes, as funcionalidades desta aplicação apenas são desbloqueadas após a conexão com o robot, que se efetua através do botão “*link*”.

Figura 4.9-3 Interface do Configurador de Robots com conexão estabelecida – parte 1

Figura 4.9-4 Interface do Configurador de Robots com conexão estabelecida – parte 2

Como apresentado nas Figuras 4.9-3 e 4.9-4, quando a ligação é estabelecida todos os valores são preenchidos, a exceção dos níveis das velocidades dos motores, pois como já explicado essas funções foram desativadas por falta de memória no robot.

Igualmente a aplicacao de testes temos disponível um botão "*unlink*" para se terminar a conexão com o robot. Temos também um botão "*Valores Default*" que preenche todos os campos com valores predefinidos. Ao clicar-se no botão "*Get Valores*" este volta a carregar todos os valores do robot. Finalmente no botão "Gravar Alterações", como o próprio nome indica serve para guardar os valores alterados na EEPROM do robot.

Por motivos de performance quando se clica no botão "Gravar Alterações" apenas os valores que foram alterados são enviados para o robot, é guardado um registo de todos os valores quando estes são carregados e registam-se as alterações ao longo da utilização de forma a saber quais são os valores que foram alterados e quais se mantém inalterados.

Todos os campos de preenchimento dos valores encontram-se com as devidas limitações e validações, ou seja, os valores dos pins estão limitados entre 0 e 13, os valores dos *delays* entre 0 e 10000, os valores das velocidades entre 0 e 255 e os valores dos níveis respeitam todas as regras definidas para os níveis na secção 4.5.1 (os valores das *labels* atualizam-se automaticamente a cada alteração para um melhor controlo dos valores dos níveis).

4.10. Utilização da Biblioteca na conceção de uma biblioteca com foco educacional

Um dos contextos de utilização da biblioteca de abstração é o educacional, assim foi implementada uma biblioteca educacional que permitisse avaliar posteriormente a utilização da biblioteca de abstração. Essa biblioteca é focalizada no ensino da programação, sendo que toda a estrutura da biblioteca foi desenvolvida com esse objetivo em mente.

De forma a testar a biblioteca foram desenhados uma serie de exercícios destinados ao ensino de vários conceitos de programação. A conceção dos exercícios não foi objetivo da tese, estes foram-nos disponibilizados e procedemos a sua implementação.

Foram criados 5 packages principais destinados a alocar classes com funções do robot. Os packages são: *ACT*, *REACT*, *LEARN, EVOLVE* e *INSIST*. No package *ACT* encontra-se uma classe com as funções responsáveis por controlar os atuadores do robot e no package *REACT* ficou definida uma classe composta pelas funções que controlam os sensores do robot. Os restantes packages ficaram criados sem conteúdo e guardados para possíveis evoluções da biblioteca no futuro.

As classes da biblioteca de ensino são compostas por alguns métodos da biblioteca de abstração, e em alguns casos é feita uma reutilização dos métodos da biblioteca de abstração para a criação de novos métodos que interagem com o robot.

Um dos desafios iniciais na construção desta biblioteca foi a divisão das funções do robot em duas classes. Note-se que o primeiro e principal requisito para a execução de uma função é o estabelecimento prévio da comunicação com o robot, e que apenas poderá haver um canal de comunicação de cada vez. Na biblioteca de abstração todas as funções estão apenas numa classe, onde é estabelecida a comunicação com o robot e todas as funções recorrem a esse canal de transmissão para comunicarem com o robot. Como na biblioteca educacional era pretendido dividir-se as funções em duas classes, para a execução das funções em cada classe teria de existir um canal de comunicação.

A solução passou pela criação de uma classe *Singleton*. Como demonstrado na Figura 4.10 -1, nesta classe é denominada por *RobotSingleton* onde é criado um objeto da classe Robot apenas uma vez, ou seja, o objeto é inicializado a null e quando o método *getInstance* é chamado este inicializa o objeto apenas quando este é null e devolve-o no fim do método. Encontra-se também nesta classe o método para terminar a comunicação com o robot.

```
public class RobotSingleton {

    private static Robot robot = null;

    protected RobotSingleton() {
        // Exists only to defeat instantiation.
    }

    public static Robot getInstance() {
        if (robot == null) {
            robot = new Robot();
        }
        return robot;
    }

    public static void terminarComunicacao() {
        if (robot != null) {
            robot.terminarComunicacao();
        }
    }
}
```

Figura 4.10-1 Classe RobotSingleton

Nas classes dos packages *ACT* e *REACT*, é declarado um objeto do tipo Robot para cada uma delas, mas inicializados com o método *getInstance* da classe *RobotSingleton*.

```
private final Robot instanciaRobot;

public FuncoesACT() {
    instanciaRobot = bibliotecaensino.RobotSingleton.getInstance();
}
```

Figura 4.10-2 Criação do objeto do tipo Robot na classe *FuncoesACT*

Com a utilização da classe *RobotSingleton*, garantimos que apenas uma instância do robot é criada e por consequente ira existir apenas um canal de transmissão entre o computador e o robot.

Demonstramos na Figura 4.10-3 o diagrama de classes da biblioteca educacional desenvolvida.

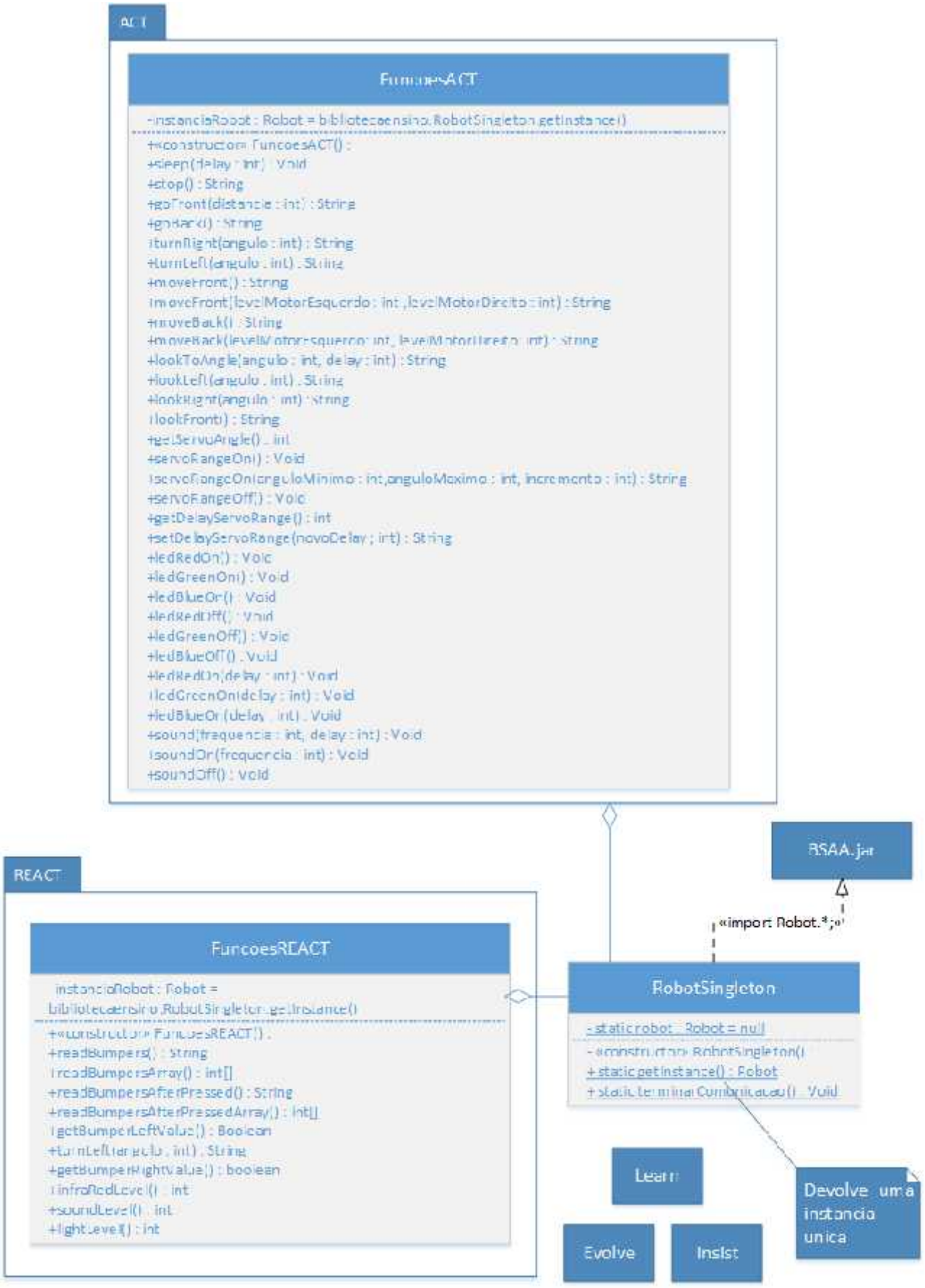


Figura 4.10-3 Diagrama de classes da Biblioteca de Ensino

4.10.1. *Classe FuncoesACT*

Esta classe é responsável pelas funções que controlam os atuadores do robot. Nela estão contidas chamadas a algumas funções da classe Robot da biblioteca de abstração e é feita uma reutilização de algumas funções na criação de novas funções.

```
public void sleep(int delay) {
    instanciaRobot.sleep(delay);
}
```

Figura 4.10-4 Exemplo de uma chamada a uma função da biblioteca de abstração

Passamos então a detalhar os métodos que foram criados especificamente para a biblioteca educacional.

MÉTODO GOFRONT(DISTANCIA : INT) : STRING

Tabela 4.10.1-1 Detalhes do método goFront

+ goFront(distancia : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
goFront	É enviada uma mensagem para o robot para andar em frente por uma distância em centímetros definida pelo utilizador. Este ira se mover com velocidades e *delays* de *default*.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.10.1-2 Parâmetros do método goFront

Nome do parâmetro	Tipo	Descrição
Distancia	int	A distância em centimetros. O valor da distância tem de ser múltiplo de 5.

Para a implementação desta funcionalidade no robot, foi utilizado o método "*goFront() : String*", que desloca o robot aproximadamente 5cm em frente. Foi então criado um algoritmo que reutiliza esse método e desloca o robot a distância recebia como parâmetro. Como o método original desloca o robot 5cm, a distância enviada neste método tem de ser um múltiplo de 5.

O movimento executado pelo robot é semelhante ao representado na Figura 4.5-3.

MÉTODO GOBACK(DISTANCIA : INT) : STRING

Tabela 4.10.1-3 Detalhes do método goBack

+ goBack(distancia : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
goBack	É enviada uma mensagem para o robot para andar para trás por uma distância em centímetros definida pelo utilizador. Este ira se mover com velocidades e *delays* de *default*.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.10.1-4 Parâmetros do método goBack

Nome do parâmetro	**Tipo**	**Descrição**
Distancia	int	A distância em centimetros. O valor da distância tem de ser múltiplo de 5.

Para a implementação desta funcionalidade no robot, foi utilizado o método "*goBack() : String*", que desloca o robot aproximadamente 5cm para trás. De forma semelhante ao método "*goFront(distancia : int) : String*" também foi então criado um algoritmo que reutiliza esse método e desloca o robot a distância recebia como parâmetro. Como o método original desloca o robot 5cm, a distância enviada neste método tem de ser um múltiplo de 5.

O movimento executado pelo robot é semelhante ao representado na Figura 4.5-4.

MÉTODO TURNLEFT(ANGULO : INT) : STRING

Tabela 4.10.1-5 Detalhes do método turnLeft

+ turnLeft(distancia : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnLeft	É enviada uma mensagem para o robot para girar para a esquerda x graus definidos pelo utilizador. Este ira se mover com velocidades e *delays* de *default*.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.10.1-6 Parâmetros do método turnLeft

Nome do parâmetro	Tipo	Descrição
Angulo	int	Valor do angulo em graus para virar o robot para a esquerda. O valor do angulo tem de ser superior a 5º e inferior a 360º, e tem de ser múltiplo de 5.

Para a implementação desta funcionalidade no robot, foi utilizado o método "*turnLeft() : String*", que gira o robot aproximadamente 5 graus para a esquerda. Neste método também foi criado um algoritmo que reutiliza o método da classe Robot e gira o robot o número de ângulos recebidos como parâmetro. Como o método original gira o robot 5 graus, o valor do angulo enviado neste método tem de ser um múltiplo de 5, por último definimos que o angulo tem de ser um ângulo inferior a 360 graus.

O movimento executado pelo robot é semelhante ao representado na Figura 4.5-6.

MÉTODO TURNRIGHT(ANGULO : INT) : STRING

Tabela 4.10.1-7 Detalhes do método turnRight

+ turnRight (distancia : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
turnRight	É enviada uma mensagem para o robot para girar para a direita x graus definidos pelo utilizador. Este ira se mover com velocidades e *delays* de *default*.	A resposta editada enviada pelo robot.

Este método tem como parâmetros:

Tabela 4.10.1-8 Parâmetros do método turnRight

Nome do parâmetro	Tipo	Descrição
Angulo	int	Valor do angulo em graus para virar o robot para a direita. O valor do angulo tem de ser superior a 5º e inferior a 360º, e tem de ser múltiplo de 5.

Nesta funcionalidade, foi utilizado o método "*turn*Right*() : String"*, que gira o robot aproximadamente 5 graus para a direita. Foi também criado um algoritmo que reutiliza o método da classe Robot e gira o robot o número de ângulos recebidos como parâmetro. Como o método original gira o robot 5 graus, o valor do angulo enviado neste método tem de ser um múltiplo de 5, por último definimos que o ângulo tem de ser um angulo inferior a 360 graus.

O movimento executado pelo robot é semelhante ao representado na Figura 4.5-5.

No desenvolvimento das seguintes funcionalidades, *lookLeft* e *lookRight*, foi utilizado o método "*lookToAnglet() : String"*, que posiciona o servo num determinado angulo enviado como parâmetro e recebe um delay que é o tempo que o servo demora a alcançar a posição de destino.

A função *lookLeft* coloca o motor servo num angulo definido pelo utilizador, rodando o servo para a esquerda e o *delay* que é enviado corresponde ao valor do angulo * 5.

A função *lookRight* coloca o motor servo num angulo definido pelo utilizador, rodando o servo para a direita e o *delay* que é enviado corresponde ao valor do angulo * 5.

Método lookLeft (angulo : int) : String

Tabela 4.10.1-9 Detalhes do método lookLeft

+ turnRight (distancia : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
lookLeft	Vira o servo para a esquerda x ângulos definidos pelo utilizador.	A resposta editada enviada pelo robot, que informa sobre o novo angulo do servo.

Este método tem como parâmetros:

Tabela 4.10.1-10 Parâmetros do método lookLeft

Nome do parâmetro	Tipo	Descrição
Angulo	int	Valor em graus que vai virar o servo. Tem de estar entre 1º e 176º, assim como a posição final do servo não pode exceder os 178º.

Método lookRight (angulo : int) : String

Tabela 4.10.1-11 Detalhes do método lookRight

+ turnRight (distancia : int) : String		
Nome do Método	**Descrição do método**	**O que retorna?**
lookRight	Vira o servo para a direita x ângulos definidos pelo utilizador.	A resposta editada enviada pelo robot, que informa sobre o novo ângulo do servo.

Este método tem como parâmetros:

Tabela 4.10.1-12 Parâmetros do método lookLeft

Nome do parâmetro	Tipo	Descrição
Angulo	int	Valor em graus que vai virar o servo. Tem de estar entre 1º e 176º, assim como a posição final do servo não pode ser inferior a 2º

4.10.2. *Classe FuncoesREACT*

Nesta classe apenas são utilizadas funções da biblioteca de abstração não havendo a necessidade de criar novas Funções.

4.10.3. *Modelo de código da Biblioteca Educacional*

Demonstramos na Figura 4.10-5 o modelo de código criado para a utilização das funções da biblioteca Educacional.

```
package bibliotecaensino;

//import para utilizar as funcoes do robot
import ACT.FuncoesACT;

/**
 * Criado por:
 * Data:
 * Este ficheiro é um template inicial para a utilização do robot
 */

public class Template {

    public static void main(String[] args) {
        // Instancializar o objeto do robot
        FuncoesACT robotACT = new FuncoesACT();

        // Inserir codigo aqui!!!
        //robotACT.goFront();

        ///Terminar sempre o programa com bibliotecaensino.RobotSingleton.TerminarComunicacao();
        bibliotecaensino.RobotSingleton.terminarComunicacao();
    }
}
```

Figura 4.10-5 Template da Biblioteca educacional

O modelo de código para a utilização da biblioteca educacional é muito semelhante ao *template* da biblioteca de abstração. É necessário adicionar-se ambas as bibliotecas, tanto a de abstração como a biblioteca de ensino. Apos termos as bibliotecas adicionadas ao nosso projeto, temos de fazer o *import* ou da classe *FuncoesACT* ou da classe *FuncoesREACT*, inicializar os objetos das classes e poderemos aceder as funções dessas classes. No fim do código tal como na biblioteca de abstração é obrigatório terminar-se a conexão com o robot.

4.10.4. *Exercícios Desenvolvidos*

Para os testes da biblioteca, foram-nos fornecidos um conjunto de exercícios de programação. Esses exercícios encontram-se separados por 5 grupos, aumentando o grau de dificuldade a cada grupo. A biblioteca Educacional foi estruturada de forma a possibilitar a resolução desses exercícios. Para facilitar a interpretação das ações do robot, de modo a confirmar se este faz o que realmente é pretendido, definiu-se que todos os exercícios correm dentro de um ciclo de repetição for, que corre 20 vezes.

EXERCÍCIOS GRUPO A

Os exercícios contidos neste grupo são exercícios que atuam apenas com os atuadores do robot, através da classe *FuncoesACT* para ambientar os utilizadores ao robot. Para a resolução dos exercícios deste grupo são utilizadas funções diretas da classe *FuncoesACT* sem introdução de uma complexidade acrescida.

- **Exercício A1:** O robot anda para a frente 3 vezes, após concluir esses movimentos aguarda 1 segundo e depois anda para trás 3 vezes.
- **Exercício A2:** O robot olha para a direita, olha para a esquerda e volta a olhar para a frente. (O servo começa centrado).
- **Exercício A3:** O robot buzina 3 vezes.
- **Exercício A4:** O robot pisca o led 3 vezes.
- **Exercício A5:** O robot vira duas vezes para a direita 180º e volta a virar 2 vezes para a esquerda 180º. Entre cada movimento do robot, este espera 3 segundos.
- **Exercício A6:** O robot anda para a frente, vira a esquerda 90º, anda para frente, vira 90º.
- **Exercício A7:** O robot acende o led enquanto anda para a frente, apita enquanto anda para trás.
- **Exercício A8:** O robot movimenta-se “desenhando” um quadrado com os seus movimentos.
- **Exercício A9:** O robot executa uma dança livre utilizando o que aprendeu ate agora.

EXERCÍCIOS GRUPO B

Neste grupo de exercícios, são introduzidas as funções da classe *FuncoesREACT*, introduzindo os sensores do robot aos utilizadores. Nestes exercícios são introduzidas nas resoluções as instruções

de seleção (são instruções que permitem selecionar um de vários caminhos alternativos na execução do programa). Para a resolução destes exercícios são necessárias ambas as classes *FuncoesACT* e *FuncoesREACT*.

- **Exercício B1**: Se detetar luz no sensor, o robot apita 3 vezes.
- **Exercício B2**: Lê o valor dos *bumpers*, se for o *bumper* esquerdo a estar pressionado, o robot acende o led 3 vezes, se for o *bumper* direito, o robot apita 3 vezes.
- **Exercício B3:** Se detetar um obstáculo no sensor de IR, o robot apita 3 vezes.
- **Exercício B4:** Se detetar som, o robot apita 3 vezes.
- **Exercício B5:** Se detetar luz, o robot avança 20cm.
- **Exercício B6:** Se detetar som, o robot avança 20cm.
- **Exercício B7:** Se detetar som ou luz, o robot avança 20cm.
- **Exercício B8:** Se detetar luz, o robot avança 20cm, se detetar som, o robot recua 20cm.
- **Exercício B9:** Se detetar obstáculo no *bumper* esquerdo, o robot olha 30º para a esquerda, aguarda 3 segundos e volta a olhar em frente. Se detetar obstáculo no *bumper* direito, o robot olha 30º para a direita, aguarda 3 segundos e volta a olhar em frente.
- **Exercício B10:** O robot executa uma dança livre utilizando o que aprendeu ate agora.

EXERCÍCIOS GRUPO C

Nestes exercícios também são utilizadas as funções de ambas as classes *FuncoesACT* e *FuncoesREACT*. São introduzidos as instruções de repetição, ou ciclos, que permitem repetir um conjunto de instruções enquanto se verifique determinada condição.

- **Exercício C1:** O robot avança em incrementos (de x centímetros) enquanto tiver luz.
- **Exercício C2:** O robot roda enquanto deteta um obstáculo no *bumper* x. (Ex: Se toca no *bumper* direito, roda para a esquerda, se toca no *bumper* esquerdo, roda para direita).
- **Exercício C3:** O robot avança enquanto deteta luz, e recua enquanto deteta um obstáculo no sensor de IR (o robot deteta luz e avança, ao detetar obstáculo com IR recua).
- **Exercício C4:** Enquanto luz apita alternado, se deteta um obstáculo num dos *bumpers* apita continuamente.
- **Exercício C5:** O robot avança enquanto deteta som.

- **Exercício C6:** O robot avança enquanto deteta som, se detetar um obstáculo com IR recua.

- **Exercício C7:** O robot avança enquanto deteta luz, se detetar obstáculo com um *bumper* desvia para o lado contrário do *bumper*.

- **Exercício C8:** O robot executa uma dança livre utilizando o que aprendeu até agora.

EXERCÍCIOS GRUPO D

Neste grupo o grau de dificuldade de resolução dos exercícios aumenta. Nesta altura o utilizador já se encontra ambientado ao robot e ao seu funcionamento, então os exercícios passam pelo desenvolvimento algoritmos mais complexos que recorrem ao uso de instruções do robot. Nestes exercícios também são utilizados as funções de ambas as classes.

- **Exercício D1:** O robot buzina enquanto a luz detetada no sensor é maior do que o valor máximo de luz já registado. Quando clica num dos *bumpers* faz *reset* ao máximo.

- **Exercício D2:** O robot buzina enquanto um obstáculo detetado no sensor de IR se encontra mais próximo do que já registado. Quando clica num dos *bumpers* faz *reset* ao máximo. (Ex: aproxima-se a mão e o robot buzina, volta-se a aproximar a mão a mesma distância e o robot já não reage. Aproxima-se a mão mais próximo e o robot volta a buzinar).

- **Exercício D3:** O robot procura a origem da luz rodando a torre (sugestão: esquerda – frente – direita), após concluir a sua busca, pisca o LED e olha para o lado onde detetou a luz maior.

