Hugo Emanuel Fonseca Tavares

Concepção e desenvolvimento de pequenos robôs para auto-aprendizagem

Hugo Emanuel Fonseca Tavares

Concepção e desenvolvimento de pequenos robôs para auto-aprendizagem

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica do Doutor Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Robertt Angelo Fontes Valente
Professor auxiliar da Universidade de Aveiro

Professor Doutor Carlos Baptista Cardeira
Professor auxiliar do Instituto Superior Técnico de Lisboa

Professor Doutor Vítor Manuel Ferreira dos Santos
Professor associado da Universidade de Aveiro (orientador)

agradecimentos /
acknowledgements

Este trabalho não poderia ter sido realizado, fundamentalmente, sem a ajuda de várias pessoas. Quero agradecer este trabalho a todas as pessoas que, de uma maneira ou de outra, me ajudaram a concretizá-lo.

Em primeiro lugar, quero prestar os meus, mais sinceros, agradecimentos ao Professor Doutor Vítor Santos, que, ao longo da realização deste trabalho, sempre se mostrou disponível para me orientar e apoiar, com muita paciência e dedicação. Além disso, sem o seu conhecimento, profundo, na área da robótica e electrónica, a realização deste trabalho não seria possível. Foi um enorme prazer poder trabalhar com o Professor.

Agradeço ao Daniel Afonso, ao Emanuel Ávila, ao Eduardo Durana, ao Miguel Oliveira e às restantes pessoas do Laboratório de Automação e Robótica do Departamento de Engenharia Mecânica da Universidade de Aveiro, pela sua ajuda e companheirismo ao longo da realização deste trabalho.

Quero agradecer, também, ao Eng. António Festas a sua ajuda e disponibilidade no processo de concepção e fabrico de várias peças dos robôs.

Deixo um grande abraço ao Miguel, à Xana, à Beta e todos os outros meus amigos, pela sua amizade e por que, durante a realização deste trabalho, me proporcionaram momentos de bem estar.

À Fatinha quero agradecer a paciência, disponibilidade, compreensão, amor, carinho e a ajuda que me deu para poder realizar este trabalho. Um grande beijinho.

Quero, também, agradecer ao meu irmão, não só pela ajuda e conselhos que me deu para a realização deste trabalho, mas também pela ajuda e conselhos que me deu ao longo da vida. Um grande abraço!

Finalmente, quero agradecer aos meus Pais por, além de me criarem, cuidarem e de me terem transmitido os melhores valores que uma pessoa pode ter, me terem dado a oportunidade e condições de poder finalizar o meu curso e este trabalho. A eles devo tudo o que sou e tudo o que alcanço ao longo da minha vida. Muito Obrigado.

Palavras-chave

Robôs autónomos, *reinforcement learning*, modularidade, redundância sensorial.

Resumo

O objectivo principal deste trabalho foi o de projectar, desenvolver e testar pequenos robôs autónomos orientados para aprendizagem de comportamentos de locomoção. O projecto preconizou também uma abordagem modular para possibilitar a adição de novos dispositivos.

Embora os robôs desenvolvidos possam ser programados com vários métodos de aprendizagem, a opção escolhida foi a de usar aprendizagem por *reinforcement learning*, que é um tipo de aprendizagem não supervisionada. Neste tipo de aprendizagem é o robô que avalia o resultado das acções que executa; da avaliação resulta uma "recompensa", que pode ser positiva ou negativa, consoante o resultado da acção. Para um robô avaliar as acções que executa deve estar dotado de dispositivos que permitam a obtenção de informação sobre o ambiente que o rodeia. Uma grande parte dos pequenos robôs autónomos comuns possui apenas sensores de distância, sensores de movimento e alguns câmaras digitais para aquisição de imagens, entre outros dispositivos. No entanto, são menos comuns os que possuem redundância sensorial e sensores de contacto, e isso foi levado em consideração no desenvolvimento destes robôs.

Nestas condições, um robô deve experimentar o maior número possível de acções, para poder aprender qual a melhor opção que deve executar perante cada situação. Isto implica que o robô possa entrar em contacto físico com o ambiente em que se encontra. Nesse sentido, procurou-se também projectar robôs robustos.

Para os testes de aprendizagem, os robôs foram programados com o método de aprendizagem SARSA (*State-Action-Reward-State-Action*), tendo-se fixado o objectivo de navegar num dado ambiente sem colidir.

Os resultados foram bastante satisfatórios, tendo-se concluído que os robôs aprenderam de facto a movimentarem-se sem colidir