- **Exercício D4:** O robot procura a origem do som rodando a torre (sugestão: esquerda – frente – direita), após concluir a sua busca, pisca o LED e olha para o lado onde detetou a luz maior.

- **Exercício D5:** O robot encontra-se num labirinto, percorre-o e procura a saída (através do sensor IR). Desvia-se se detetar um obstáculo. Quando o robot chegar ao destino, acende-se uma luz para este saber que já concluiu o percurso.

- **Exercício D6:** O robot encontra-se num labirinto, percorre-o e procura a saída (através dos *bumpers*). Se intercetar um obstáculo buzina, recua, corrige e avança. Quando o robot chegar ao destino, acende-se uma luz para este saber que já concluiu o percurso.

- **Exercício D7:** O robot encontra-se num labirinto, percorre-o ida e volta (através do sensor IR e dos *bumpers*). O robot desvia-se dos obstáculos quando detetados no IR e corrige quando bate num dos *bumbers.*

- **Exercício D8:** Faz o mesmo que no Exercício D7 mas o robot aguarda um sinal sonoro para iniciar o seu percurso.

- **Exercício D9:** O robot aguarda até detetar um sinal de luz no sensor para iniciar uma dança. Após detetar o sinal de luz, o robot posiciona-se uma posição a sua escolha, apita 3 vezes e inicia a sua dança. O robot não sabe onde a sua posição inicial.

EXERCÍCIOS GRUPO E

Este grupo de exercícios foi criado para trabalho futuro do projeto desta tese, pois para resolução de alguns destes exercícios era requerido hardware mais potente e completo, como a utilização de outros motores no robot.

Neste grupo os exercícios atingem um grau de dificuldade bastante elevado, para demonstrar que a biblioteca pode ser usada numa grande variedade de exercícios desde exercícios de complexidade simples como os do Grupo A, mas também exercícios bastante difíceis e complexos. Nestes exercícios também são utilizados as funções de ambas as classes.

- **Exercício E1:** O robot encontra-se num labirinto, percorre-o (através do sensor IR e dos *bumpers*) e procura uma luz (máximo absoluto). O robot desvia-se dos obstáculos quando detetados no IR e corrige quando bate num dos *bumbers.* Tem como objetivo encontrar a localização de uma luz no labirinto, quando a encontrar apita.

- **Exercício E2:** O robot encontra-se num labirinto, percorre-o (através do sensor IR e dos *bumpers*) e procura o som (máximo absoluto). O robot desvia-se dos obstáculos quando detetados no IR e corrige quando bate num dos *bumbers.* Tem como objetivo encontrar a localização da origem de um som no labirinto, quando a encontrar apita.

- **Exercício E3:** O robot encontra-se num labirinto, percorre-o (através do sensor IR e dos *bumpers*) e procura uma luz (máximo absoluto). O robot desvia-se dos obstáculos quando detetados no IR e corrige quando bate num dos *bumbers.* Tem como objetivo encontrar a localização de uma luz no labirinto, quando a encontrar apita e volta a posição inicial.

- **Exercício E4:** O robot encontra-se num labirinto, percorre-o (através do sensor IR e dos *bumpers*) e procura duas luzes (máximo absoluto). O robot desvia-se dos obstáculos quando detetados no IR e corrige quando bate num dos *bumbers.* Tem como objetivo encontrar a localização da origem de duas luzes no labirinto, apita cada vez que encontra uma luz. Após encontrar a primeira luz o robot apita, a luz é apagada e o robot continua a procurar a segunda luz, voltando a apitar quando encontrar a segunda luz.

- **Exercício E5:** Semelhante ao Exercício E4, mas as luzes nunca se apagam mantendo-se sempre duas luzes acesas no labirinto.

- **Exercício E6:** Semelhante ao Exercício E4, mas após encontrar as duas luzes, o robot volta a posição inicial.

- **Exercício E7:** Semelhante ao Exercício E5, mas após encontrar as duas luzes, o robot volta a posição inicial.

Capítulo 5
Avaliação e discussão de resultados

Nesta fase é descrita a integração da biblioteca em aplicações java, detalhando passo a passo o processo de criação de um projeto java em dois IDE's distintos que utilizam a biblioteca para interagir com o robot farrusco, descrevendo esses mesmos IDE's.

Essencialmente descrevemos os testes efetuados e discutimos os resultados.

5.1. Metodologia

Após a conclusão da biblioteca era necessário testar o resultado final para comprovar se realmente funciona.

A metodologia utilizada para avaliação da biblioteca de abstração foi a seguinte:

- Numa primeira fase, a biblioteca foi testada em diferentes IDE's, concretamente o IDE Netbeans e o Eclipse IDE.
- Numa segunda fase, para a avaliação da biblioteca de abstração foi criada uma segunda biblioteca tendo como base de construção a biblioteca de abstração e implementados exercícios de forma a avaliar as funções de abstração do robot.

O contexto de utilização escolhido para a avaliação da biblioteca de abstração foi o contexto educacional. O ambiente de testes utilizado foi o ambiente que será utilizado em contexto de aprendizagem final, que demonstramos na Figura 5.1-1.

Figura 5.1-1 Ambiente de testes da biblioteca

5.2. Integração da Biblioteca em aplicações Java

A biblioteca foi desenvolvida na linguagem de programação java, e permite ser integrada em qualquer projeto java. Nos testes da biblioteca foram utilizados dois IDE's, o Netbeans IDE e o Eclipse IDE. Para a utilização da biblioteca num projeto java basta proceder a criação normal de um projeto java, adicionar o jar da biblioteca ao projeto e proceder ao imports da classe *Robot*.

Nota: A biblioteca desenvolvida utiliza as bibliotecas rxtx [38] para a comunicação série em java. A biblioteca rxtx vem disponível para diversos sistemas operativos: Windows (32 bits e 64 bits), Mac OS X (x86 e ppc), Linux (x86, x86_64, ia64) e Solaris (sparc). Para se utilizar a biblioteca rxtx em projetos java, tem de se proceder a uma pequena "instalação" apropriada para cada tipo de sistema operativo.

Para a instalação em ambiente Windows procede-se aos seguintes passos:

- Copie o ficheiro RXTXcomm.jar para o diretório do jdk utilizado no projeto dentro de \jre\lib\ext (dentro da pasta de instalação do java)
- Copie o ficheiro rxtxSerial.dll para o diretório \jre\bin

Para a instalação da biblioteca rxtx em outros sistemas operativos, as instruções encontram-se disponíveis em anexo.

[38] **Fonte**: http://rxtx.qbang.org/wiki

5.2.1. *Emparelhamento dos dispositivos Bluethooth*

Após a instalação da biblioteca rxtx, procede-se ao emparelhamento dos dispositivos Bluetooth do robot com o computador. É necessário criar-se a ligação entre os dois dispositivos e antes de cada utilização deve-se certificar que os dispositivos se encontram emparelhados e ativos.

As instruções de emparelhamento dos dispositivos encontramse disponíveis em anexo.

5.2.2. *Netbeans IDE*

Figura 5.2-1 Logotipo Netbeans

Segundo informação disponível no site oficial do netbeans[39], em junho de 2000, o NetBeans foi tornado open-source pela Sun Microsystems, que continuou a ser o patrocinador do projeto até janeiro de 2010, ano em que a Sun Microsystems tornou-se uma subsidiária da Oracle. Netbeans IDE (integrated development environment) fornece suporte para várias linguagens de programação, nomeadamente Java, PHP, JavaFX, C/C++, JavaScript, entre outros e frameworks. O NetBeans IDE é escrito em Java e pode ser executado em Windows, OS X, Linux, Solaris e outras plataformas que suportam um JVM (Java Virtual Machine).

O NetBeans utiliza componentes, também conhecidos como módulos, para permitir o desenvolvimento de *software*. NetBeans instala dinamicamente módulos e permite aos utilizadores fazerem o *download* de recursos atualizados e upgrades digitalmente autenticados. Os Módulos do NetBeans IDE incluem NetBeans Profiler, uma interface gráfica do utilizador (GUI) e NetBeans JavaScript Editor.

[39] ***Fontes***: https://netbeans.org/about/index.html

5.2.3. *Eclipse IDE*

Figura 5.2-2 Logotipo Eclipse IDE

Segundo (Chen e Marx, 2005), Eclipse IDE é uma plataforma centrada com várias ferramentas de desenvolvimento integradas dentro. É um IDE universal "para tudo e nada em particular." Eclipse é um IDE desenvolvido em Java, baseado no modelo open-source de desenvolvimento de *software*, é o IDE líder de mercado e foi formado por um consórcio liderado pela IBM. Apesar de ser um IDE escrito em Java, a biblioteca gráfica utilizada pelo Eclipse, chamada de SWT, usa componentes nativos do sistema operativo. Logo recomenda-se o uso da versão correspondente ao sistema operativo usado no computador.

Este IDE baseia-se na utilização de *plugins*. Existem centenas de plugins gratuitos, entre eles: plugins para gerar diagramas UML, suporte a servidores de aplicação, visualizadores de base de dados e muitos outros. Com o uso de *plugins*, pode ser usado não só para desenvolver em Java, mas também em C / C++, PHP, ColdFusion e Python.

5.2.4. *Criação de um novo projeto para utilização da biblioteca no Netbeans IDE*

Encontra-se em anexo, detalhado o processo de criação de um novo projeto java de forma simples e rápida, para o controlo de um robot com placa Arduino.

No exemplo apresentado foi utilizado o NetBeans IDE e o robot farrusco.

5.2.5. *Criação de um novo projeto para utilização da biblioteca no Eclipse IDE*

Encontra-se em anexo, detalhado o processo de criação de um projeto equivalente ao criado com o Netbeans IDE, mas em Eclipse IDE.

5.3. **Descrição dos testes com a biblioteca educacional**

Como já referido, o contexto de utilização que foi escolhido para a avaliação da biblioteca de abstração foi o contexto educacional. Para isso foi desenvolvida uma segunda biblioteca composta por funções que permitiam a resolução de um conjunto de exercícios. Esses exercícios consistem em controlar o robot e o seu comportamento numa mesa de testes construída para o efeito.

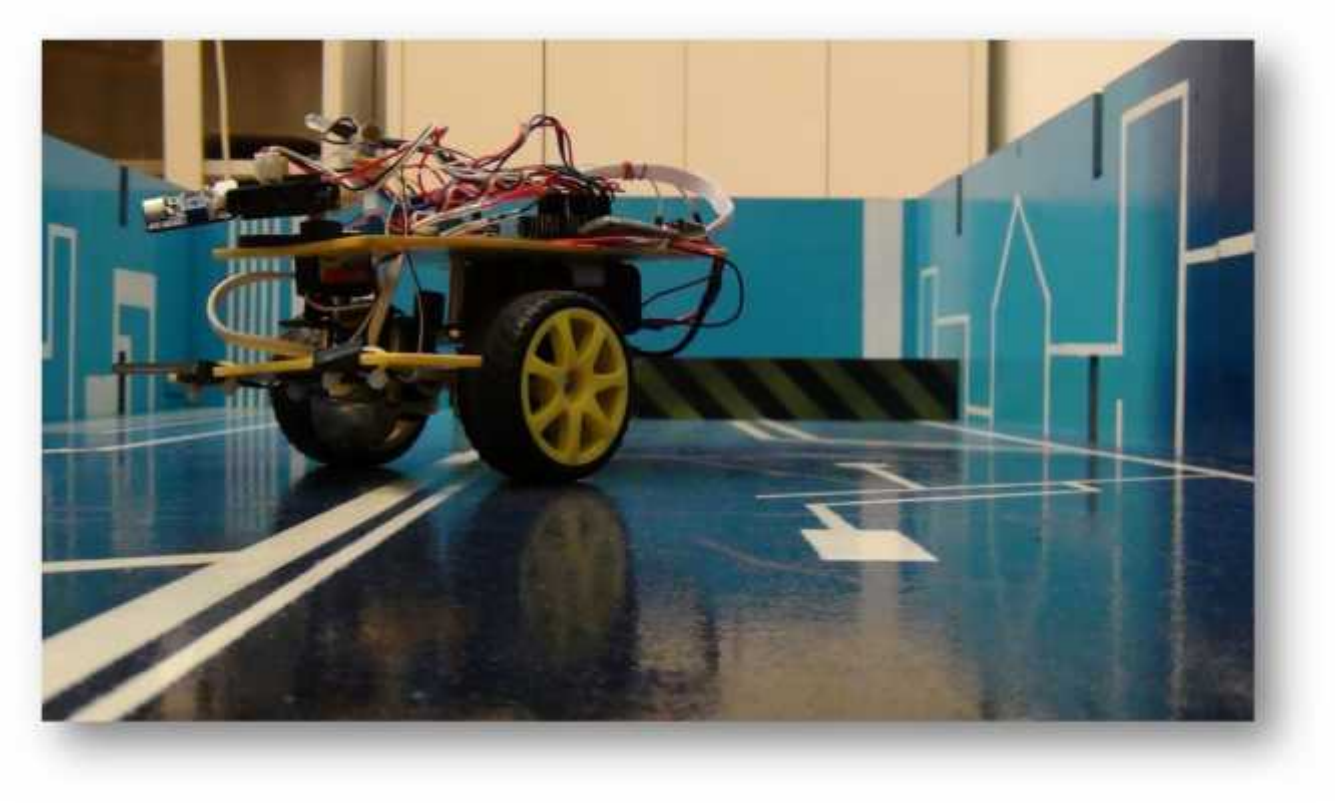


Figura 5.3-1 Robot na mesa de testes

Apresentamos nas próximas secções os testes realizados nos diferentes grupos de exercícios.

5.3.1. *Testes com os exercícios do grupo A*

Relembramos que os exercícios deste grupo servem para os utilizadores se ambientarem com o robot e a biblioteca através dos atuadores do robot. Para a resolução destes exercícios é necessário utilizar as funções contidas na classe *FuncoesACT.*

Nas resoluções dos exercícios do grupo A, são usadas as funções disponíveis sem ser necessário algoritmos complexos, as funções são usadas diretamente.

Antes de entrarmos em detalhes sobre as resoluções dos exercícios, temos de primeiro aceder as funções necessárias para as resoluções destes. Nestes exercícios são usadas funções da classe *FuncoesACT*. Para se usarem as funções dessa classe, temos de fazer o *import* da mesma e criar um objeto do tipo *FuncoesACT*, como demonstrado nas Figuras 5.3-1 e 5.3-2.

```
import ACT.FuncoesACT;
```

Figura 5.3-2 *Import* da classe *FuncoesACT*

```java
public FuncoesACT robotACT;

public ExerciciosA() {
    robotACT = new FuncoesACT();
}
```

Figura 5.3-3 Criar e instanciar o objeto do tipo *FuncoesACT*

EXERCÍCIO A1

O robot anda para a frente 3 vezes, após concluir esses movimentos aguarda 1 segundo e depois anda para trás 3 vezes.

Com da biblioteca desenvolvida para a resolução deste exercício, é necessário o código apresentado na Figura 5.3-4.

```java
public void ExercicioA1() {
    robotACT.goFront(5); // anda para a frente
    robotACT.goFront(5); // anda para a frente
    robotACT.goFront(5); // anda para a frente

    robotACT.sleep(1000); // tempo em que esta parado

    robotACT.goBack(5); // anda para tras
    robotACT.goBack(5); // anda para tras
    robotACT.goBack(5); // anda para tras
}
```

Figura 5.3-4 Resolução do exercício A1 com a biblioteca (Java)

Para a resolução deste exercício simples, sem o auxílio das bibliotecas desenvolvidas era necessário desenvolver-se o código apresentado na Figura 5.3-5.

```
int pinMotorDireitoFrente = 5;
int pinMotorDireitoTras = 3;
int pinMotorEsquerdoFrente = 6;
int pinMotorEsquerdoTras = 11;

void setup()
{
    pinMode(pinMotorDireitoFrente, OUTPUT);
    pinMode(pinMotorDireitoTras, OUTPUT);
    pinMode(pinMotorEsquerdoFrente, OUTPUT);
    pinMode(pinMotorEsquerdoTras, OUTPUT);
}

void parar(){
  analogWrite(pinMotorDireitoFrente, 0);
  analogWrite(pinMotorDireitoTras, 0);
  analogWrite(pinMotorEsquerdoFrente, 0);
  analogWrite(pinMotorEsquerdoTras, 0);
}

void andarFrente()
{
  parar();
  analogWrite(pinMotorDireitoFrente, 255);
  analogWrite(pinMotorEsquerdoFrente, 255);
  delay(1000);
  parar();
}

void andarTras(){
  parar();
  analogWrite(pinMotorDireitoTras, 255);
  analogWrite(pinMotorEsquerdoTras), 255);
  delay(1000);
  parar();
}

void loop()
{
  andarFrente();
  andarFrente();
  andarFrente();

  delay(3000);

  andarTras();
  andarTras();
  andarTras();
}
```

Figura 5.3-5 Resolução do exercício A1 sem a biblioteca (Arduíno)

Para possibilitar a resolução deste exercício com as bibliotecas foi necessário desenvolver bastante código.

No arduíno teve de se criar um *scrit* que guardava as variáveis *default* na *EEPROM* e criar métodos *get* para as poder aceder no código do robot, como explicado na secção 4.7.1. No código do robot teve de se programar os métodos para as funcionalidades em si. Este processo foi utilizado na generalidade das funções.

Demonstramos um exemplo das funções usadas no exercício A1 na Figura 5.3-6.

```
void cmdAndarFrente(int velDir, int velEsq, int valDelay, int valDelayStop){
  cmdParar(0);
  analogWrite(getPinMotorDireitoFrente(), velDir);
  analogWrite(getPinMotorEsquerdoFrente(), velEsq);
  delay(valDelay/2);
  analogWrite(getPinMotorDireitoFrente(), velDir/1.5);
  analogWrite(getPinMotorEsquerdoFrente(), velEsq/1.5);
  delay(valDelay/2);
  cmdParar(valDelayStop);

  enviarMsg(velDir, velEsq, valDelay, valDelayStop, ANDAR_FRENTE);
}

void cmdAndarTras(int velDir, int velEsq, int valDelay, int valDelayStop){
  cmdParar(0);
  analogWrite(getPinMotorDireitoTras(), velDir);
  analogWrite(getPinMotorEsquerdoTras(), velEsq);
  delay(valDelay/2);
  analogWrite(getPinMotorDireitoTras(), velDir/1.5);
  analogWrite(getPinMotorEsquerdoTras(), velEsq/1.5);
  delay(valDelay/2);
  cmdParar(valDelayStop);

  enviarMsg(velDir, velEsq, valDelay, valDelayStop, ANDAR_TRAS);
}
```

Figura 5.3-6 Funções de mover o robot em frente e para trás no Arduíno

Naturalmente, para o código funcionar é necessária toda a estrutura de código que processa a comunicação entre o robot e o IDE java, desde o estabelecimento da comunicação ate á gestão das funções no Arduíno, que se encontra explicado detalhadamente na secção 4.4. Também aqui, todo o processo é igual para todas as funções.

Por fim é essencial a criação das funções no lado do java. Demonstramos nas Figuras 5.3-7, 5.3-9 e 5.3-10 o código de algumas dessas funções.

Para a função *sleep*, o código java encontra-se descrito na biblioteca de abstração, e na biblioteca educacional apenas se encontra uma chamada ao método da biblioteca de abstração.

```
public void sleep(int delay) {
    // valida todas as variaveis recebidas para garantir que estao todas dentro dos limites admissiveis
    if (delay >= 0 && delay <= 10000) {

        // caso todas as variaveis estejam dentro dos limites, cria-se uma lista e coloca-se la todas as variaves,
        // garantindo que todas estao no formato de uma string
        List<String> variaveis = new ArrayList<>();
        variaveis.add(String.valueOf(delay));

        // envia uma mensagem para o robot com o comando correspondente
        // utiliza o metodo construirMensagens para formular a mensagem que é enviada para o robot
        comunicacao.enviarMSG(msg.construirMensagens("delay", variaveis), 0);
        comunicacao.receberMSG(); // devolve para o utilizador a resposta do robot

        // caso as variaveis não estejam dentro dos limites, sao emitidas mensagens de erro para o utilizador
    } else {
        msg.mensagemErro("Erros:\n\t* Os delays têm de estar entre 0 e 10000!!!");
    }
}
```

Figura 5.3-7 Método sleep na Biblioteca de abstração

As funções responsáveis por mover o robot em frente e para trás são descritas com valores de default na biblioteca de abstração. São depois usadas na construção das funções de mover o robot em frente e para trás x centímetros recebidos por parâmetro na biblioteca de abstração.

Figura 5.3-8 Robot executa método goFront

```
public String goFront() {
    comunicacao.enviarMSG("AF", 0); // envia uma mensagem para o robot com o comando correspondente
    return receiveMSG(); // devolve para o utilizador a resposta do robot
}
```

Figura 5.3-9 Método goFront na Biblioteca de abstração

```
public String goFront(int distancia) {
    String retorno = "Erros: ";
    boolean existeErros = true;

    if ((distancia % 5) == 0) {
        existeErros = false;

        String a = "";
        for (int i = 0; i < (distancia / 5); i++) {
            a = instanciaRobot.goFront();
            sleep(200);
        }
        retorno = "O robot deslocou-se " + (distancia / 5) + " vezes:\n\t" + a;
    } else {
        retorno += "\n\t* A distancia (em centimetros) tem de ser um multiplo de 5";
    }
    // verifica se existem erros nas variaveis, caso exista é emitida uma mensagem de erro para o utilizador,
    // caso não exista é devolvido ao utilizador a mensagem recebida pelo robot
    return (existeErros ? instanciaRobot.getMensagem().mensagemErro(retorno) : retorno);
}
```

Figura 5.3-10 Método goFront na Biblioteca Educacional

O método *goBack* na biblioteca educacional foi desenvolvido seguindo o mesmo raciocínio, mas obviamente recorrendo ao método *goBack* da biblioteca de abstração.

Figura 5.3-11 Robot executa método goBack

EXERCÍCIO A2

O robot olha para a direita, olha para a esquerda e volta a olhar para a frente. (O servo começa centrado).

Apresentamos a resolução desse exercício com a utilização das funções criadas nas bibliotecas na Figura 5.3-11.

```
public void ExercicioA2() {
    robotACT.lookRight(70); // olha para a direita
    robotACT.sleep(250); // tempo em que esta parado
    robotACT.lookLeft(140); // olha para a esquerda
    robotACT.sleep(250); // tempo em que esta parado
    robotACT.lookRight(70); // volta a olhar em frente
}
```

Figura 5.3-12 Resolução do exercício A2 com a biblioteca (Java)

Para resolver este exercício sem as bibliotecas:

```
#include <Servo.h>

Servo servo;
int pinServo = 9;

void setup()
{
  servo.attach(pinServo); // servo attach pin
  servo.write(90); // centra o servo
  delay(250);
}

void loop()
{
  servo.write(20);
  delay(250);

  servo.write(160);
  delay(250);

  servo.write(90);
  delay(250);
}
```

Figura 5.3-13 Resolução do exercício A2 sem a biblioteca (Arduíno)

Para a resolução deste exercício com as bibliotecas, criou-se um método no arduíno que move o servo para uma posição recebida como parâmetro desse método, juntamente com o *delay* da função. Demonstrado na Figura 5.13-14.

```
void moverServo(int angulo, int valDelay){
  servo.write(angulo);
  delay(valDelay);

  ImpNum(servo.read());
}
```

Figura 5.3-14 Função moverServo no Arduino

Do lado do java, na biblioteca de abstração foi desenvolvido o método *lookToAngle*, que recebe os mesmos parâmetros que o método *moverServo* no ardido, mas aqui faz todas as verificações necessárias para garantir a integridade da função e dos seus parâmetros. Apresentamos o método lookToAngle na Figura 5.3-15.

```
public String lookToAngle(int angulo, int delay) {
    String retorno = "Erros: ";
    boolean existeErros = true;

    // valida todas as variaveis recebidas para garantir que estao todas dentro dos limites admissiveis
    if (delay >= 0 && delay <= 10000) {
        if (angulo >= 2 && angulo <= 178) {

            // caso todas as variaveis estejam dentro dos limites, cria-se uma lista e coloca-se la todas as variaves,
            // garantindo que todas estao no formato de uma string
            List<String> variaveis = new ArrayList<>();
            variaveis.add(String.valueOf(angulo));
            variaveis.add(String.valueOf(delay));

            //System.out.println("angulo = " + angulo + " delay = " + delay);
            // envia uma mensagem para o robot com o comando correspondente
            // utiliza o metodo construirMensagens para formular a mensagem que é enviada para o robot
            comunicacao.enviarMSG(msg.construirMensagens("movS", variaveis), 0);

            retorno = "O servo está agora no agulo " + comunicacao.receberMSG(); // devolve para o utilizador a resposta do robot editada
            existeErros = false;

            // caso as variaveis não estejam dentro dos limites, sao emitidas mensagens de erro para o utilizador
        } else {
            retorno += "\n\t* O angulo tem de estar entre 2° e 178°!!!";
        }
    } else {
        retorno += "\n\t* Os delays têm de estar entre 0 e 10000!!!";
    }

    // verifica se existem erros nas variaveis, caso exista é emitida uma mensagem de erro para o utilizador,
    // caso não exista é devolvido ao utilizador a mensagem recebida pelo robot
    return (existeErros ? msg.mensagemErro(retorno) : retorno);
}
```

Figura 5.3-15 Método *lookToAngle* na Biblioteca de Abstração

Na biblioteca Educacional, foram criados 3 métodos que utilizam o método *lookToAngle*. Esses métodos são o *lookLeft*, *lookRight* e o *lookFront*. Apresentámos dois desses métodos nas Figuras 5.3-16 e 5.3-17.

```
public String lookLeft(int angulo) {
    String retorno = "Erros: ";
    boolean existeErros = true;

    if (angulo >= 1 && angulo <= 176) {
        int anguloAtual = getServoAngle();
        if ((anguloAtual + angulo) <= 178) {
            retorno = instanciaRobot.lookToAngle((anguloAtual + angulo), angulo * 5);
            System.out.println("\tturnLeft - angulo = " + anguloAtual + " incremento = " + angulo + "retorno = " + retorno);
            existeErros = false;
        } else {
            System.out.println("\tturnLeft - angulo = " + anguloAtual + " incremento = " + angulo);
            retorno += "\n\t* A posição final do servo (" + (anguloAtual + angulo) + ") nao pode exceder 178°!!!";
        }
    } else {
        retorno += "\n\t* O angulo tem de estar entre 1° e 176°!!!";
    }

    return (existeErros ? instanciaRobot.getMensagem().mensagemErro(retorno) : retorno);
}
```

Figura 5.3-16 Método lookLeft na Biblioteca Educacional

```
public String lookFront() {
    return instanciaRobot.lookToAngle(90, 300);
}
```

Figura 5.3-17 Método lookFront na Biblioteca Educacional

EXERCÍCIO A3

O robot buzina 3 vezes.

```
public void ExercicioA3() {
    robotACT.soundOn(500); // apita o buzzer com uma frequencia de 500Hz
    robotACT.sleep(1000); // tempo em que esta a apitar
    robotACT.soundOff(); // desliga o buzzer

    // apita o buzer com uma frequencia de 1000Hz durante 1000 milisegundos
    robotACT.sound(1000, 1000);

    // apita o buzer com uma frequencia de 1500Hz durante 1000 milisegundos
    robotACT.sound(1500, 1000);
}
```

Figura 5.3-18 Resolução do Exercício A3 com a biblioteca (Java)

Para a resolução deste exercício, sem o auxílio das bibliotecas desenvolvidas era necessário desenvolver-se o código apresentado na Figura 5.3-19.

```
int pinBuzzer = 10;

void setup()
{
  pinMode(pinBuzzer, OUTPUT);
}

void loop()
{
  tone(pinBuzzer, 500);
  delay(1000);
  noTone(pinBuzzer);
  delay(250);

  tone(pinBuzzer, 500);
  delay(1000);
  noTone(pinBuzzer);
  delay(250);

  tone(pinBuzzer, 500);
  delay(1000);
  noTone(pinBuzzer);

  delay(5000);
}
```

Figura 5.3-19 Resolução do Exercício A3 sem a biblioteca (Arduíno)

Para a implementação deste exercício foram criados alguns métodos no Arduíno que controlam o *buzzer* do robot. Demonstramos alguns desses métodos nas Figura 5.3-20 e 5.3-21. Na Figura 5.3-20 temos o método *ligarBuzzer* que apita uma frequência durante um *delay*, que recebe como parâmetros e o método *loopBuzzerOn* que coloca o *buzzer* a apitar continuamente uma frequência recebida em parâmetro até que lhe enviem um comado para parar o *buzzer*.

```
void setupBuzzer(){
  pinMode(getPinBuzzer(), OUTPUT);
}

void ligarBuzzer(int frequencia, int valDelay){
  tone(getPinBuzzer(), frequencia);
  delay(valDelay);
  Serial.print("justBuzzed*");
}

void loopBuzzerOn(int frequencia)
{
  frequenciaBuzzer = frequencia;
  buzzerOnLoop = true;
  Serial.print("isBuzzing*");
}
```

Figura 5.3-20 Métodos de control do Buzzer do Arduino – parte 1

Na Figura 5.3.20 está representado o pedaço de código que coloca o *buzzer* a apitar continuamente. Este excerto de código esta na função *loop* do Arduíno. Existe uma variável *boolean* (*buzzerOnLoop*) que é colocada a *true* no método *loopBuzzerOn*, neste pedaço de código quando essa variável se encontra a *true* começa a apitar e para quando esta a *false*.

```
if(buzzerOnLoop){
  tone(getPinBuzzer(), frequenciaBuzzer);
}else{
  noTone(getPinBuzzer());
}
```

Figura 5.3-21 Métodos de controlo do Buzzer do Arduíno – parte 2

Na biblioteca de abstração estão declarados três métodos. O método *buzz*, *buzzOn* e *buzzOff*. O método *buzz* é o equivalente ao método *ligarBuzzer* no arduíno, o método *buzzOn* é o chama o método *loopBuzzerOn* e o *buzzOff* para o *buzzer*. Demonstramos o método *buzzOn* na Figura 5.3-21. O método recebe como parâmetro a frequência, valida-a e envia um comando para o arduíno para executar a função equivalente. Os restantes são de funcionamento semelhante.

Figura 5.3-22 Método buzzOn na biblioteca de Abstração

Na biblioteca educacional, estes métodos apenas fazem chamadas dos métodos equivalentes na biblioteca de abstração.

EXERCÍCIO A4

O robot pisca o led 3 vezes.

A resolução deste exercício com utilização das funções das bibliotecas está representado na Figura 5.3-23.

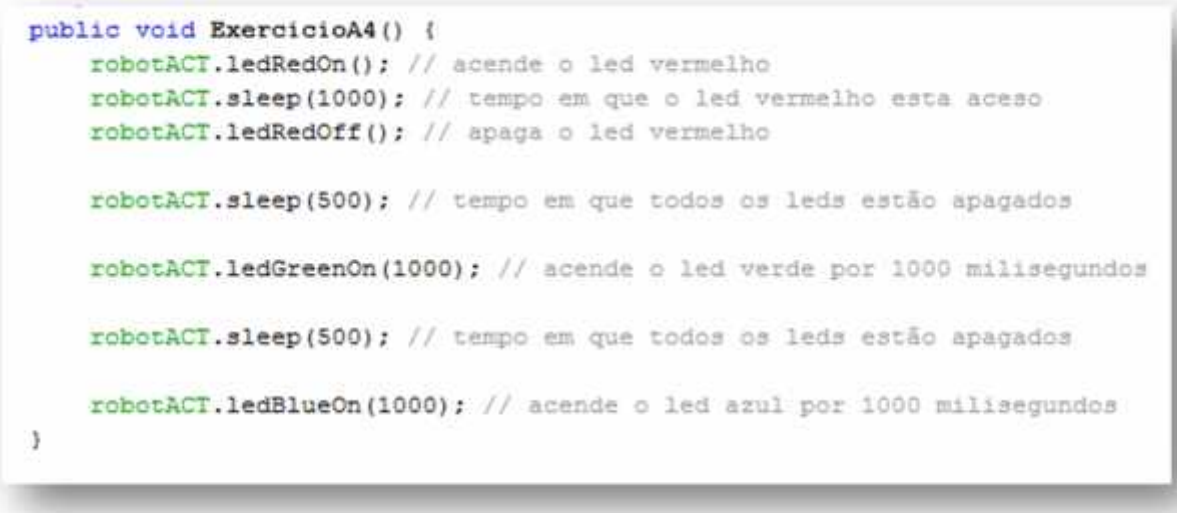


Figura 5.3-23 Resolução do Exercício A4 com a biblioteca (Java)

Para a resolução deste exercício, sem o auxílio das bibliotecas desenvolvidas era necessário desenvolver-se o código apresentado na Figura 5.3-24.

```
int pinLedRed = 8;
int pinLedGreen = 12;
int pinLedBlue = 13;

void setup()
{
  pinMode(pinLedRed, OUTPUT);
  pinMode(pinLedGreen, OUTPUT);
  pinMode(pinLedBlue, OUTPUT);
}

void loop()
{
  digitalWrite(pinLedRed, HIGH);
  delay(1000);
  digitalWrite(pinLedRed, LOW);

  delay(1000);

  digitalWrite(pinLedGreen, HIGH);
  delay(1000);
  digitalWrite(pinLedGreen, LOW);

  delay(1000);

  digitalWrite(pinLedBlue, HIGH);
  delay(1000);
  digitalWrite(pinLedBlue, LOW);

  delay(5000);
}
```

Figura 5.3-24 Resolução do Exercício A4 sem a biblioteca (Arduíno)

No arduíno foi desenvolvido uma função para controlar os leds. Esta função é uma função genérica e serve para todos os leds (*Red*, *Green* e *Blue*). Recebe como parâmetros o pin do led que vai controlar, o *val1* que corresponde a se acende ou apaga o led, o *valDelay* que indica quanto tempo o led se mantém no estado de *val1*, o *val2* que indica se o led se vai manter aceso ou apagado após a conclusão do método e a *msg* que é o que é retornado para o IDE.

```
void setupLeds(){
  pinMode(getPinLedRed(), OUTPUT);
  pinMode(getPinLedGreen(), OUTPUT);
  pinMode(getPinLedBlue(), OUTPUT);
}

void controladorLed(int pinLed, int val1, int valDelay, int val2, String msg){
  digitalWrite(pinLed, val1);
  delay(valDelay);
  digitalWrite(pinLed, val2);

  Serial.print(msg);
}
```

Figura 5.3-25 Método para controlar os leds

Apesentámos um exemplo de um método de controlo de leds da biblioteca de abstração na Figura 5.3-26.

```
public void ledGreenOn(int delay) {
    // valida todas as variaveis recebidas para garantir que estao todas dentro dos limites admissiveis
    if (delay >= 0 && delay <= 10000) {

        // caso todas as variaveis estejam dentro dos limites, cria-se uma lista e coloca-se la todas as variaves,
        // garantindo que todas estao no formato de uma string
        List<String> variaveis = new ArrayList<>();
        variaveis.add(String.valueOf(delay));

        // envia uma mensagem para o robot com o comando correspondente
        // utiliza o metodo construirMensagens para formular a mensagem que é enviada para o robot
        comunicacao.enviarMSG(msg.construirMensagens("GreenOnOff", variaveis), 0);
        comunicacao.receberMSG(); // devolve para o utilizador a resposta do robot

        // caso as variaveis não estejam dentro dos limites, sao emitidas mensagens de erro para o utilizador
    } else {
        msg.mensagemErro("Erros:\n\t* Os delays têm de entre 0 e 10000!!!");
    }
}
```

Figura 5.3-26 Método *ledGreenOn* da biblioteca de abstração

Na biblioteca educacional, os métodos que controlam os leds consistem apenas em chamadas para os métodos equivalentes na biblioteca de abstração.

EXERCÍCIO A5

O robot vira duas vezes para a direita 180º e volta a virar 2 vezes para a esquerda 180º. Entre cada movimento do robot, este espera 3 segundos.

Apresentamos uma possível resolução na Figura 5.3-25.

```
public void ExercicioA5() {
    robotACT.turnRight(5); // o robo roda 180° para a direita
    robotACT.sleep(3000); // fica parado 3000 milisegundos

    robotACT.turnRight(5); // o robo roda 180° para a direita
    robotACT.sleep(3000); // fica parado 3000 milisegundos

    robotACT.turnLeft(5); // o robo roda 180° para a esquerda
    robotACT.sleep(3000); // fica parado 3000 milisegundos

    robotACT.turnLeft(5); // o robo roda 180° para a esquerda
}
```

Figura 5.3-27 Resolução do Exercício A4 com a biblioteca (Java)

Para a resolução deste exercício, sem o auxílio das bibliotecas desenvolvidas era necessário desenvolver-se o código apresentado nas Figuras 5.3-28 e 5.3-29.

```
int pinMotorDireitoFrente = 5;
int pinMotorDireitoTras = 3;
int pinMotorEsquerdoFrente = 6;
int pinMotorEsquerdoTras = 11;

void setup()
{
    pinMode(pinMotorDireitoFrente, OUTPUT);
    pinMode(pinMotorDireitoTras, OUTPUT);
    pinMode(pinMotorEsquerdoFrente, OUTPUT);
    pinMode(pinMotorEsquerdoTras, OUTPUT);
}

void parar(){
  analogWrite(pinMotorDireitoFrente, 0);
  analogWrite(pinMotorDireitoTras, 0);
  analogWrite(pinMotorEsquerdoFrente, 0);
  analogWrite(pinMotorEsquerdoTras, 0);
}
```

Figura 5.3-28 Resolução do Exercício A4 sem a biblioteca (Arduíno) – parte 1

```
void girarEsquerda(){
  parar();
  analogWrite(pinMotorDireitoFrente, 255);
  analogWrite(pinMotorEsquerdoTras, 255);
  delay(600);
  parar();
}
void girarDireita(){
  parar();
  analogWrite(pinMotorEsquerdoFrente, 255);
  analogWrite(pinMotorDireitoTras, 255);
  delay(600);
  parar();
}

void loop()
{
  girarEsquerda();
  delay(1000);
  girarEsquerda();
  delay(1000);
  girarDireita();
  delay(1000);
  girarDireita();

  delay(5000);
}
```

Figura 5.3-29 Resolução do Exercício A4 sem a biblioteca (Arduíno) – parte 1

Apresentamos o método responsável por virar o robot para a direita no Arduíno.

```
void cmdGirarDireita(int velDir, int velEsq, int valDelay, int valDelayStop){
  cmdParar(0);
  analogWrite(getPinMotorEsquerdoFrente(), velEsq);
  analogWrite(getPinMotorDireitoTras(), velDir);
  delay(valDelay);
  cmdParar(valDelayStop);

  enviarMsg(velDir, velEsq, valDelay, valDelayStop, GIRAR_DIREITA);
}
```

Figura 5.3-30 Método *cmdGirarDireita* no Arduíno

Na biblioteca de abstração o método para virar o robot a direita, apenas consiste num método que envia um comando para o Arduíno para acionar a função equivalente. Esta função vira aproximadamente 5º.

```
public String turnRight() {
    comunicacao.enviarMSG("GD", 0); // envia uma mensagem para o robot com o comando correspondente
    return receiveMSG(); // devolve para o utilizador a resposta do robot
}
```

Figura 5.3-31 Método turnRight na biblioteca de abstração

Na biblioteca educacional, foi criado um método que usa o método *turnRight* para criar o seu próprio método que recebe o angulo que vai virar como parâmetro.

```
public String turnRight(int angulo) {
    String retorno = "Erros: ";
    boolean existeErros = true;

    if ((angulo % 5) == 0) {
        if (angulo >= 5 && angulo <= 360) {
            existeErros = false;

            String a = "";
            for (int i = 0; i < (angulo / 5); i++) {
                a = instanciaRobot.turnRight();
                sleep(200);
            }
            retorno = "O robot deslocou-se " + (angulo / 5) + " vezes:\n\t" + a;
        } else {
            retorno += "\n\t* O angulo tem de estar entre 5 e 360!!!";
        }
    } else {
        retorno += "\n\t* O angulo (em graus) tem de ser um multiplo de 5!!!";
    }
    // verifica se existem erros nas variaveis, caso exista é emitida uma mensagem de erro para o utilizador,
    // caso não exista é devolvido ao utilizador a mensagem recebida pelo robot
    return (existeErros ? instanciaRobot.getMensagem().mensagemErro(retorno) : retorno);
}
```

Figura 5.3-32 Método *turnRight* na biblioteca educacional

Verifica-se que os testes realizados com os exercícios deste grupo foram bem-sucedidos provando que as funções da biblioteca desenvolvida estão funcionais, permitindo ao utilizador controlar todos os atuadores do robot.

5.3.2. *Testes com os exercícios do grupo B*

Os exercícios efetuados neste grupo servem para iniciar a interação do robot com o meio ambiente através do controlo dos sensores do robot. Para a resolução destes exercícios é necessário utilizar as funções contidas nas classes *FuncoesACT e FuncoesREACT.* Para se usarem as funções dessas classes, temos de fazer o *import* delas e criar um objeto de cada tipo, como demonstrado nas Figuras 5.3-33 e 5.3-34.

```
import ACT.FuncoesACT;
import REACT.FuncoesREACT;
```

Figura 5.3-33 Import das classes *FuncoesACT* e *FuncoesREACT.*

```
public FuncoesACT robotACT;
public FuncoesREACT robotREACT;

public ExerciciosB() {
    robotACT = new FuncoesACT();
    robotREACT = new FuncoesREACT();
}
```

Figura 5.3-34 Criar e instanciar os objetos do tipo *FuncoesACT e FuncoesREACT.*

EXERCÍCIO B3

Se detetar um obstáculo no sensor de IR, o robot apita 3 vezes.

Como possível resolução deste exercício, apresentamos o código da Figura 5.3-35

```
public void ExercicioB3() {
    if (robotREACT.infraRedLevel() >= 4) { // veifica o nivel do sensor
        robotACT.sound(500, 1000);// apita o buzer com uma frequencia de 500Hz durante 1000 milisegundos
        robotACT.sound(1000, 1000);// apita o buzer com uma frequencia de 1000Hz durante 1000 milisegundos

        //robotACT.sound(1500, 1000);// apita o buzer com uma frequencia de 1500Hz durante 1000 milisegundos

        robotACT.soundOn(1500); // apita o buzzer com uma frequencia de 1500Hz
        robotACT.sleep(1000); // tempo em que esta a apitar
        robotACT.soundOff(); // desliga o buzzer

    }
}
```

Figura 5.3-35 Resolução do exercício B3 com a biblioteca (Java)

Para a resolução deste exercício, sem o auxílio das bibliotecas desenvolvidas era necessário desenvolver-se o código apresentado na Figura 5.3-36.

```
int pinBuzzer = 10;
int pinIR = 1;

void setup()
{
  pinMode(pinBuzzer, OUTPUT);
}

void loop()
{
  if(analogRead(pinIR) > 300)
  {
    tone(pinBuzzer, 500);
    delay(1000);
    noTone(pinBuzzer);
    delay(250);

    tone(pinBuzzer, 500);
    delay(1000);
    noTone(pinBuzzer);
    delay(250);

    tone(pinBuzzer, 500);
    delay(1000);
    noTone(pinBuzzer);
  }
}
```

Figura 5.3-36 Resolução do exercício B3 sem a biblioteca (Arduino)

No Arduíno foram desenvolvidas algumas funções para a leitura do sensor de IR. Como funções principais temos a função *lerIR* que devolve o valor sem tratamento do sensor, e temos a função *getNivelIR* que devolve o nível equivalente ao valor do sensor, consoante comparação com os valores de referência guardados *EEPROM*.

Apresentamos as funções do sensor de IR na Figura 5.3-37.

```
void lerIR(){
  ImpNum(getValorIR());
}

int getValorIR(){
  int val = analogRead(getPinIR());
  delay(40);
  return val;
}

void getNivelIR(){
  int val = getValorIR();
  int nivel1 = word(EEPROM.read(36), EEPROM.read(37));
  int nivel2 = word(EEPROM.read(38), EEPROM.read(39));
  int nivel3 = word(EEPROM.read(40), EEPROM.read(41));
  int nivel4 = word(EEPROM.read(42), EEPROM.read(43));
  int nivel5 = word(EEPROM.read(44), EEPROM.read(45));

  if(val >= 0 && val <= nivel1)
    ImpNum(1);
  else if(val >= (nivel1 + 1) && val <= nivel2)
    ImpNum(2);
  else if(val >= (nivel2 + 1) && val <= nivel3)
    ImpNum(3);
  else if(val >= (nivel3 + 1) && val <= nivel4)
    ImpNum(4);
  else if(val >= (nivel4 + 1) && val <= nivel5)
    ImpNum(5);

}
```

Figura 5.3-37 Funções de leitura do sensor de IR no Arduíno

Na biblioteca de abstração temos os métodos equivalentes. Apresentamos um deles na Figura 5.3-38.

```
public int infraRedLevel() {
    comunicacao.enviarMSG("getNIR", 0); // envia uma mensagem para o robot com o comando correspondente
    return Integer.parseInt(comunicacao.receberMSG()); // devolve para o utilizador a resposta do robot no formato inteiro
}
```

Figura 5.3-38 Método de leitura do nível do sensor na biblioteca de abstração

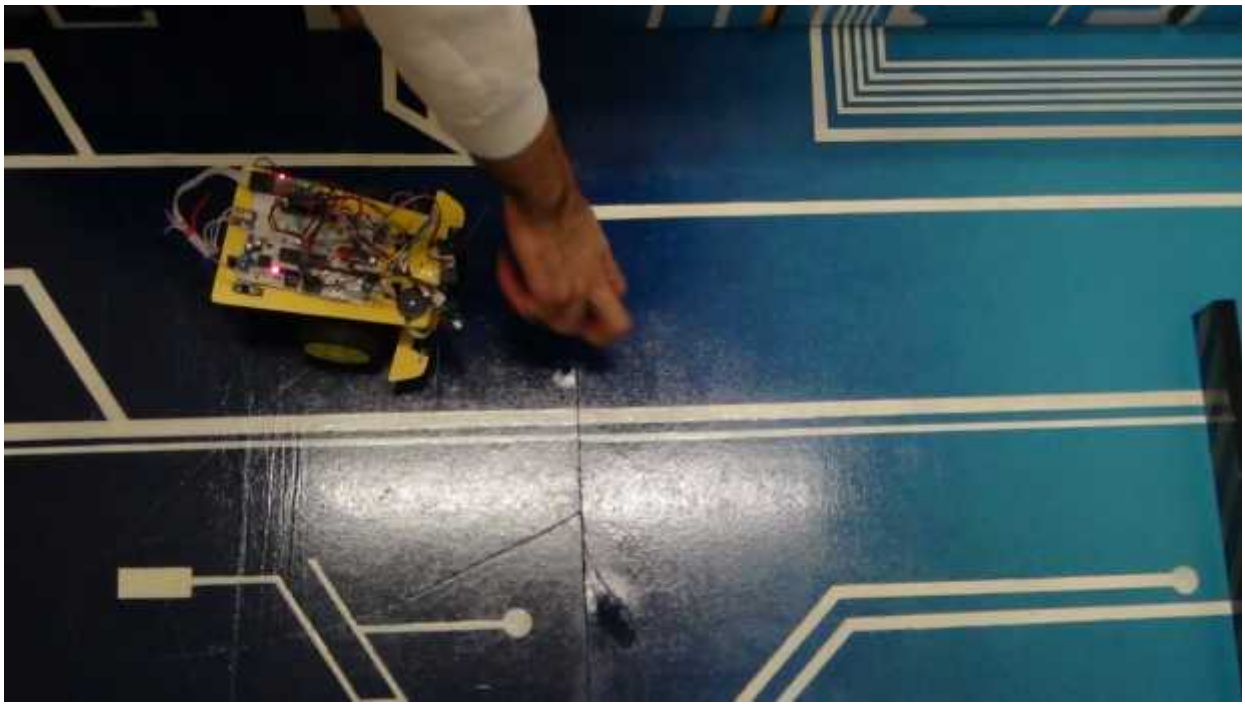


Figura 5.3-39 Robot a interagir com leituras do sensor de IR

EXERCÍCIO B4

Se detetar som, o robot apita 3 vezes.

Apresentamos uma possível resolução para este exercício na Figura 5.3-40.

```
public void ExercicioB4() {
    if (robotREACT.soundLevel() >= 4) { // veifica o nivel do sensor
        robotACT.ledBlueOn(1000); // acende o led azul durante 1000 milisegundos
        robotACT.sleep(500); // tempo em que esta a dormir
        robotACT.ledBlueOn(1000); // acende o led azul durante 1000 milisegundos
        robotACT.sleep(500); // tempo em que esta a dormir
        robotACT.ledBlueOn(1000); // acende o led azul durante 1000 milisegundos

        /*
         robotACT.ledBlueOn(); // acende o led azul
         robotACT.sleep(3000); // tempo que o led esta aceso
         robotACT.ledBlueOff(); // apaga o led azul
         */
    }
}
```

Figura 5.3-40 Resolução do exercício B4 com a biblioteca (Java)

Para a resolução deste exercício, sem o auxílio das bibliotecas desenvolvidas era necessário desenvolver-se o código apresentado na Figura 5.3-41.

```
int pinLedRed = 8;
int pinLedGreen = 12;
int pinLedBlue = 13;
int pinSom = 3;

const int numberOfSamples = 64;
long sinal;

void setup()
{
  pinMode(pinLedRed, OUTPUT);
  pinMode(pinLedGreen, OUTPUT);
  pinMode(pinLedBlue, OUTPUT);

  digitalWrite(pinSom, HIGH);
}

long valorSom(){
  long soma = 0;

  for (int i = 0; i < numberOfSamples; i++) {
    sinal = analogRead(pinSom) >> 3;
    sinal *= sinal;
    soma += sinal;
  }

  return soma;
}
void loop()
{
  if(valorSom() > 3200)
  {
    digitalWrite(pinLedRed, HIGH);
    delay(1000);
    digitalWrite(pinLedRed, LOW);

    delay(1000);

    digitalWrite(pinLedGreen, HIGH);
    delay(1000);
    digitalWrite(pinLedGreen, LOW);

    delay(1000);

    digitalWrite(pinLedBlue, HIGH);
    delay(1000);
    digitalWrite(pinLedBlue, LOW);
  }
}
```

Figura 5.3-41 Resolução do exercício B4 sem a biblioteca (Arduino)

Apresentamos os métodos criados para a leitura do sensor de som na Figura 5.3-42.

```
long getValorSom(){
  long soma = 0;

  for (int i = 0; i < numberOfSamples; i++) {
    //sinal = (analogRead(pinSom) - middleValue) >> 3;
    sinal = analogRead(pinSom) >> 3;
    sinal *= sinal;
    soma += sinal;
  }

  /*
  if (soma > threshold){
    digitalWrite(getPinLedRed(), HIGH);
  }else{
    digitalWrite(getPinLedRed(), LOW);
  }
  */

  return soma;
}

void getNivelSom(){
  int val = getValorSom();
  int nivel1 = word(EEPROM.read(76), EEPROM.read(77));
  int nivel2 = word(EEPROM.read(78), EEPROM.read(79));
  int nivel3 = word(EEPROM.read(80), EEPROM.read(81));
  int nivel4 = word(EEPROM.read(82), EEPROM.read(83));
  int nivel5 = word(EEPROM.read(84), EEPROM.read(85));

  if(val >= 0 && val <= nivel1)
    ImpNum(1);
  else if(val >= (nivel1 + 1) && val <= nivel2)
    ImpNum(2);
  else if(val >= (nivel2 + 1) && val <= nivel3)
    ImpNum(3);
  else if(val >= (nivel3 + 1) && val <= nivel4)
    ImpNum(4);
  else if(val >= (nivel4 + 1) && val <= nivel5)
    ImpNum(5);
}
```

Figura 5.3-42 Métodos de leitura do sensor de som no Arduíno.

Na biblioteca de abstração temos os métodos equivalentes, que fazem apenas uma chamada ao método no Arduíno.

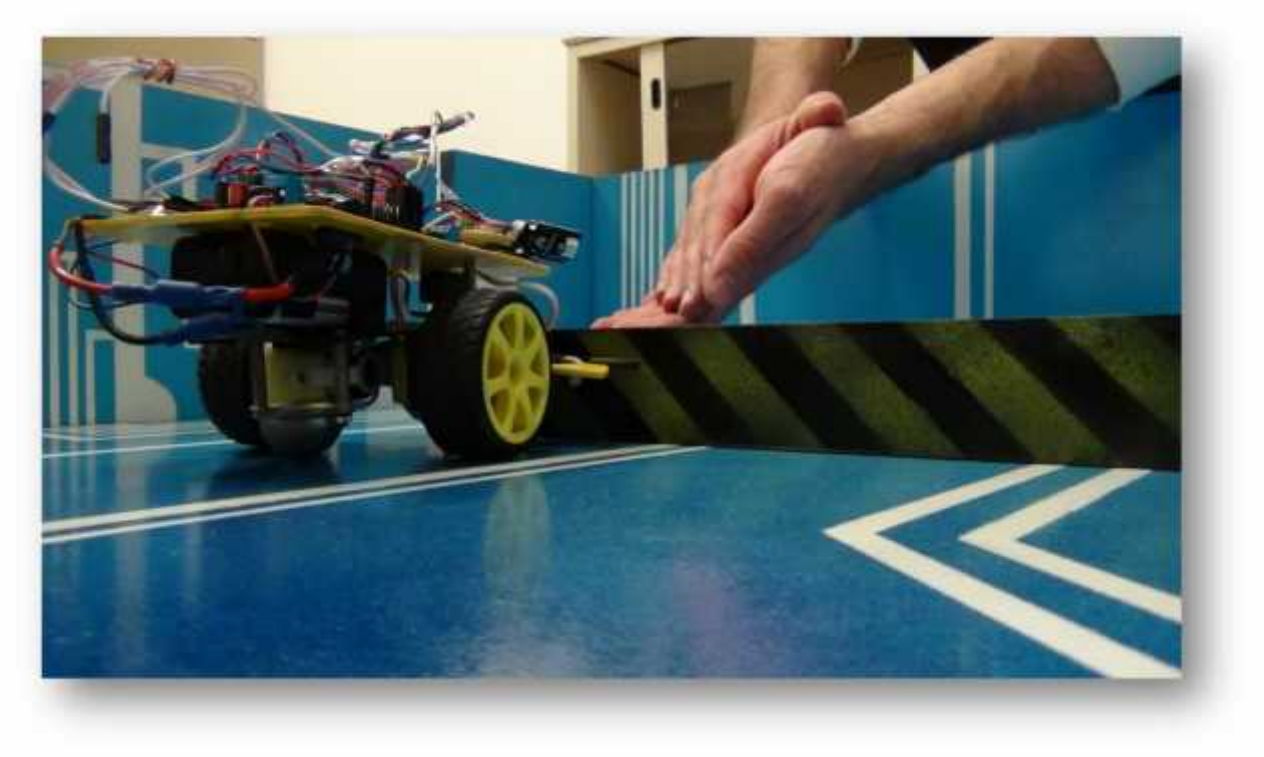


Figura 5.3-43 Robot deteta som no sensor de som

EXERCÍCIO B5

Se detetar luz, o robot avança 20cm.

Possível implementação do exercício na Figura 5.3-38.

```
public void ExercicioB5() {
    if (robotREACT.lightLevel() >= 4) { // veifica o nivel do sensor
        robotACT.goFront(20); // anda para a frente caso estejam acima de um determinado nivel
    }
}
```

Figura 5.3-44 Resolução do exercício B5 com a biblioteca (Java)

Para a resolução deste exercício, sem o auxílio das bibliotecas desenvolvidas era necessário desenvolver-se o código apresentado na Figura 5.3-45.

```
int pinMotorDireitoFrente = 5;
int pinMotorDireitoTras = 3;
int pinMotorEsquerdoFrente = 6;
int pinMotorEsquerdoTras = 11;

int pinLuz = 5;

void setup()
{
    pinMode(pinMotorDireitoFrente, OUTPUT);
    pinMode(pinMotorDireitoTras, OUTPUT);
    pinMode(pinMotorEsquerdoFrente, OUTPUT);
    pinMode(pinMotorEsquerdoTras, OUTPUT);
}

void parar(){
  analogWrite(pinMotorDireitoFrente, 0);
  analogWrite(pinMotorDireitoTras, 0);
  analogWrite(pinMotorEsquerdoFrente, 0);
  analogWrite(pinMotorEsquerdoTras, 0);
}
void andarFrente()
{
  parar();
  analogWrite(pinMotorDireitoFrente, 255);
  analogWrite(pinMotorEsquerdoFrente, 255);
  delay(1000 * 4);
  parar();
}


void loop()
{
  if(analogRead(pinLuz) < 300){
    andarFrente();
  }
}
```

Figura 5.3-45 Resolução do exercício B5 sem a biblioteca (Arduíno)

EXERCÍCIO B7

Se detetar som ou luz, o robot avança 20cm.

Possível implementação do exercício na Figura 5.3-46. Neste exercício são conjugadas as leituras de dois sensores numa instrução if.

```
public void ExercicioB7() {
    if (robotREACT.lightLevel() >= 4 || robotREACT.soundLevel() >= 3) { // verifica os niveis dos sensores
        robotACT.goFront(20); // anda para a frente caso estejam acima de um determinado nivel
    }
}
```

Figura 5.3-46 Resolução do exercício B7 com a biblioteca (Java)

Para a resolução deste exercício, sem o auxílio das bibliotecas desenvolvidas era necessário desenvolver-se o código apresentado nas Figuras 5.3-47 e 5.3-48.

```
int pinMotorDireitoFrente = 5;
int pinMotorDireitoTras = 3;
int pinMotorEsquerdoFrente = 6;
int pinMotorEsquerdoTras = 11;

int pinLuz = 5;
int pinSom = 3;

const int numberOfSamples = 64;
long sinal;

void setup()
{
  digitalWrite(pinSom, HIGH);

  pinMode(pinMotorDireitoFrente, OUTPUT);
  pinMode(pinMotorDireitoTras, OUTPUT);
  pinMode(pinMotorEsquerdoFrente, OUTPUT);
  pinMode(pinMotorEsquerdoTras, OUTPUT);

  parar();
}
```

Figura 5.3-47 Resolução do exercício B7 sem a biblioteca (Arduíno) – Parte 1

```
void parar(){
  analogWrite(pinMotorDireitoFrente, 0);
  analogWrite(pinMotorDireitoTras, 0);
  analogWrite(pinMotorEsquerdoFrente, 0);
  analogWrite(pinMotorEsquerdoTras, 0);
}

void andarFrente()
{
  parar();
  analogWrite(pinMotorDireitoFrente, 255);
  analogWrite(pinMotorEsquerdoFrente, 255);
  delay(1000);
  parar();
}

long valorSom(){
  long soma = 0;

  for (int i = 0; i < numberOfSamples; i++) {
    sinal = analogRead(pinSom) >> 3;
    sinal *= sinal;
    soma += sinal;
  }

  return soma;
}
void loop()
{
  if(valorSom() > 3200)
  {
    andarFrente();
  }

  delay(500);
}
```

Figura 5.3-48 Resolução do exercício B7 sem a biblioteca (Arduíno) – Parte 2

EXERCÍCIO B9

Se detetar obstáculo no *bumper* esquerdo, o robot olha 30º para a esquerda, aguarda 3 segundos e volta a olhar em frente. Se detetar obstáculo no *bumper* direito, o robot olha 30º para a direita, aguarda 3 segundos e volta a olhar em frente.

Possível implementação do exercício na Figura 5.3-49. Neste exercício é feita uma leitura dos valores dos *bumpers*. Depois dos *bumpers* lidos, verifica-se qual dos *bumpers* foi pressionado e executa-se as devidas instruções.

```
public void ExercicioB9() {
    // le os valores dos bumpers
    robotREACT.readBumpersAfterPressed(); //robotACT.readBumpers();

    if (robotREACT.getBumperLeftValue()) { // verifica se o bumper esquerdo esta pressionado
        robotACT.lookLeft(30); // olha 30º para a esquerda
        robotACT.sleep(3000); // tempo que o robot espera
        robotACT.lookFront(); // olha para a frente
    }

    if (robotREACT.getBumperRightValue()) { // verifica se o bumper direito esta pressionado
        robotACT.lookRight(30); // olha 30º para a direita
        robotACT.sleep(3000); // tempo que o robot espera
        robotACT.lookFront(); // olha para a frente
    }
}
```

Figura 5.3-49 Resolução do exercício B9 com a biblioteca (Java)

Para a resolução deste exercício, sem o auxílio das bibliotecas desenvolvidas era necessário desenvolver-se o código apresentado nas Figuras 5.3-50 e 5.3-51.

```
#include <Servo.h>
Servo servo;
int pinServo = 9;
int pinBumperDireito = 2;
int pinBumperEsquerdo = 4;

void setup()
{
  servo.attach(pinServo); // servo attach pin
  servo.write(90); // centra o servo
  delay(250);

  pinMode(pinBumperDireito, INPUT);
  pinMode(pinBumperEsquerdo, INPUT);
  digitalWrite(pinBumperDireito, HIGH);
  digitalWrite(pinBumperEsquerdo, HIGH);
}
```

Figura 5.3-50 Resolução do exercício B9 sem a biblioteca (Arduíno) – Parte 1

```
void loop()
{
  if(digitalRead(pinBumperDireito))
  {
    servo.write(20);
    delay(250);
    servo.write(90);
    delay(250);
  }
  else if(digitalRead(pinBumperEsquerdo))
  {
    servo.write(160);
    delay(250);
    servo.write(90);
    delay(250);
  }
}
```

Figura 5.3-51 Resolução do exercício B9 sem a biblioteca (Arduíno) – Parte 2

Foram desenvolvidos alguns métodos para a leitura dos bumpers no *Arduino*. Como mais importantes, temos os métodos *lerBumpers* e *valBumpers*. O método *lerBumpers* apenas envia para o IDE do arduíno o valor dos *bumpers* se um deles estiver pressionado. O método *valBumpers* envia o valor do *bumpers* independentemente se um *bumpers* esta pressionado ou nenhum.

Apresentamos esses métodos na figura 5.3-52.

```
void setupBumpers(){
  pinMode(getPinBumperDireito(), INPUT);
  pinMode(getPinBumperEsquerdo(), INPUT);
  digitalWrite(getPinBumperDireito(), HIGH);
  digitalWrite(getPinBumperEsquerdo(), HIGH);
}

void lerBumpers(){
  int drt = getValBumperDireito();
  int esq = getValBumperEsquerdo();

  if((drt == HIGH || esq == HIGH) && readBumpers){
    msgBumpers(esq, drt);
    readBumpers = false;
  }
}

void valBumpers(){
  msgBumpers(getValBumperEsquerdo(), digitalRead(getPinBumperDireito()));
}

void msgBumpers(int esq, int drt){
  String msg = "[";
  msg += esq;
  msg += "-";
  msg += drt;
  msg += "]*";
  Serial.print(msg);
}
```

Figura 5.3-52 Métodos de leitura dos bumpers no arduíno

Verifica-se que os testes realizados com os exercícios deste grupo foram bem-sucedidos. Foi demonstrado que através da biblioteca desenvolvida é possível desenvolver programas que façam uma recolha de dados sensoriais do meio ambiente fazendo uso de todos os sensores do robot.

5.3.3. *Testes com os exercícios do grupo C*

Nestes exercícios também são utilizadas as funções de ambas as classes *FuncoesACT* e *FuncoesREACT* e são introduzidos as instruções de repetição. A partir deste ponto já todas as funções se encontram devidamente apresentadas e os exercícios começam a aumentar o nivel de complexidade.

EXERCÍCIO C1

O robot avança em incrementos (de x centímetros) enquanto tiver luz.

Possível implementação do exercício na Figura 5.3-53.

```
public void ExercicioC1() {
    while (robotREACT.lightLevel() >= 3) { // enquanto o nivel do sensor de luz for igual ou superior ao nivel pretendido
        robotACT.goFront(15); // o robot avança
    }
}
```

Figura 5.3-53 Resolução do exercício C1 com a biblioteca (Java)

EXERCÍCIO C2

O robot roda enquanto deteta um obstáculo no *bumper* x. (Ex: Se toca no *bumper* direito, roda para a esquerda, se toca no *bumper* esquerdo, roda para direita).

Possível implementação do exercício na Figura 5.3-54.

```
public void ExercicioC2() {
    do {
        robotREACT.readBumpersAfterPressed(); // le o valor dos bumpers
        if (robotREACT.getBumperLeftValue()) { // se bumper esquerdo pressionado
            robotACT.turnLeft(90); // robot roda para a esquerda
        } else if (robotREACT.getBumperRightValue()) { // se bumper direito pressionado
            robotACT.turnRight(90); // robot roda para a direita
        }
    } while (robotREACT.getBumperLeftValue() || robotREACT.getBumperLeftValue()); //continua enquanto houver um bumper pressionado
}
```

Figura 5.3-54 Resolução do exercício C2 com a biblioteca (Java)

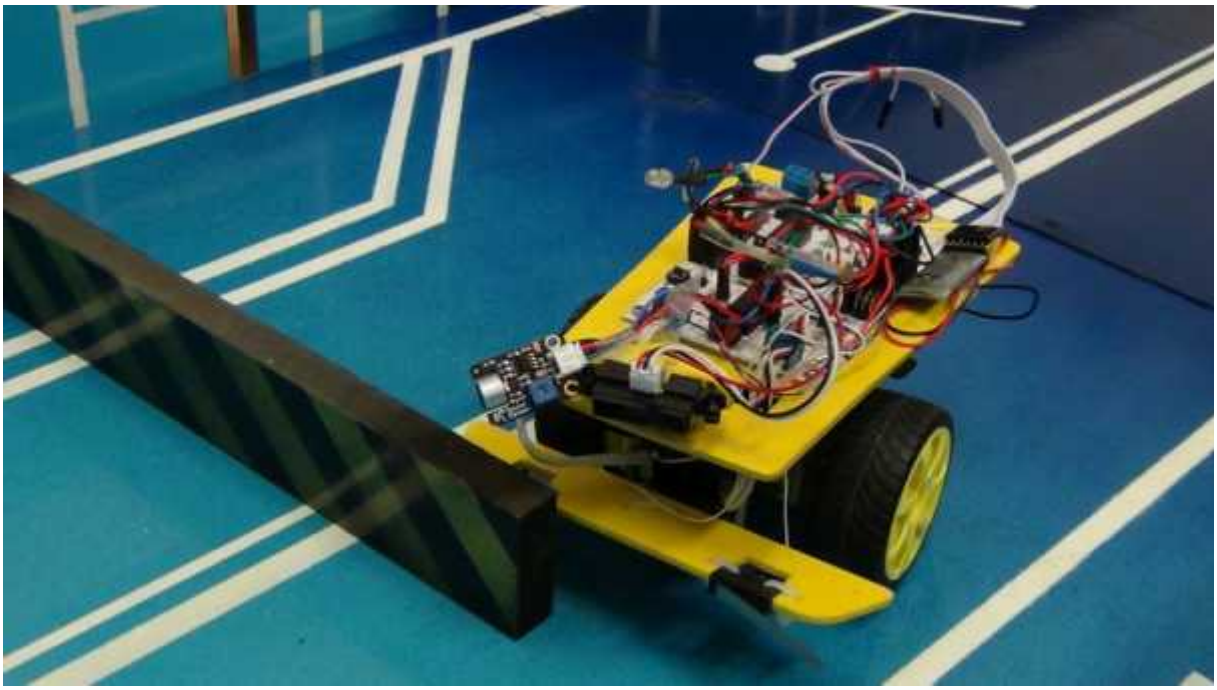


Figura 5.3-55 Robot deteta obstáculo com *bumper* direito

EXERCÍCIO C4

Enquanto luz apita alternado, se deteta um obstáculo num dos *bumpers* apita continuamente.

Possível implementação do exercício na Figura 5.3-56.

```
public void ExercicioC4() {
    int level;
    do {
        level = robotREACT.lightLevel(); // le o nivel do sensor de luz
        robotREACT.readBumpers(); // le o valor dos bumpers

        if (level >= 4) { // verifica o valor do sensor de luz
            robotACT.soundOff(); // desliga o som caso este se encontre ativo
            robotACT.sound(1500, 500); // o robot apita durante 500 milisegundos

        // verifia se algum bumper se encontra pressionado
        } else if (robotREACT.getBumperLeftValue() || robotREACT.getBumperRightValue()) {
            robotACT.soundOn(500); // o robot comeca a apitar
        }
        // continua enquanto deteta luz no sensor ou algum dos bumpers se encontrar pressionado
    } while (level >= 4 || robotREACT.getBumperLeftValue() || robotREACT.getBumperRightValue());
    robotACT.soundOff();
}
```

Figura 5.3-56 Resolução do exercício C4 com a biblioteca (Java)

EXERCÍCIO C7

O robot avança enquanto deteta luz, se detetar obstáculo com um *bumper* desvia para o lado contrário do *bumper*.

Possível implementação do exercício na Figura 5.3-57.

```
public void ExercicioC7() {
    int level;

    do {
        level = robotREACT.lightLevel(); // le o nivel do sensor de luz
        robotREACT.readBumpers(); // le o valor dos bumpers

        if (level >= 4) { // verifica o nivel do sensor
            robotACT.goFront(10); // o robot avança
        }
        if (robotREACT.getBumperLeftValue()) { // verifica se o bumper esquerdo esta pressionado
            robotACT.turnRight(90); // o robot vira para a direita
        }
        if (robotREACT.getBumperRightValue()) { // verifica se o bumper direito esta pressionado
            robotACT.turnLeft(90); // o robot vira para a esquerda
        }
        // continua enquanto o nivel do sensor de luz estiver acima de um valor ou algum dos bumpers estiver pressionado
    } while (level >= 4 || robotREACT.getBumperLeftValue() || robotREACT.getBumperRightValue());
}
```

Figura 5.3-57 Resolução do exercício C4 com a biblioteca (Java)

Verifica-se que os testes realizados com os exercícios deste grupo foram bem-sucedidos. Nestes exercícios são utilizadas as funções de ambas as classes *FuncoesACT* e *FuncoesREACT*, tirando partido de todos os atuadores e sensores disponíveis no robot.

5.3.4. *Testes com os exercícios do grupo D*

A partir deste grupo de exercícios, o grau de complexidade de resolução aumenta demonstrando que a biblioteca pode ser usada em exercícios de alta complexidade de resolução e não apenas instruções simples e diretas.

EXERCÍCIO D2

O robot buzina enquanto um obstáculo detetado no sensor de IR se encontra mais próximo do que já registado. Quando clica num dos *bumpers* faz *reset* ao máximo. (Ex: aproxima-se a mão e o robot buzina, volta-se a aproximar a mão a mesma distância e o robot já não reage. Aproxima-se a mão mais próximo e o robot volta a buzinar).

Possível implementação do exercício na Figura 5.3-58.

```
public void ExercicioD2() {
    int level; // variavel auxiliar
    do { // inicia o ciclo
        level = robotREACT.infraRedLevel(); // le o nivel do sensor
        System.out.println("level = " + level + " maxLevel = " + maxLevel);
        if (level > maxLevel) { // caso o nivel do sensor for maior que o registado anteriormente, ou seja, o obstaculo esta mais perto
            maxLevel = level; // atualiza o nivel do sensor
            robotACT.soundOn(1000); // comeca a apitar
            robotACT.sleep(1000);
            robotACT.soundOff();
        }
    } while (level > maxLevel); // continua enquanto o nivel do infra vermelhos é maior do que a maxima luz registada

    robotREACT.readBumpers(); // le os bumpers
    if (robotREACT.getBumperLeftValue() || robotREACT.getBumperRightValue()) { // verifica se algum dos bumpers esta pressionado
        maxLevel = 1; // faz reset ao valor maximo
    }
}
```

Figura 5.3-58 Resolução do exercício D2 com a biblioteca (Java)

EXERCÍCIO D3

O robot procura a origem da luz rodando a torre (sugestão: esquerda – frente – direita), após concluir a sua busca, pisca o LED e olha para o lado onde detetou a luz maior.

Possível implementação do exercício na Figura 5.3-59 e 5.3-60.

```
public void ExercicioD3() {
    robotACT.sound(1500, 500); // apita, para debug (saber que vai iniciar o movimento do servo
    int posMax = ExercicioD3_1(); // devolve o angulo do servo onde esta a maior intencidade de luz
    robotACT.sound(1500, 500); // apita para debug (saber que concluiu o movimento do servo)

    if (robotACT.getServoAngle() != posMax) { // caso o servo ainda nao esteja na posicao que regista o valor do sensor mais alto
        robotACT.lookToAngle(posMax, 200); // coloca o servo no angulo correspondente ao valor do sensor mais alto
    }

    robotACT.ledRedOn(1000); // acende o led 1segundo
}
```

Figura 5.3-59 Resolução do exercício D3 com a biblioteca (Java) – Part 1

```
// varre as posicoes do servo., devolve o angulo do servo onde deteta maior intencidade no sensor de luz
public int ExercicioD3_1() {
    int incremento = 60; // incremento que o servo vai rodar
    boolean chegouAoFim = false; // indica se o servo ja percorreu todas as posicoes ou nao
    int levelMax = 0; // guarda o valor maximo registado
    int posMax = 0; // guarda o angulo correspondente ao valor maximo registado

    int i = 0; // variavel para debug

    // (90 - incremento) olha para a direita / (90 + incremento) olha para a esquerda
    if (robotACT.getServoAngle() != (90 - incremento)) { // caso o servo ainda nao se encontre na posicao inicial (a olhar para a direita) este e colocado nessa posicao
        System.out.println("initial position = " + robotACT.lookToAngle(90 - incremento, 500)); // posiciona o servo a olhar para a direita
    }

    // vai correr as posicoes 30 - 90 - 150
    do { // inicia o ciclo
        System.out.println("i = " + i);
        int pos = robotACT.getServoAngle(); // le o angulo do servo
        System.out.println("\tpos = " + pos);
        int level = robotREACT.lightLevel(); // le o nivel do sensor
        System.out.println("\tlux level = " + level);
        System.out.println("\tlevelMax = " + levelMax);

        if (level > levelMax) { // caso o nivel do sensor seja superior ao registado anteriormente, atualiza os valores
            levelMax = level; // atualiza o valor do nivel do sensor
            posMax = pos; // atualiza o angulo correspondente ao angulo do sensor
            System.out.println("\t(level > minLevel) => posMax = " + posMax + " levelMax = " + levelMax);
        }

        if ((pos + incremento) <= 150) { // verifica se o servo ainda pode rodar mais uma vez
            System.out.println("\t\t\trobotACT.lookLeft(incremento) = " + robotACT.lookLeft(incremento)); // vira o servo 60° para a sua esquerda
        } else { // se nao encontrou a luz, nem puder virar mais uma vez
            chegouAoFim = true; //indica que chegou ao fim
            System.out.println("\n\t\tChegou ao fim / i = " + i + "\n");
        }
        i++; // incrementa mais uma iteracao
    } while (!chegouAoFim); // termina o ciclo quando encontrar a luz ou chegar ao fim da sua rotacao

    return posMax;
}
```

Figura 5.3-60 Resolução do exercício D3 com a biblioteca (Java) – Part 2

EXERCÍCIO D5

O robot encontra-se num labirinto, percorre-o e procura a saída (através do sensor IR). Desvia-se se detetar um obstáculo. Quando o robot chegar ao destino, acende-se uma luz para este saber que já concluiu o percurso.

Figura 5.3-61 Robot percorre labirinto com o sensor de IR

Possível implementação do exercício na Figura 5.3-62 e 5.3-63.

```
public void ExercicioD5() {
    while (robotRERCT.lightLevel() <= 4) { // continua enaquanto nao deteta luz

        int anguloServo = ExercicioD5_1(); // le o sensor IR, devolve 90 caso nao haja obstaculos a frente,
        // 30 caso o caminho a seguir seja a direita
        // 150 caso o caminho a seguir seja a esquerda

        if (anguloServo < 90) { // direita - o caminho a seguir é a direita, ou seja o angulo do servo com obstaculo mais longe é a direita
            robotACT.turnRight(5); // vira a direita
        } else if (anguloServo > 90) { // esquerda - o caminho a seguir é a esquerda, ou seja o angulo do servo com obstaculo mais longe é a esquerda
            robotACT.turnLeft(5); // vira a esquerda
        }

        robotACT.goFront(5); // anda em frente
    }
}
```

Figura 5.3-62 Resolução do exercício D5 com a biblioteca (Java) – Part 1

```java
// varre as posicoes no servo.. devolve o angulo do servo onde deteta menor intencidade no IR
public int ExercicioD5_1() {
    int incremento = 60; // valor que vai incrementar o servo
    int pos; // variavel para guardar a posicao do servo sem obstaculo, ou seja, a direcao a seguir

    if (robotACT.getServoAngle() != 90) { // verifica se o servo nao esta centrado, e centra-o caso este nao esteja
        robotACT.lookFront(); // centra o servo
    }

    int level = robotREACT.infraRedLevel();
    System.out.println("\nanguloRobot = " + robotACT.getServoAngle() + " level = " + level);

    if (level < 3) { // le o valor do sensor no angulo 90, caso este nao tenha obstaculo devolve 90°,
        pos = 90; // ou seja, indica que a direcao a seguir é em frente
        System.out.println("Angulo do servo 90 - nao tem obstaculo");
    } else { // caso exista um obstaculo a frente do robot, vamos calcular o caminho a seguir
        System.out.println("Angulo do servo 90 - tem obstaculo\n\nVirar a esquerda");

        robotACT.lookLeft(incremento); // olha para a esquerda
        int levelEsquerda = robotREACT.infraRedLevel(); // le o valor do sensor a esquerda
        pos = robotACT.getServoAngle(); // le o angulo do robot
        System.out.println("\tanguloRobot(pos) = " + robotACT.getServoAngle() + " pos(nivel sensor) = " + pos + "\n\nVirar a direita");


        robotACT.lookRight(incremento * 2); // olha para a direta
        int levelDireita = robotREACT.infraRedLevel(); // le o valor do sensor a direita
        int pos2 = robotACT.getServoAngle(); // le o angulo do robot
        System.out.println("\tanguloRobot (pos2) = " + pos2 + " levelDireita = " + levelDireita);

        // se o valor do sensor a direita for menor que a esquerda (ou seja o obstaculo esta mais longe a direta)
        if (levelDireita < levelEsquerda) {
            System.out.println("\n\t\nlevelDireita < pos");
            pos = pos2; // atualiza o valor a devolver para o angulo a direita
        }
    }

    return pos; // devolve o angulo do servo sem obstaculo
}
```

Figura 5.3-63 Resolução do exercício D5 com a biblioteca (Java) – Part 2

EXERCÍCIO D6

O robot encontra-se num labirinto, percorre-o e procura a saída (através dos *bumpers*). Se intercetar um obstáculo buzina, recua, corrige e avança. Quando o robot chegar ao destino, acende-se uma luz para este saber que já concluiu o percurso.

Possível implementação do exercício na Figura 5.3-64.

```java
public void ExercicioD6() {

    System.out.println("robotREACT.lightLevel() = " + robotREACT.lightLevel()); // le o valor do sensor de luz (para propositos de debug)

    while (robotREACT.lightLevel() <= 4) { // continua enquanto nao for detetada uma luz que indique que chegou ao destino
        robotACT.goFront(5); // anda para a frente
        ExercicioD6_1(); // le os valores dos bumpers e faz as devidas correcoes
    }
}

// lê os bumpers, anda pa tras.. vira um pouco (45º) para o lado oposto de onde bateu..
public boolean ExercicioD6_1() {
    int[] varBumpers; // array para guardar os valores dos bumpers ( para poder imprimir para debug )
    varBumpers = robotREACT.readBumpersArray(); // le os valores dos bumpers
    System.out.println("bumpers[0] = " + varBumpers[0] + " bumpers[1] = " + varBumpers[1]); // imprime os valores para propositos de debug

    if (robotREACT.getBumperLeftValue()) { // caso o bumper esquerdo tenha um obstaculo
        robotACT.sound(400, 500); // apita
        System.out.println("goBack(10) - " + robotACT.goBack(5)); // anda para tras
        System.out.println("turnRight(5) - " + robotACT.turnRight(5)); // vira o robot a direita
        return true;
    } else if (robotREACT.getBumperRightValue()) { // caso o bumper direito tenha um obstaculo
        robotACT.sound(400, 500); // apita
        System.out.println("goBack(10) - " + robotACT.goBack(5)); // anda para tras
        System.out.println("turnLeft(5) - " + robotACT.turnLeft(5)); // vira o robot a esquerda
        return true;
    }
    return false;
}
```

Figura 5.3-64 Resolução do exercício D6 com a biblioteca (Java)

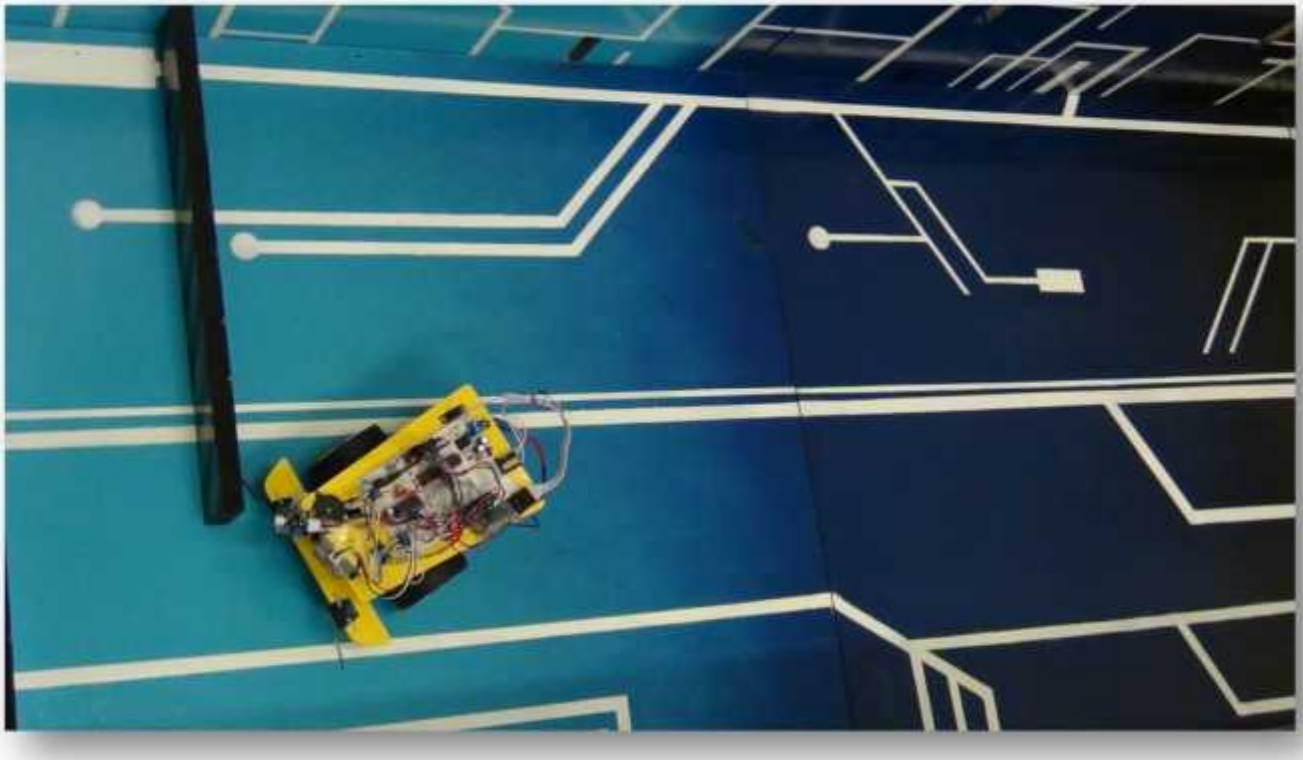


Figura 5.3-65 Robot percorre o labirinto, guiado pelos *bumpers*

EXERCÍCIO D7

O robot encontra-se num labirinto, percorre-o ida e volta (através do sensor IR e dos *bumpers*). O robot desvia-se dos obstáculos quando detetados no IR e corrige quando bate num dos *bumpers.*

Possível implementação do exercício na Figura 5.3-66.

```
public void ExercicioD7() {
    while (robotREACT.lightLevel() <= 4) { // continua enquanto nao for detetada uma luz que indique que chegou ao outro lado
        ExercicioD7_1(); // percorre o labirinto
    }
    robotACT.turnRight(180); // inverte a direcao rodando 180º a direita
    ExercicioD7_1(); // volta a percorrer o labirinto
}

// robot percorre um labirinto.. desvia-se dos obstaculos quando detetados no IR e corrige quando bate nos bumbers
public void ExercicioD7_1() {
    ExercicioD6_1(); // le os bumpers, corrige a direcao do robot caso tenha batido

    int anguloServo = ExercicioD5_1(); // le o sensor IR, devolve 90 caso nao haja obstaculos a frente,
    // 30 caso o caminho a seguir seja a direita
    // 150 caso o caminho a seguir seja a esquerda

    if (anguloServo < 90) { // direita - o caminho a seguir é a direita, ou seja o angulo do servo com obstaculo mais longe é a direita
        robotACT.turnRight(5); // vira a direita
    } else if (anguloServo > 90) { // esquerda - o caminho a seguir é a esquerda, ou seja o angulo do servo com obstaculo mais longe é a esquerda
        robotACT.turnLeft(5); // vira a esquerda
    }

    robotACT.goFront(5); // anda em frente
}
```

Figura 5.3-66 Resolução do exercício D7 com a biblioteca (Java)

Verifica-se que os testes realizados com os exercícios deste grupo foram bem-sucedidos. O grau de dificuldade de resolução destes exercícios aumentou consideravelmente neste grupo mas todos os exercícios foram resolvidos e testados com sucesso. As funções existentes na biblioteca demonstraram estar funcionais e capazes de ser utilizadas em algoritmos de maior complexidade.

5.3.5. *Testes com os exercícios do grupo E*

Os exercícios contidos neste grupo foram desenvolvidos para trabalho futuro do projeto. São exercícios de alta complexidade de resolução exigindo a construção de algoritmos complexos.

No entanto, 3 desses exercícios foram resolvidos, para testarmos o comportamento da biblioteca na implementação de algoritmos deste grau de complexidade.

EXERCÍCIO E1

O robot encontra-se num labirinto, percorre-o (através do sensor IR e dos *bumpers*) e procura uma luz (máximo absoluto). O robot desvia-se dos obstáculos quando detetados no IR e corrige quando bate num dos *bumpers.* Tem como objetivo encontrar a localização de uma luz no labirinto, quando a encontrar apita.

Possível implementação do exercício na Figura 5.3-67.

```
public void ExercicioE1() {
    while (robotREACT.lightLevel() <= 4) {
        boolean obstaculosBumpers = exD.ExercicioD6_1(); // le os bumpers, corrige a direcao do robot caso tenha batido

        int anguloServoIR = exD.ExercicioD5_1(); // le o sensor IR, devolve 90 caso nao haja obstaculos a frente,
        // 90 caso o caminho a seguir seja a direita
        // 150 caso o caminho a seguir seja a esquerda

        if (anguloServoIR < 90) { // direita - o caminho a seguir é a direita, ou seja o angulo do servo com obstaculo mais longe é a direita
            robotACT.turnRight(5); // vira a direita
        } else if (anguloServoIR > 90) { // esquerda - o caminho a seguir é a esquerda, ou seja o angulo do servo com obstaculo mais longe é a esquerda
            robotACT.turnLeft(5); // vira a esquerda
        }

        if (!obstaculosBumpers && anguloServoIR == 90) {
            int anguloServoLuz = exD.ExercicioD3_1();

            if (anguloServoLuz < 90) { // direita - o caminho a seguir é a direita, ou seja o angulo do servo com obstaculo mais longe é a direita
                robotACT.turnRight(5); // vira a direita
            } else if (anguloServoLuz > 90) { // esquerda - o caminho a seguir é a esquerda, ou seja o angulo do servo com obstaculo mais longe é a esquerda
                robotACT.turnLeft(5); // vira a esquerda
            }
        }

        robotACT.goFront(5); // anda em frente
    }
    exD.ExercicioD3(); // O robot procura a luz (torre) - (esquerda - frente - direita) quando encontra pisca LED e vira para o lado da luz maior
}
```

Figura 5.3-67 Resolução do exercício E1 com a biblioteca (Java)

Figura 5.3-68 Robot percorre o labirinto e procura a origem da luz

5.4. Utilização da solução em algoritmos mais complexos

A biblioteca desenvolvida tem potencialidade para ser integrada em projetos com algoritmos mais complexos, como é o caso de algoritmos de inteligência artificial e de redes neuronais. Os exercícios contidos no grupo E são um exemplo disso e foram exercícios desenvolvidos para trabalho futuro do projeto. São exercícios de alta complexidade de resolução exigindo a construção de algoritmos complexos.

A biblioteca pode ser utilizada para testes de algoritmos de projetos de inteligência artificial e redes neuronais, podendo assim ser possível obter-se uma representação real com um robot sem a necessidade dos programadores se preocuparem com a programação do robot e aspetos adjacentes ao desenvolvimento direto dos algoritmos específicos das áreas em que trabalham e investigam.

No entanto, note-se que não é objetivo do projeto o desenvolvimento deste tipo de algoritmos.

5.5. Testes de Qualidade do Sistema

Os testes de sistema têm como base o documento de especificação de requisitos, para além de avaliar os atributos de qualidade referidos na seção de requisitos. Normalmente estes testes são da responsabilidade de uma equipa independente, neste caso foram realizados pelo programador da aplicação.

5.5.1. *Hardware e Software Utilizados*

Para a realização dos seguintes testes foi utilizado um computador com as seguintes características:

Tabela 5.5.1-1 Hardware e Software do computador utilizado nos testes de qualidade

Portátil	
Componente	**Requisito**
Processador	Intel(R) Core™ i7-263QM CPU @2.00GHz
Memória RAM	6.00GB
Disco Rígido	500GB 7200rpm
Vídeo	ATI Radeon ™ HD 6770M
Sistema Operativo	Windows 7 Home Premium (Service Pack 1)
IDE	Netbeans

5.5.2. *Testes de Desempenho*

Esta categoria engloba três pontos essenciais: **capacidade de processamento**, **capacidade de armazenamento** e **capacidade de resposta**.

CAPACIDADE DE PROCESSAMENTO

Tabela 5.5.2-1 Teste de desempenho – Capacidade de processamento

ID	TS1
Titulo	Capacidade de Processamento de Informação
Introdução	O objetivo é testar a capacidade de processamento de código, utilizando para tal compilação de código que possua um conjunto de funções da biblioteca desenvolvida.
Itens a serem testados	Capacidade de processamento de dados, que utilize uma determinada percentagem do processador.
Características a serem testadas	Neste teste, iremos testar a capacidade de processamento de código, através da inserção de registos, que utilize somente 50% do processador.
Aproximação	Compilação de blocos de código que utilize funções da biblioteca desenvolvida e posterior observação da percentagem de utilização da CPU, o que nos dará

	informação sobre a capacidade de processamento de informação por parte do Sistema.
Responsabilidades	Testar a capacidade de processamento de dados e percentagem de utilização da CPU correspondente.
Resultado	Os resultados estão de acordo com o esperado, sendo que a utilização do CPU vai variando ao longo da operação mas na maior parte do tempo não ultrapassa os 50%.

CAPACIDADE DE ARMAZENAMENTO

Tabela 5.5.2-2 Teste de desempenho – Capacidade de armazenamento

ID	TS2
Titulo	Capacidade de Armazenamento de Informação
Introdução	O objetivo é testar a capacidade de armazenamento de dados, utilizando para tal a inserção de dados na EEPROM do robot.
Itens a serem testados	Capacidade de armazenamento de dados, previamente inseridos no Robot.
Características a serem testadas	Neste teste, iremos testar a capacidade de armazenar dados, através da inserção posterior de variáveis na EEPROM do robot.
Aproximação	Inserção de todas as variáveis default das funções, e posterior observação da capacidade da EEPROM de armazenar os dados.
Responsabilidades	Testar a capacidade de armazenamento de dados tendo em conta a inserção das variáveis default das funções do robot.
Resultado	Os resultados estão de acordo com o esperado.

CAPACIDADE DE RESPOSTA

Tabela 5.5.2-3 Teste de desempenho – Capacidade de resposta

ID	TS3
Titulo	Capacidade de Resposta
Introdução	O objetivo é testar a capacidade de resposta do sistema.
Itens a serem testados	Capacidade de resposta do robot, de forma a ser possível aos utilizadores executar uma determinada ação com o robot dentro de um tempo limite aceitável.
Características a serem testadas	Neste teste, iremos testar a capacidade de resposta do Robot que permita ao utilizador em questão, executar a ação desejada num determinado tempo.
Aproximação	Capacidade de resposta do sistema, permitindo ao utilizador iniciar a ação pretendida em 1 segundo, ou no pior caso, ao fim de 10 segundos.

Responsabilidades	Testar a capacidade de resposta do sistema, quando o utilizador pretende executar uma determinada ação do robot.
Resultado	Os resultados estão de acordo com o esperado, pois na simulação de uma utilização comum do robot este respondeu sempre dentro do tempo esperado.

5.5.3. *Testes de Disponibilidade*

Esta categoria permite determinar a utilidade do Sistema no desempenho de tarefas para as quais foi concebido, e é constituído por três pontos fundamentais: **fiabilidade**, **manutibilidade** e **integridade**.

FIABILIDADE

Tabela 5.5.3-1 Teste de Disponibilidade – Fiabilidade

ID	TS4
Titulo	Fiabilidade
Introdução	O objetivo é testar a fiabilidade do sistema.
Itens a serem testados	Capacidade da biblioteca funcionar sem falhas, tendo em conta um determinado tempo de utilização por parte do utilizador.
Características a serem testadas	Neste teste, iremos testar a fiabilidade da biblioteca, ou seja, iremos garantir que o utilizador usufrui do serviço, com qualidade e a qualquer hora.
Aproximação	Utilização e monitorização da biblioteca e das suas funções durante uma semana para calcular o seu uptime com base nisso. O ideal é um *uptime* de 99,9% mas algo na ordem dos 99% já é aceitável.
Responsabilidades	Testar a fiabilidade do Sistema, através da compilação de blocos de código com funções do robot e posteriores consultas e monitorização durante um determinado período de tempo.
Resultado	Teste não efetuado.

MANUTIBILIDADE

Tabela 5.5.3-2 Teste de Disponibilidade – Manutibilidade

ID	TS5
Titulo	Manutibilidade
Introdução	O objetivo é testar a manutibilidade, para medir o tempo médio de reparações entre falhas.
Itens a serem testados	Capacidade de reparação de falhas, medindo o tempo que demora essas mesmas reparações.
Características a serem testadas	Neste teste, iremos testar a capacidade de reparar possíveis falhas que ocorram no Sistema.
Aproximação	Inserção de erros na biblioteca, para posterior verificação do tempo médio de reparação entre as falhas que ocorreram. O tempo médio estipulado como ótimo, é de 60 minutos, sendo que o tempo máximo é de 8 horas.
Responsabilidades	Testar a capacidade de reparação de falhas que possam ocorrer no Sistema.
Resultado	Os resultados estão de acordo com o esperado é possível recuperar o código na integra de todos os componentes da biblioteca.

INTEGRIDADE

Tabela 5.5.3-3 Teste de Disponibilidade – Integridade

ID	TS6
Titulo	Integridade
Introdução	O objetivo é testar a integridade, para medir a capacidade do sistema manter as suas funcionalidades disponíveis de forma integra no decorrer das operações do mesmo
Itens a serem testados	Capacidade do sistema manter as suas funcionalidades disponíveis de forma integra no decorrer das operações do mesmo, medindo a percentagem de funções afetadas por erros de integridade.
Características a serem testadas	Neste teste, iremos testar a integridade das funções da biblioteca aquando da sua utilização.
Aproximação	Monitorização das funções da biblioteca durante uma semana de utilização e posterior análise da integridade das funções. A percentagem de funções afetadas por problemas de integridade deverá ser nula, mas em todo o caso nunca deverá ultrapassar os 1%
Responsabilidades	Testar o nível de integridade das funções da biblioteca.
Resultado	Teste não efetuado

5.5.4. *Testes de Adaptabilidade*

Esta categoria consiste numa medida que permite definir a capacidade que o Sistema tem para se adaptar a novas situações, sem perder a sua eficiência. Consiste em **Extensibilidade**, **portabilidade** e **acessibilidade**.

EXTENSIBILIDADE

Tabela 5.5.4-1 Teste de Adaptabilidade – Extensibilidade

ID	TS7
Titulo	Extensibilidade
Introdução	O objetivo é testar a extensibilidade do sistema.
Itens a serem testados	Capacidade de extensibilidade do sistema, caso sejam necessárias posteriores atualizações.
Características a serem testadas	Verificar a resposta do sistema quando se alteram funcionalidades e novos dados são inseridos.
Aproximação	Alterações de determinadas funcionalidades, para posterior observação do impacto no desempenho (correr de novo os testes de desempenho) do sistema. Espera-se que a performance do sistema não seja degrada em mais de 50%.
Responsabilidades	Testar a capacidade de extensibilidade do Sistema, através da introdução de alterações.
Resultado	Os resultados estão de acordo com o esperado. O sistema foi desenvolvido de forma modular e ao longo do desenvolvimento não se assistiu a uma redução significativa do desempenho com a adição de novos módulos.

PORTABILIDADE

Tabela 5.5.4-2 Teste de Adaptabilidade – Portabilidade

ID	TS8
Titulo	Portabilidade
Introdução	O objetivo é testar a portabilidade do sistema.
Itens a serem testados	Capacidade de portabilidade do sistema, caso seja necessário corrê-lo em outras plataformas de software.
Características a serem testadas	Verificar se o sistema executa corretamente em várias plataformas.
Aproximação	Configurar a aplicação em ambiente Windows, Linux e Mac OS X. Verificar se funciona tudo sem problemas. Espera-se que funcione em todos mas que pelo menos em Windows não haja qualquer tipo de problemas visto que foi a plataforma onde o sistema foi desenvolvido.
Responsabilidades	Testar a capacidade de portabilidade para outras plataformas de software, tentando configurá-lo e operá-lo noutros sistemas operativos.

Resultado	O sistema foi testado com sucesso funcionando em todos os sistemas operativos listados. Ainda não foi testado nas outras plataformas.

ACESSIBILIDADE

Tabela 5.5.4-3 Teste de Adaptabilidade – Acessibilidade

ID	TS9
Titulo	Acessibilidade
Introdução	O objetivo é testar a acessibilidade do sistema.
Itens a serem testados	Capacidade de acessibilidade do sistema a partir de vários IDE's
Características a serem testadas	Verificar se o sistema é apresentado corretamente em vários IDE's.
Aproximação	Testar a biblioteca com cada um dos IDE (Netbeans, Eclipse, Gel, JEdit e OptimalJ) e correr um conjunto de código para determinar se cada um deles comunica com o robot corretamente. Espera-se que todos funcionem corretamente mas em último caso espera-se que funcione perfeitamente em Eclipce e Netbeans.
Responsabilidades	Testar a capacidade de acessibilidade do sistema a partir de vários IDE's.
Resultado	Os resultados estão de acordo com o esperado, sendo que foi testado em todos os IDE's listados com sucesso.

5.5.5. *Testes de Usabilidade*

Diz respeito à extensão na qual um produto pode ser usado por utilizadores específicos para alcançar objetivos específicos com efetividade, eficiência e satisfação num contexto de uso específico. Consiste em **facilidade de aprendizagem**, **satisfação**, **eficiência na utilização** e **resistência a erros**.

FACILIDADE DE APRENDIZAGEM

Tabela 5.5.5-1 Teste de Usabilidade – Facilidade de Aprendizagem

ID	TS10
Titulo	Facilidade de Aprendizagem
Introdução	O objetivo é testar a capacidade que o utilizador tem para aprender a trabalhar com a biblioteca.
Itens a serem testados	Capacidade de aprendizagem do utilizador num determinado período de tempo.
Características a	Neste teste, iremos testar a capacidade de aprendizagem do utilizador,

serem testadas	baseando-nos no tempo que este demora a assimilar a correta utilização de um conjunto de funções da biblioteca.
Aproximação	Capacidade que o utilizador tem para adquirir os conhecimentos necessários para utilizar a biblioteca de forma simples. O tempo estipulado será de 30 minutos, sendo o pior caso o utilizador demorar 90 minutos.
Responsabilidades	Testar a capacidade de aprendizagem do utilizador, para utilizar a biblioteca.
Resultado	Teste não efetuado.

EFICIÊNCIA NA UTILIZAÇÃO

Tabela 5.5.5-2 Teste de Usabilidade – Eficiência na Utilização

ID	TS11
Titulo	Eficiência na Utilização
Introdução	O objetivo é testar a eficiência do Sistema.
Itens a serem testados	Capacidade do utilizador para realizar um conjunto de tarefas, de forma eficaz.
Características a serem testadas	Neste teste, iremos testar a eficiência do Sistema, recorrendo para tal à realização de ações por parte do utilizador.
Aproximação	Permitir ao utilizador uma boa eficiência na utilização da aplicação, isto é, medir a eficiência do utilizador na realização de um conjunto de tarefas. O objetivo é que o utilizador execute o conjunto de ações em 20 tentativas ou menos. Na pior das hipóteses espera-se que necessite de 50 tentativas.
Responsabilidades	Testar a capacidade do utilizador realizar determinadas tarefas de forma eficientes, isto é, com o menor esforço possível por parte do utilizador.
Resultado	Teste não efetuado.

RESISTÊNCIA A ERROS

Tabela 5.5.5-3 Teste de Usabilidade – Resistência a Erros

ID	TS12
Titulo	Resistência a Erros
Introdução	O objetivo é testar a quantidade de erros no sistema.
Itens a serem testados	Vamos testar a quantidade de erros que surgem quando um utilizador utiliza o sistema.
Características a serem testadas	Neste teste iremos testar a quantidade de erros que podem surgir durante a utilização do sistema por parte de um utilizador.
Aproximação	Para a realização deste teste iremos testar a biblioteca á procura de erros possíveis agindo como um utilizador que não tem conhecimento da estrutura

	da biblioteca.
Responsabilidades	Testar a aplicação como um todo á procura para prevenir erros que possam surgir e não foram contemplados na fase de análise da aplicação.
Resultado	Teste não efetuado.

SATISFAÇÃO

Tabela 5.5.5-4 Teste de Usabilidade – Satisfação

ID	**TS13**
Titulo	Satisfação
Introdução	O objetivo é testar o grau de satisfação do utilizador.
Itens a serem testados	Capacidade de satisfazer as expectativas do cliente.
Características a serem testadas	Neste teste, iremos testar o grau de satisfação do cliente, grau este, que será medido tendo em conta a utilização do sistema durante um determinado período de tempo.
Aproximação	Capacidade que o sistema tem para satisfazer o cliente, ou seja, o sistema deverá conter características que satisfaçam os desejos, necessidades e expectativas do cliente. Para tal o cliente utilizará o sistema durante um determinado tempo e posteriormente será alvo de um questionário que irá determinar o seu grau de satisfação de 1 a 5 (sendo 1 para um sistema que não satisfaz as suas necessidades e 5 caso o sistema satisfaça completamente o cliente).
Responsabilidades	Testar a capacidade de satisfazer o cliente, através de questionários feitos aos utilizadores, de modo a obtermos informação necessária para determinar o grau de satisfação dos clientes.
Resultado	Teste não efetuado.

5.6. Conclusões

De acordo com o segundo objetivo que foi definido e em resposta a primeira questão definida no segundo objetivo, “em que medida é possível criar mecanismos de abstração para uma plataforma robótica”, podemos concluir que é possivel criar esses mecanismos através da biblioteca de abstração que foi desenvolvida para o efeito.

Por último verificamos que os mecanismos de abstração implementados na biblioteca permitem uma maior facilidade de interação com a plataforma. Baseamos a nossa afirmação em resultado dos testes realizados na plataforma em contexto real.

Capítulo 6
Conclusões e Trabalho Futuro

Neste capítulo podemos identificar as diversas conclusões a que chegamos com a conclusão do trabalho proposto.

Está também presente neste capítulo as propostas de trabalho futuro, que poderá ser realizado e que poderá definir uma evolução positiva de todo o projeto e que poderá satisfazer algumas possíveis e previsíveis necessidades.

6.1. Conclusões

Ao longo deste trabalho identificamos as plataformas micro robóticas, as suas características e contextos de utilização, assim como as linguagens utilizadas na comunicação. Como robot identificamos o robot Lego Mindstorm NXT, o robot Finch, o robot farruco e o robot Hemisson. Foram identificadas varias linguagens de programação utilizadas na comunicação com estas plataformas robóticas e o contexto de utilização destas plataformas mais frequente é o contexto educacional.

Em análise ao trabalho relacionado identificamos alguns projetos desenvolvidos nesta área, como o projeto tekkotsu, URBI e a biblioteca dLife. Da análise efetuada a estes projetos concluímos que existe uma crescente necessidade de criar mecanismos de abstrair os conceitos de *hardware* e de programação de baixo nível dos utilizadores de sistemas robotizados.

Em função da literatura revista e da comparação efetuada aos diferentes robots, no âmbito deste trabalho foi selecionado o robot farrusco, pois este utiliza uma placa compatível com Arduino (open-source). Desta forma a biblioteca desenvolvida não se encontra limitada a uma plataforma proprietária podendo ser utilizada em qualquer robot compatível com a plataforma arduíno.

O trabalho com robots envolve o conhecimento de áreas relacionadas com eletrónica, robótica, e programação. Para uma pessoa que não tenha nenhuns conhecimentos de eletrónica e não tenha noção de como se controlam os componentes de um robot, desde o seu funcionamento aos relacionamentos existentes entre eles, torna-se uma tarefa ingrata pois a pessoa não vai conseguir alcançar os resultados desejados. Sem um estudo aprofundado dessas áreas será quase impossível colocar um robot em funcionamento.

A construção da biblioteca apresentada vem resolver parte destes problemas. Uma pessoa que apenas tenha conhecimentos de programação java, ou que esteja ainda a aprender os conceitos básicos, poderá controlar um robot, fazendo-o interagir com o meio ambiente e assim desfrutar da satisfação de programar o comportamento de um robot sem que para isso seja necessário quaisquer conhecimentos de eletrónica e robótica, tendo assim uma total abstração desses conceitos.

No desenvolvimento desta biblioteca um dos primeiros problemas que se teve de ultrapassar foi o desconhecimento total do funcionamento de um robot, o mesmo problema com o qual era proposto solucionar. Foi então necessário um estudo intensivo do comportamento do robot para que fosse possível estruturar uma biblioteca cujo propósito fosse a abstração do utilizador de todos esses conceitos associados ao controlo de um robot.

Tendo então esse *know how* adquirido tornou-se então possível desenharmos a estrutura da biblioteca. Na construção da biblioteca, o próximo desafio que surgiu foi o estabelecimento da comunicação entre o computador do utilizador e o robot, foi necessário desenvolver algoritmos que tratassem de toda a comunicação entre ambos de forma o mais abstrata quanto possível para o utilizador. Após a comunicação estar estabelecida entre ambos foi então necessário manter a execução do código do utilizador em perfeita sintonia com as ações do robot. Criaram-se então funções bloqueantes que assegurassem o sincronismo entre código na IDE e as ações do robot.

Para o desenvolvimento de algumas funcionalidades do robot, estas têm de ser executadas a cada iteração cada uma delas influenciando o comportamento da próxima, assim durante o desenvolvimento das funções do robot, foi necessário desenvolver funções que traduzissem um comportamento específico a certos atuadores do robot, para assim possibilitar ao utilizador final tirar maior partido das possibilidades do equipamento.

Foram desenvolvidas duas aplicações para facilitar a interação com o robot. Uma aplicação de testes de funcionalidades do robot que permite de forma direta testar todas as funcionalidades do robot sem ter de se criar um projeto específico para esse efeito. Foi também desenvolvida uma aplicação de configuração do robot para calibrar todas as variáveis deste.

Finalmente para o teste funcional da biblioteca, foi criada uma segunda biblioteca que utiliza as funções disponibilizadas na biblioteca principal, adaptando-as ao seu objetivo específico. Esta segunda biblioteca foi desenvolvida direcionada para o ensino de conceitos de programação em java, seguindo um conjunto de exercícios pré-estabelecidos e fundamentados para ensinar os diversos temas de forma muito mais motivadora e agradável.

Concluímos então que a biblioteca resultante funciona e permite testar uma plataforma robótica móvel, sem a necessidade de se introduzir complexidade ao código do programa. Esta biblioteca torna então possível a um utilizador sem conhecimentos base de eletrónica e robótica programar o

comportamento de um robot que interaja com o meio ambiente, isolando completamente os detalhes associados ao hardware do robot. Através da utilização da biblioteca desenvolvida é possível realizar os exercícios de formas mais simples.

6.2. Trabalho Futuro

Como trabalho futuro da biblioteca desenvolvida, seria interessante adicionar mais sensores e atuadores ao robot, ou ate mesmo migrar o código para um robot mais robusto. Temos também a possibilidade de desenvolvermos a biblioteca em mais linguagens de programação, nomeadamente a linguagem C, que por ser uma linguagem procedimental iria ser necessário alterar a abordagem em algumas das funcionalidades, e também na linguagem C#.

Em termos de aplicabilidade da biblioteca, poderia ser utilizada para testes de algoritmos de projetos de inteligência artificial e redes neuronais, podendo assim ser possível obter-se uma representação real com um robot em substituição aos utilizados modelos virtuais gráficos.

Os exercícios contidos no grupo E são um exemplo de aplicabilidade com algoritmos de inteligência artificial e foram exercícios desenvolvidos para trabalho futuro do projeto.

Capítulo 7
Bibliografia

- Baillie, Jean-Christophe. “URBI: Towards a Universal Robotic Low-Level Programming Language”. In procedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS’05) páginas 820-825, 2005.

- Baillie, Jean-Christophe, e colegas, “The Urbi Universal Platform for Robotics”. Workshop Proceedings of SIMPAR 2008. Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS. Venice(Italy) 2008 November,3-4. ISBN 978-88-95872-01-8, paginas 580-591

- BRAGA, N. C. Robotics, Mechatronics, and Artificial Intelligence: Experimental Circuit Bloks for Designers. Worburn-MA-USA: Newnes, 2002. ISBN 0-7506-7389-3.

- Braught, Grant, (2012). “dLife: A Java Library for Multiplatform Robotics, AI and Vision in Undergraduate CS and Research”. Department of Mathematics and Computer Science, ACM 978-1-4503-1098-7/12/02, 2012.

- Brandão F., Vaz P., (1999). “Cyber-Kasparov – Um robô que joga xadrez”. [Online] Disponivel a partir de: http://robotics.dem.uc.pt/norberto/seminarios/sem990002_dem_gcg_jnp.pdf [Acedido a 22 Outubro de 2013]

- Carrara, Vardemir. “Apostila de Robótica”, Universidade Braz Cubas, Área de Ciências Exatas, Engenharia Mecânica e Engenharia de Controlo e Automação. Disponível a partir de: http://www.fbpsistemas.com.br/Arquivos/Apostila%20de%20Robotica.pdf [Acedido a 22 Outubro de 2013]

- Castilho, M. “Robotica na Educação: Com que Objetivos?”, [Online] Pontifícia univercidade Católica do Rio Grande do Sul. Disponivel a partir de: http://www.pucrs.br/eventos/desafio/mariaines.php [Acedido a 1 Outubro de 2013]

- Chaimowicz, L., Campos, M., “Nós e os Robôs”, 1999. [Online] Disponivel a partir de: http://www.brasilescola.com/informatica/robos.htm [Acedido a 3 Outubro de 2013]

- Chen, Z., Marx, D. “Experiences with eclipse ide in programming courses”, Journal of Computing Sciences in Colleges, Volume 21 Issue 2, Dezembro de 2005, páginas 104-112.

- CHUCK MCMANIS. Introduction to Servo Motors. [S.l.], 2006. Web. Acessado em Outubro, 2006. Disponível em: <http://www.uoguelph.ca/ antoon/hobby/servo1.htm>.

- Dowling K., (1996). “When did robots, as we know them today, come into existence?”. [Online] Disponivel em: http://www.cs.cmu.edu/~chuck/robotpg/robofaq/1.html [Acedido em 12 de Maio de 2013]

- Garcia, M. A and Patterson-McNeill, H. (2002). “Learn how to develop software using the toy Lego Mindstorms.” 32nd ASEE/IEEE Frontiers in Education Conference, November 6-9, 2002, Boston, MA.

- Gustavsson, Jenny, 2012, “A school student laboratory using robotics Based on Lego Mindstorms”, disponivel a partir de <http://kth.diva-portal.org/smash/record.jsf?pid=diva2:516076>, acedido em 16-03-2013

- Jiang, WangLi, et al. “Robot Program Validation Using Virtualization, Components, e Physics Engines”. Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK), ISBN 978-3-8007-3273-9, Páginas 1083 - 1087

- Lauwers, T., Nourbakhsh, I., 2010. “Designing the Finch: Creating a Robot Aligned to Computer Science Concepts”. 24th AAAI Conference on Artificial Intelligence (AAAI-10).

- MORAES, Airton Almeida de Moraes. “Robótica.” Departamento de Meios Educacionais e Gerência de Educação, Diretoria Técnica do SENAI-SP. Serviço Nacional de Aprendizagem Industrial, 2003. Apostila. Disponível a partir de <http://www.adororobotica.com/RBSENAI.pdf>. Acedido em 17 de Outubro de 2013.

- MOTT, R. L. Machine elements in mechanical design. Upper Saddle River-NJ-USA: Prentice-Hall, 1999.

- Ribeiro, C.R, (2006). RobôCarochinha: Um Estudo Qualitativo Sobre a Robótica Educativa no 1º Ciclo do Ensino Básico. Mestrado em Educação Tecnologia Educativa. Universidade do Minho Instituto de Educação e Psicologia. Braga. Através de http://repositorium.sdum.uminho.pt/bitstream/1822/6352/2/teseRoboticaCeliaribeiroFinal.pdf [Acedido a 22 Outubro de 2013]

- Santos, N., e colegas, “Seja bem-vindo ao mundo da robótica!”. [Online] Departamento de

Informática (DIN) da Universidade Estadual de Maringá (UEM). Disponive a partir de: http://www.din.uem.br/ia/robotica/index.htm. [Acedido a 22 Outubro de 2013]

- Pires, J. Norberto. "Robótica – Das maquinas gregas á moderna robótica industrial". Universidade de Coimbra, Departamento de engenharia mecânica., Disponível em: http://robotics.dem.uc.pt/norberto/nova/pdfs/gregosxxi.pdf

- Ritchie, D. "The Development of the C Language". Bell Labs / Lucent Technologies. Second History of Programming Languages conference, Cambridge, Mass., April, 1993. Disponível a partir de <http://cm.bell-labs.com/who/dmr/chist.html> Acedido em 18 de Outubro de 2013.

- "RoboScope Hands On Robotics", 2011-04-28. Disponível a partir de http://www.robo-scope.de/ [Acedido em 2 de Outubro de 2013]

- Rosheim; Mark Elling. Leonardo's Lost Robots. Springer, 2006.

- Souza, A., Paixão, A., Uzêda, D., Dias, M., Duarte, S. e Amorim, H., "A placa Arduíno: uma opção de baixo custo para experiências de física assistida pelo PC". Instituto de Física, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, 21 de Março de 2011.

- Souza, F., "Robótica". [Online] J.A.M. Filippe de Souza. Disponivel a partir de: http://webx.ubi.pt/~felippe/main_pgs/mat_didp.htm [Acedido em 15 de Outubro de 2013]

- Ullrich, Roberto A. Robótica – Uma Introdução – O porquê dos robots e seu papel no trabalho, Ed. Campus, Rio de Janeiro, 1987.

Anexos

Instruções de instalação da biblioteca rxtx

A biblioteca RXTXComm.jar pode ser adquirida no seguinte endereço: http://www.jcontrol.org/download/rxtx_en.html.

Windows

- RXTXcomm.jar goes in \jre\lib\ext (under java)
- rxtxSerial.dll goes in \jre\bin

Mac OS X (x86 and ppc) (there is an Installer with the source)

- RXTXcomm.jar goes in /Library/Java/Extensions
- librxtxSerial.jnilib goes in /Library/Java/Extensions
- Run fixperm.sh thats in the directory. Fix perms is in the Mac_OS_X subdirectory.

Linux (only x86, x86_64, ia64 here but more in the ToyBox)

- RXTXcomm.jar goes in /jre/lib/ext (under java)
- librxtxSerial.so goes in /jre/lib/[machine type] (i386 for instance)
- Make sure the user is in group lock or uucp so lockfiles work.

Solaris (sparc only so far)

- RXTXcomm.jar goes in /jre/lib/ext (under java)
- librxtxSerial.so goes in /jre/lib/[machine type]
- Make sure the user is in group uucp so lockfiles work.

A person is added to group lock or uucp by editing /etc/groups. Distributions have various tools but this works:

- lock:x:54: becomes:
- lock:x:53: jarvi,taj

Now jarvi and taj are in group lock. Also make sure jarvi and taj have read and write permissions on the port.

Instrucoes de emparelhamento dos dispositivos Bluethooth

Passemos então a apresentar os passos para este processo, abra as definições da placa Bluetooth do computador, e clique em “Nova Ligação”.

Figura 6.2-1 Emparelhamento Bluetooth – Passo 1

Escolha o método de ligação que pretende, neste caso escolhemos “Modo Expresso” mas podemos também escolher o “Modo Personalizado”.

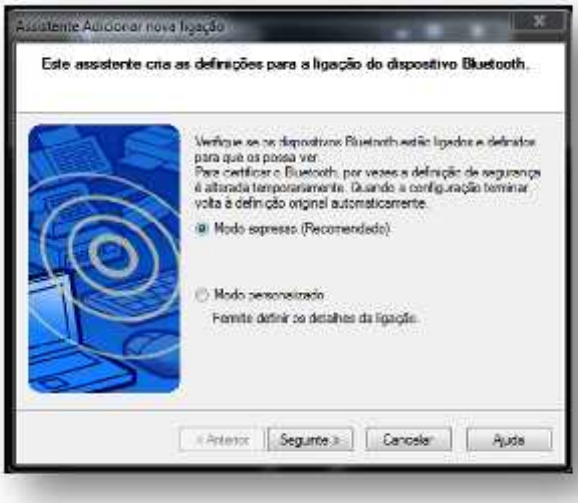


Figura 6.2-2 Emparelhamento Bluetooth – Passo 2

O assistente ira procurar por dispositivos Bluetooth disponíveis.

Figura 6.2-3 Emparelhamento Bluetooth – Passo 3

Escolha o Dispositivo do robot desejado, (por default os dispositivos Bluetooth utilizados no projeto vêm com o nome “linvor”, mas podem ser alterados posteriormente), e clique em seguinte.

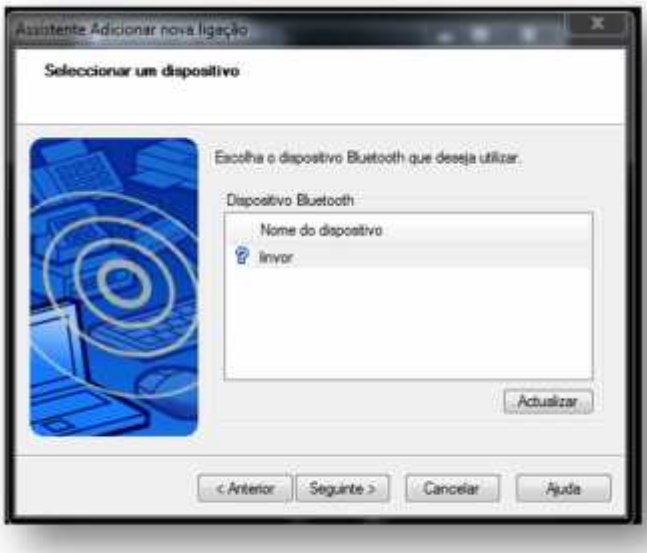


Figura 6.2-4 Emparelhamento Bluetooth – Passo 4

Aguarde enquanto o assistente adiciona o novo dispositivo.

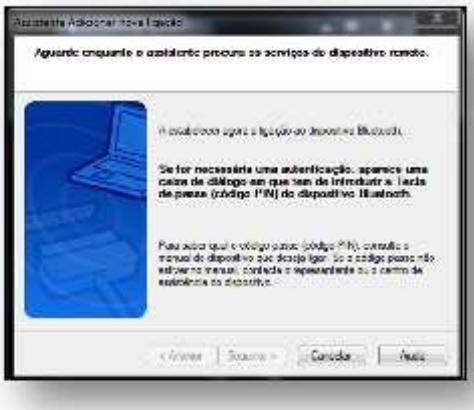


Figura 6.2-5 Emparelhamento Bluetooth – Passo 5

O assistente irá atribuir uma porta COM automaticamente a ligação. Clique Seguinte.

Figura 6.2-6 Emparelhamento Bluetooth – Passo 6

Agora já temos a ligação concluída entre os dois dispositivos. Por fim para podermos comunicar entre os dois teremos de ligar a conexão.

Figura 6.2-7 Emparelhamento Bluetooth – Passo 7

Aguarde enquanto o assistente liga a conexão.

Figura 6.2-8 Emparelhamento Bluetooth – Passo 8

Irá ser-lhe pedida uma *password*, para os dispositivos utilizados no projeto, a password *default* é 1234.

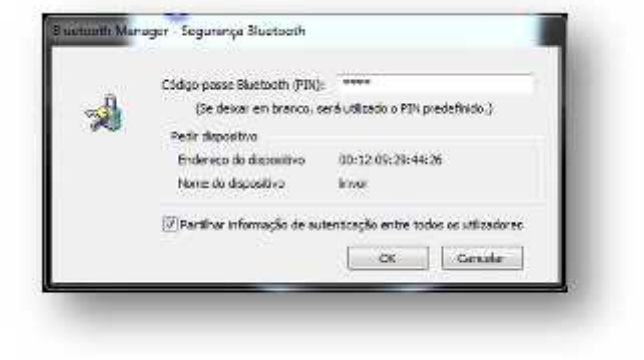


Figura 6.2-9 Emparelhamento Bluetooth – Passo 9

A ligação está ativa e pronta a iniciar a comunicação.

Figura 6.2-10 Emparelhamento Bluetooth – Passo 10

Criação de um novo projeto para utilização da biblioteca no Netbeans IDE

Primeiro inicie o NetBeans IDE. No Netbeans IDE, escolha “File” → “New Project” (Ctrl-Shift-N), como mostrado na figura abaixo.

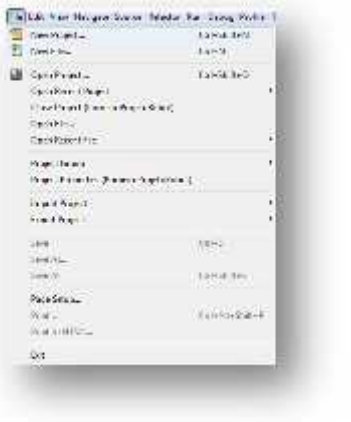


Figura 6.2-11 Ciar projeto em Netbeans utilizando a biblioteca desenvolvia – passo 2

No assistente “Choose Project”, expanda a categoria Java e selecione “Java Application”, como mostrado na figura abaixo. Em seguida, clique em” Next”.

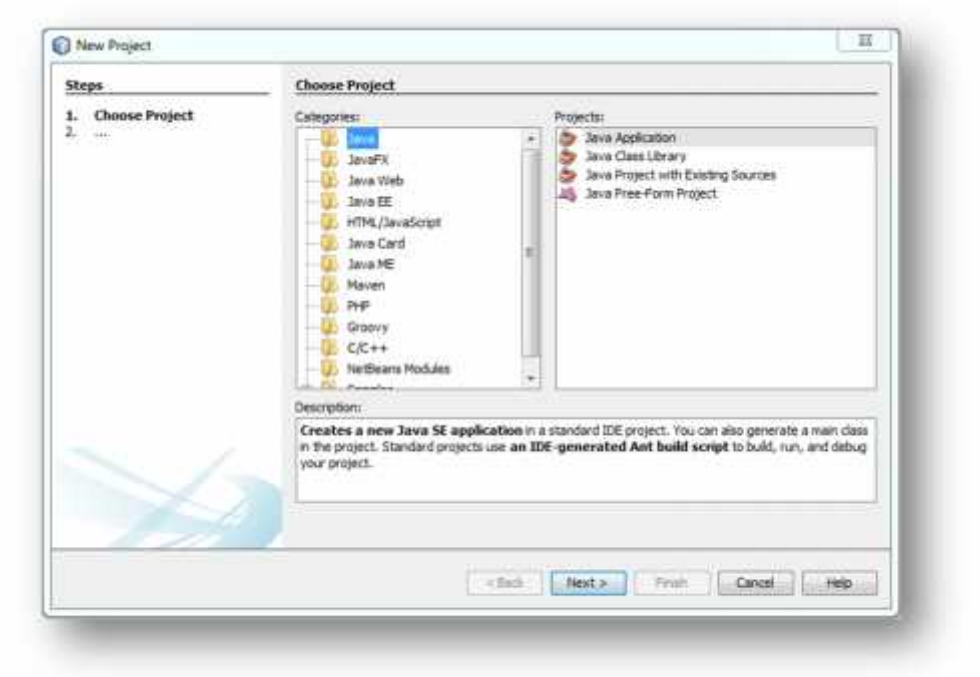


Figura 6.2-12 Ciar um projeto em Netbeans utilizando a biblioteca desenvolvida – passo 3

Na página “Name and Location” do assistente, siga o procedimento seguinte (como mostrado na figura abaixo):

a) No campo “Project Name”, digite “PrimeiroProjetoRobot”.

b) Deixe desmarcada a caixa de seleção “Use Dedicated Folder for Storing Libraries”.

c) No campo “Create Main Class”, digite “primeiroprojetorobot.PrimeiroProjetoRobot”.

d) Por fim, clique em “Finish”.

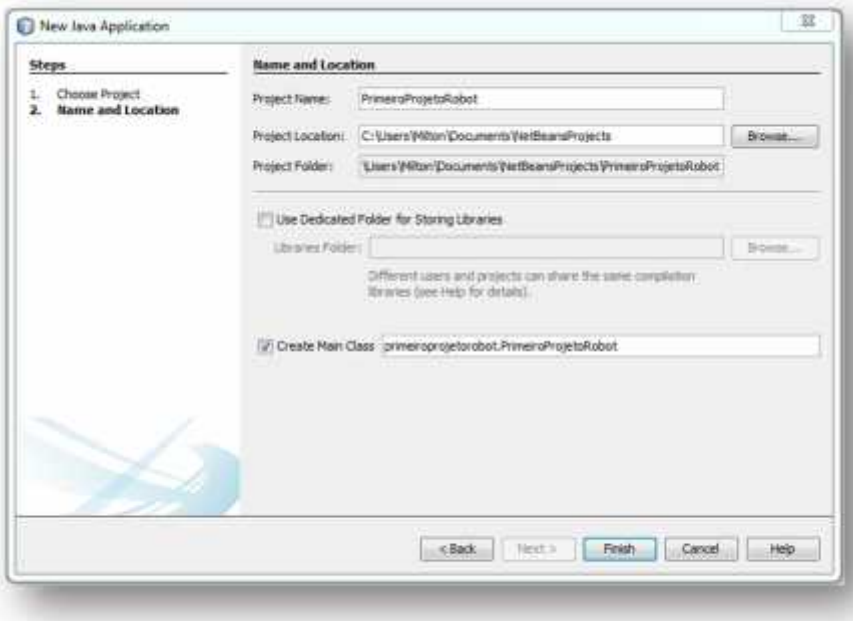


Figura 6.2-13 Ciar um projeto em Netbeans utilizando a biblioteca desenvolvia – passo 4

O projeto é criado e aberto no IDE. Agora temos disponíveis duas janelas. A janela Projetcs, que contém uma view em árvore dos componentes do projeto, incluindo arquivos de código-fonte, bibliotecas de que seu código depende, e assim por diante. A janela Editor de Código-fonte com um arquivo chamado PrimeiroProjetoRobot.java é aberta.

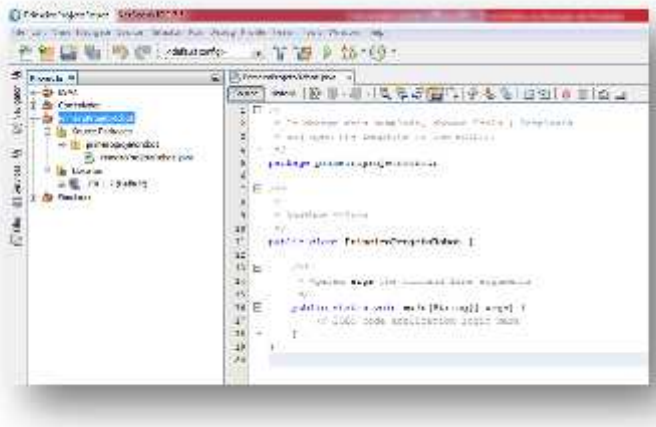


Figura 6.2-14 Ciar um projeto em Netbeans utilizando a biblioteca desenvolvia – passo

Vamos agora adicionar a biblioteca necessária ao projeto para controlarmos o robot. Clique em “Libraries” > “Add JAR/Folder…”.

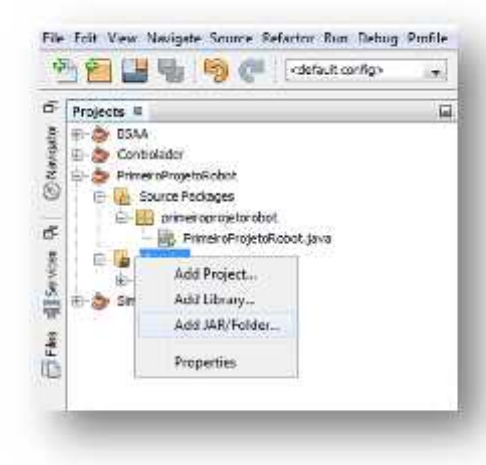


Figura 6.2-15 Ciar um projeto em Netbeans utilizando a biblioteca desenvolvia – passo 6

Selecione a biblioteca “BSAA.jar” e clique em “Open”.

Figura 6.2-16 Ciar um projeto em Netbeans utilizando a biblioteca desenvolvia – passo 7.1

Agora temos a biblioteca adicionada ao nosso projeto:

Figura 6.2-17 Ciar um projeto em Netbeans utilizando a biblioteca desenvolvia – passo 7.2

Para um exemplo simples de mover o robot 10 vezes para a frente utilizando um ciclo de repetição for:

a) Faça o import da classe “Robot” dentro do package “Robot” da nossa biblioteca:
b) Crie e inicialize um objeto do tipo “Robot” e inicialize-o. Este processo ira estabelecer a comunicação entre o computador e o robot.
c) Adicione o código para mover o robot 10 vezes:
d) Por fim terá de adicionar uma linha para terminar a comunicação entre o computador e o robot.

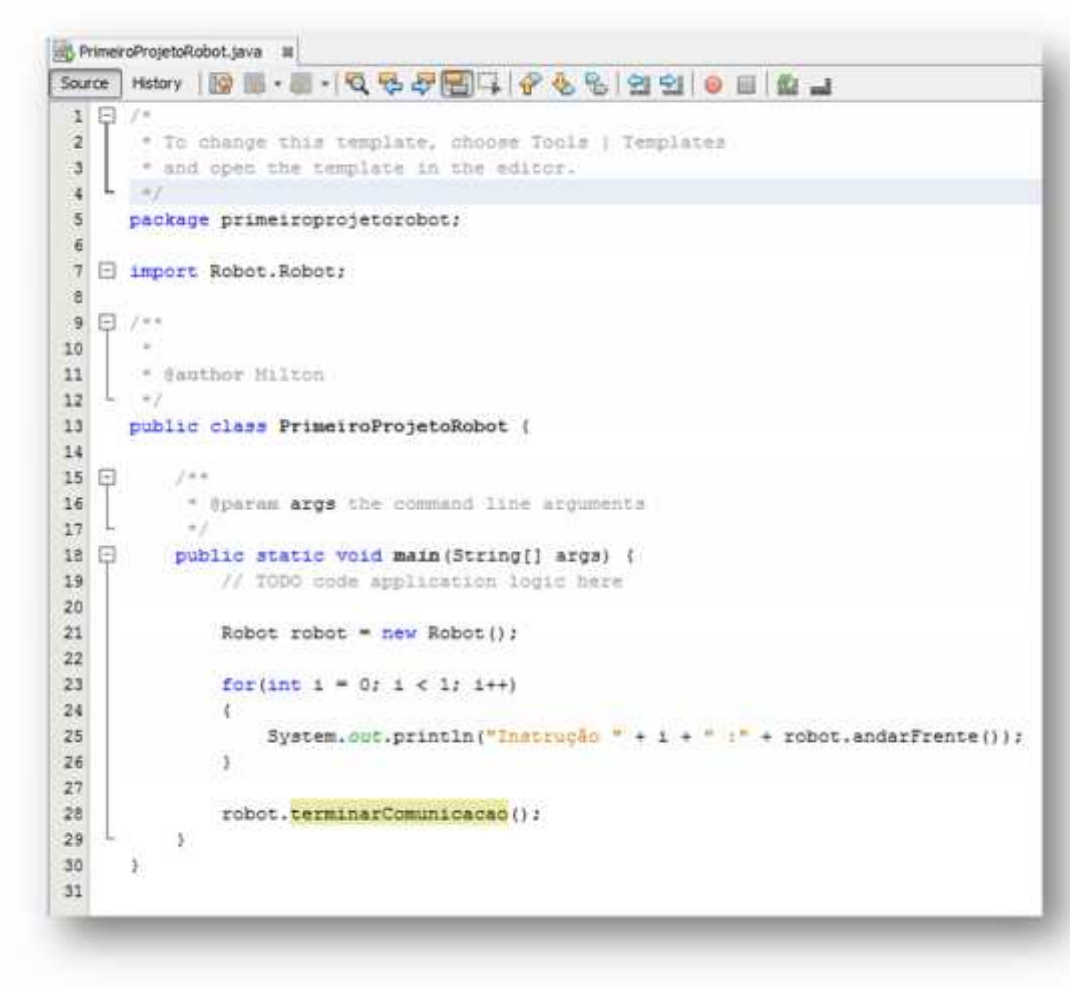


Figura 6.2-18 Ciar um projeto em Netbeans utilizando a biblioteca desenvolvida – passo 8

Para executar o programa escolha “Run” > “Run Project” (F6). Ira obter o seguinte output:

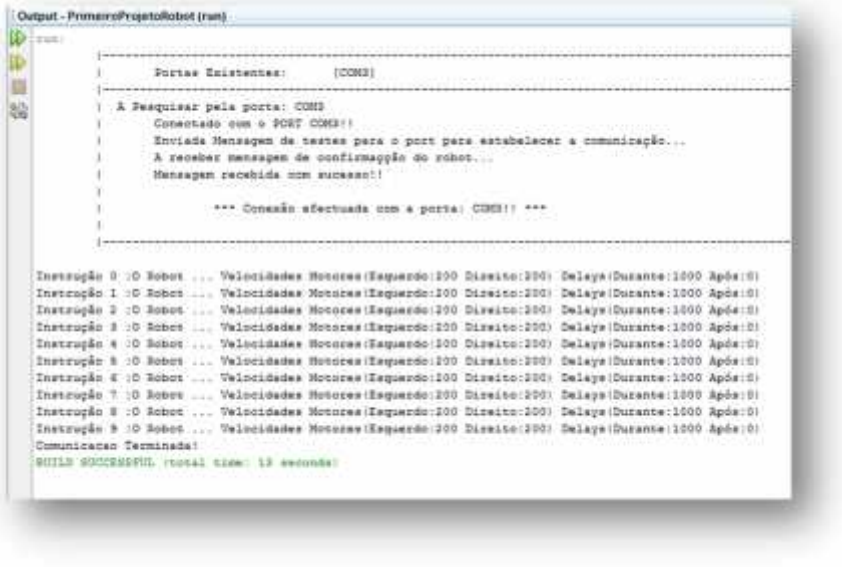


Figura 6.2-19 Ciar projeto em Netbeans Utilizando a biblioteca desenvolvia – passo 9

Nota: Se houver erros de compilação, eles são marcados com glifos vermelhos nas margens esquerda e direita do Editor de Código-fonte. Os glifos da margem esquerda indicam os erros das linhas correspondentes. Os glifos da margem direita mostram todas as áreas do arquivo que apresentam erros, incluindo os erros das linhas que não estão visíveis. É possível passar o rato sobre a marca do erro para ver a descrição deste erro. É possível clicar em um glifo da margem direita para ir para a linha que apresenta o erro.

Criação de um novo projeto para utilização da biblioteca no Eclipse IDE

Primeiro inicie o Eclipse IDE. Faça duplo clique no ficheiro eclipse.exe que se encontra no diretório extraído da pasta Eclipse. Irá surgir uma janela para preencher o local do workspace, que é onde será guardado o novo projeto.

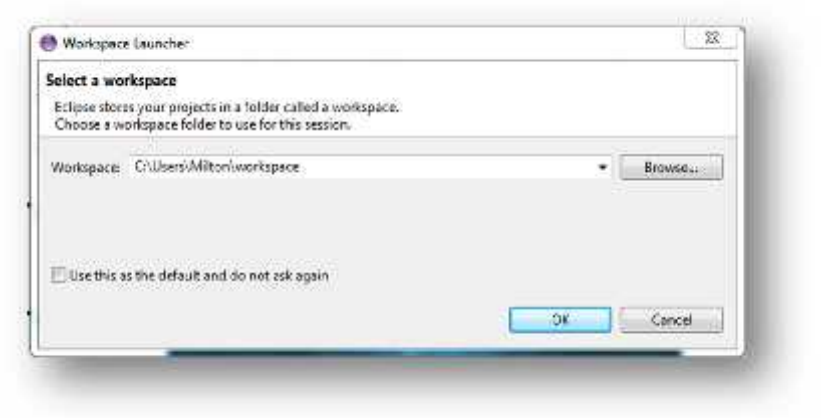


Figura 6.2-20 Ciar projeto em Eclipse utilizando a biblioteca desenvolvida – passo 1

No Eclipse IDE, escolha “New” → “Java Project”, como mostrado na figura abaixo.

Figura 6.2-21 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 2

No assistente “New Java Project” preencha da seguinte maneira:

a. No campo “Project Name”, digite “PrimeiroProjetoRobot”.
b. Deixe o resto dos campos como mostrado na figura e pressione “Finish”.

Figura 6.2-22 Ciar projeto em Eclipse utilizando a biblioteca desenvolvia – passo 3

O projeto é criado e aberto no IDE. Clique com o botão direito em “src” → “New” → “Class”.

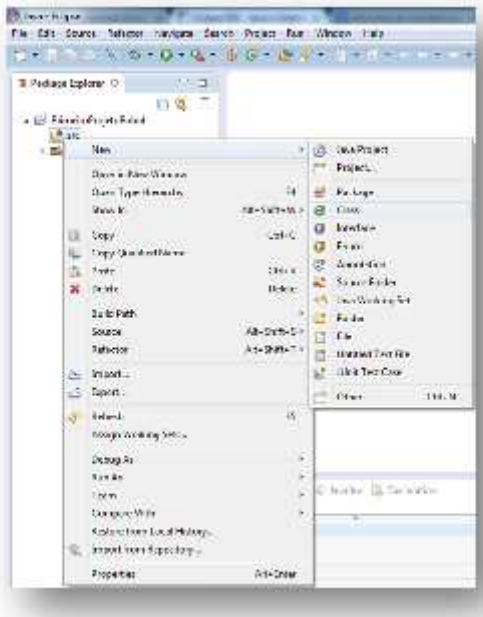


Figura 6.2-23 Ciar projeto em Eclipse utilizando a biblioteca desenvolvia – passo 4

No assistente “New Java Class”, preencha os campos. No campo “Name”, digite “Main”. Deixe marcadas as caixas de seleção “public static void main(String[] args)” e “Inherited abstract methods”.

Por fim, clique em “Finish”.

Figura 6.2-24 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 5

A classe é criada no IDE.

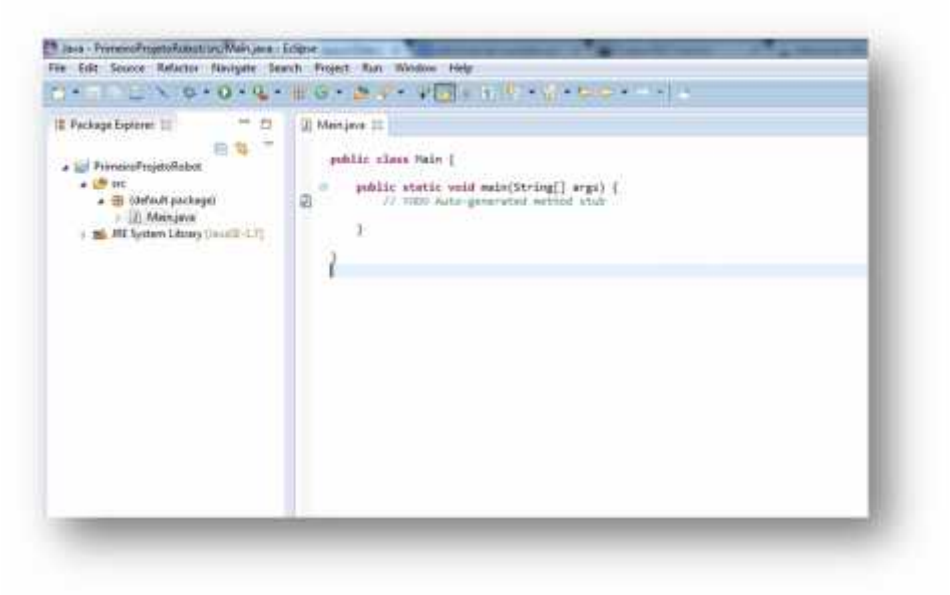


Figura 6.2-25 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 6

Vamos agora adicionar a biblioteca necessária ao projeto para controlarmos o robot. Clique com o botão direito no projeto → “Properties”.

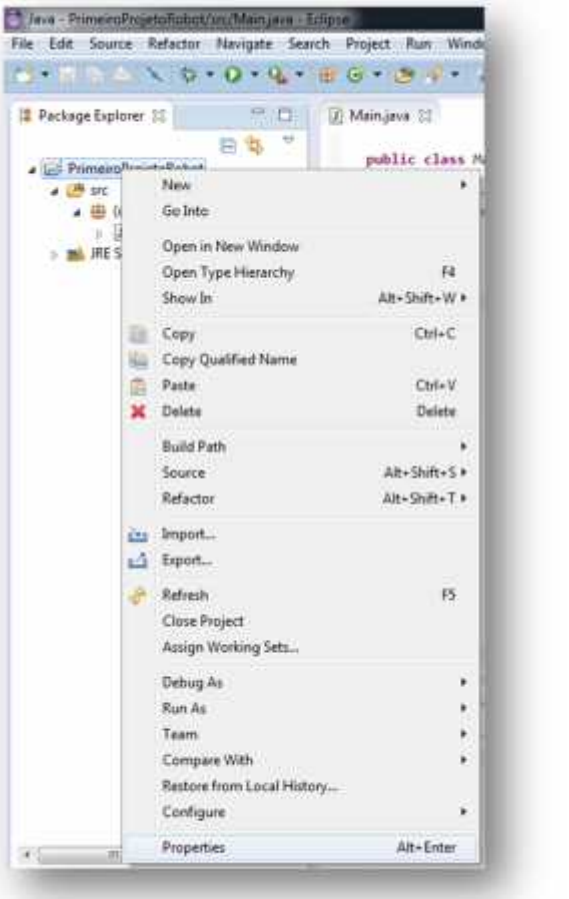


Figura 6.2-26 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvia – passo 7

2. No assistente clique em “Add External JARs…”

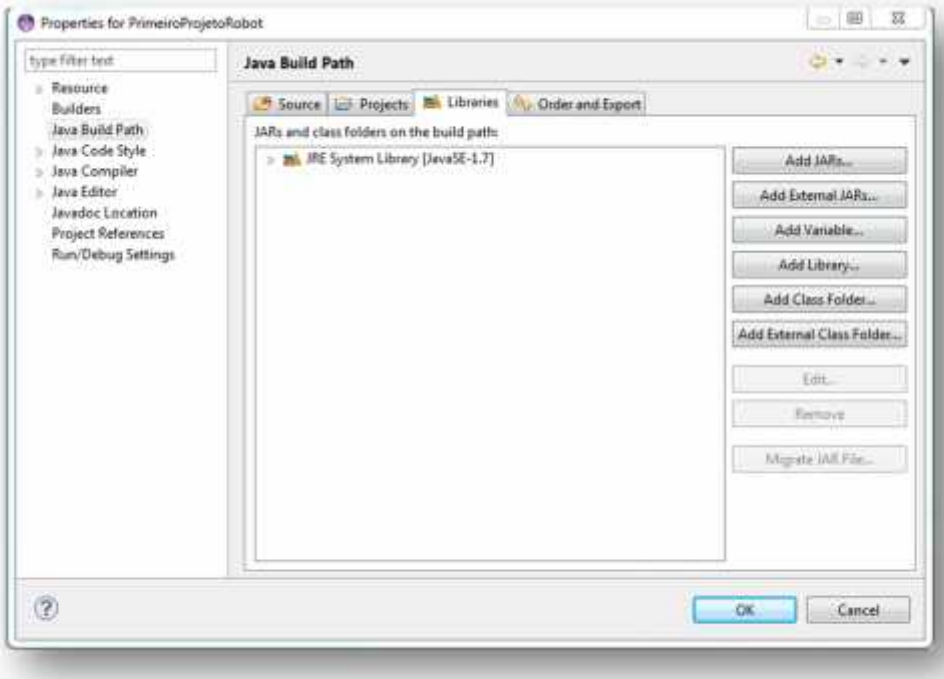


Figura 6.2-27 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 8

Selecione a biblioteca “BSAA.jar”, e clique em “Abrir”.

Figura 6.2-28 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 9

No assistente clique em “Ok” para concluir.

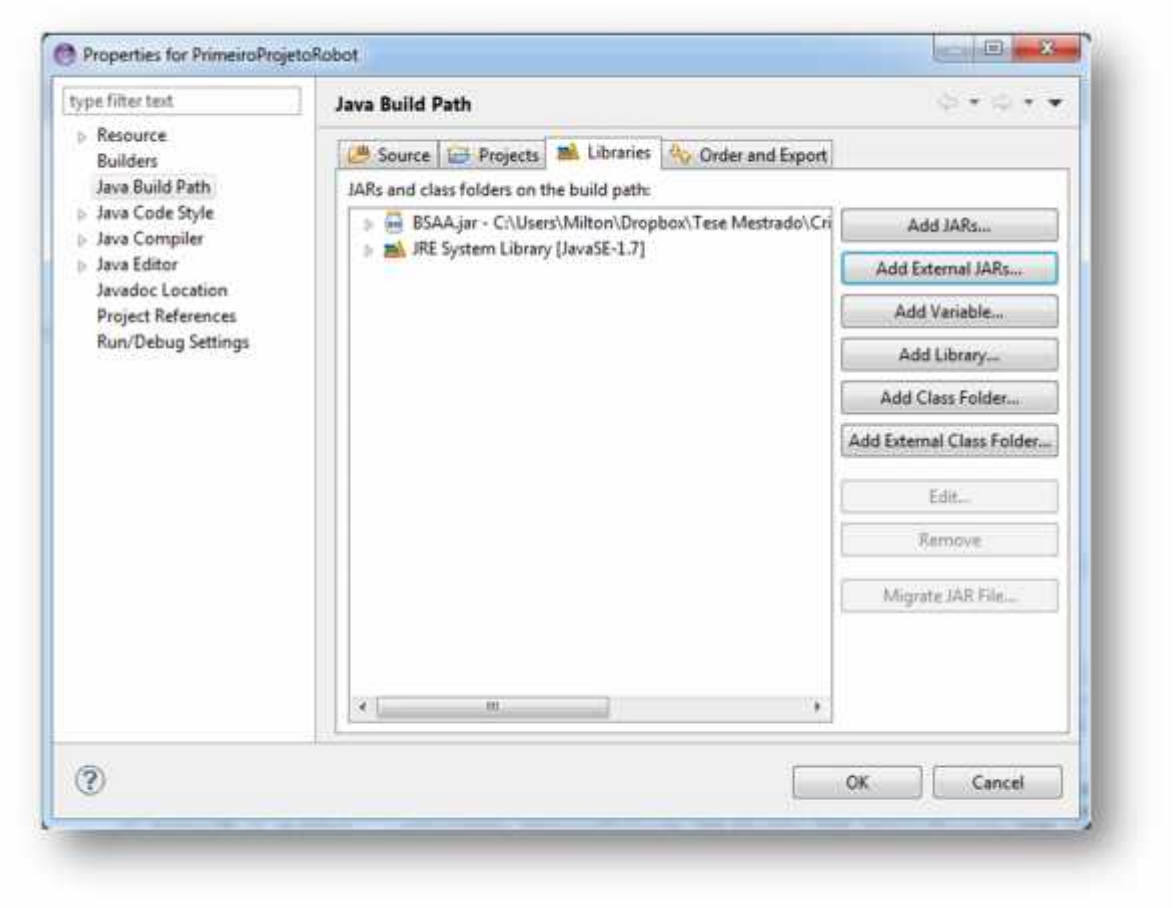


Figura 6.2-29 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 10.1

Agora temos a biblioteca adicionada ao nosso projeto:

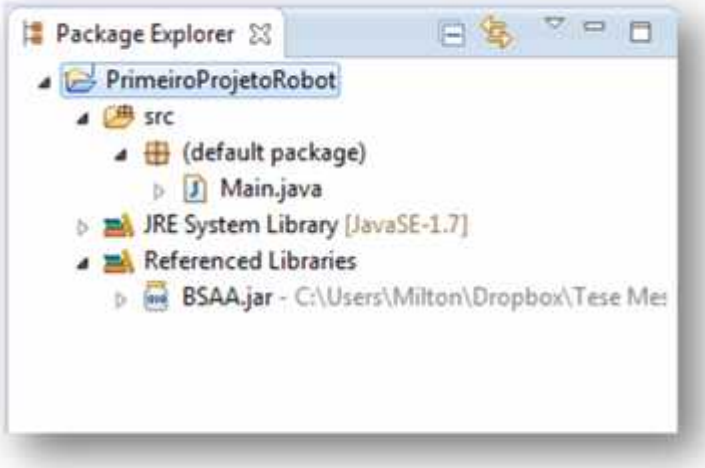


Figura 6.2-30 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 10.2

Para um exemplo simples de mover o robot 10 vezes para a frente utilizando um ciclo de repetição for:

a) Faça o import da classe “Robot” dentro do package “Robot” da nossa biblioteca:
b) Crie e inicialize um objeto do tipo “Robot” e inicialize-o. Este processo irá estabelecer a comunicação entre o computador e o robot.
c) Adicione o código para mover o robot 10 vezes
d) Por fim terá de adicionar uma linha para terminar a comunicação entre o computador e o robot.

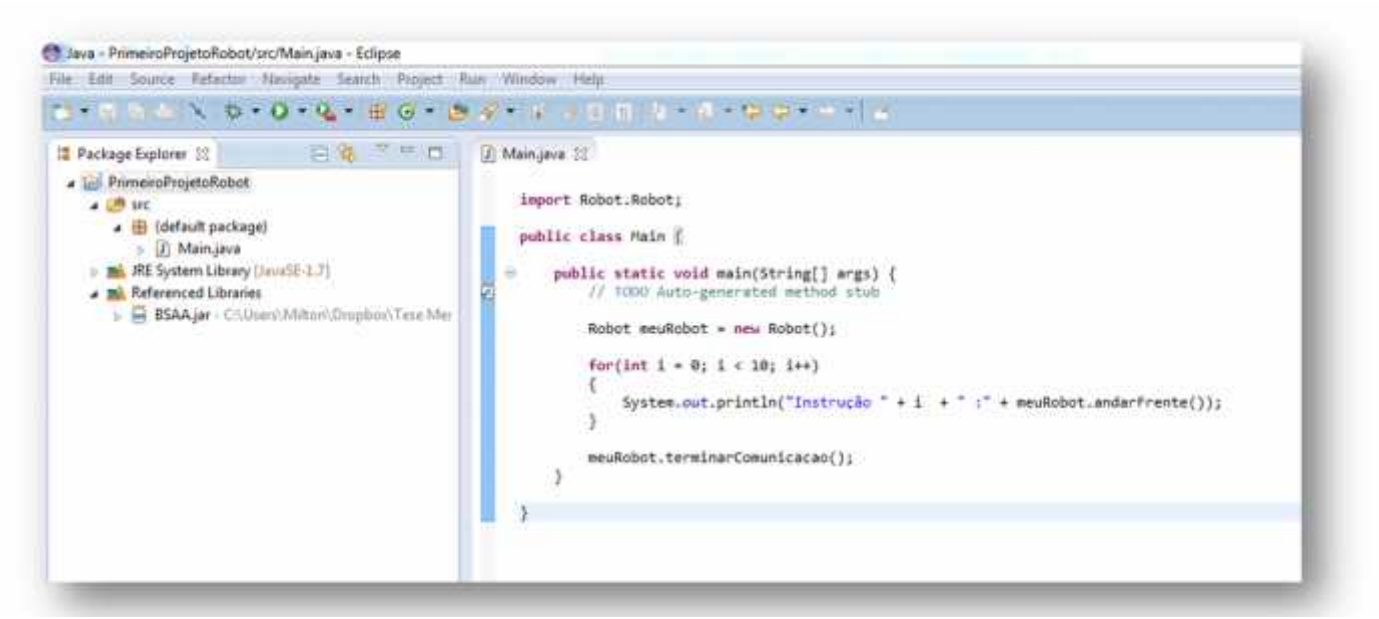


Figura 6.2-31 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 11

Para executar o programa escolha “Run” → “Run” (Ctrl + F11), irá obter o seguinte *output*, escolha quais as classes que deseja guardar e clique “Ok”.

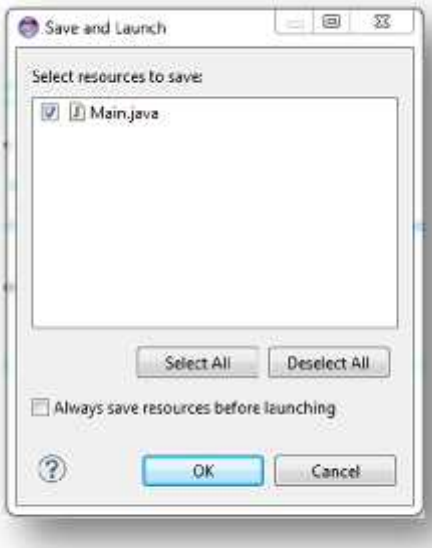


Figura 6.2-32 Ciar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 12

Por fim ira obter o seguinte output:

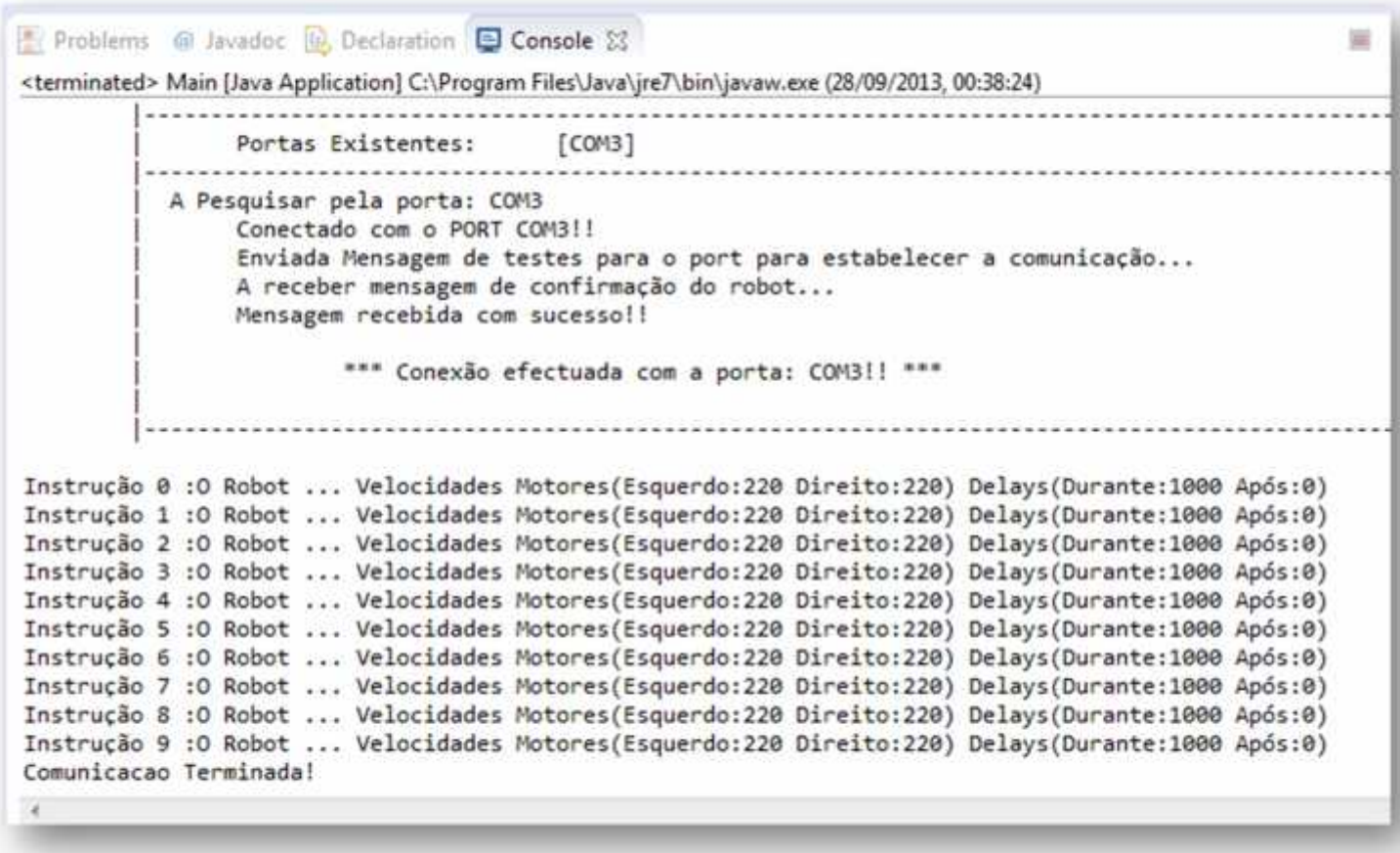


Figura 6.2-33 Criar um projeto em Eclipse utilizando a biblioteca desenvolvida – passo 13

Nota: Se houver erros de compilação, eles são marcados com glifos vermelhos nas margens esquerda e direita do Editor de Código-fonte. Os glifos da margem esquerda indicam os erros das linhas correspondentes. Os glifos da margem direita mostram todas as áreas do arquivo que apresentam erros, incluindo os erros das linhas que não estão visíveis. É possível passar o rato sobre a marca do erro para ver a descrição deste erro. É possível clicar em um glifo da margem direita para ir para a linha que apresenta o erro.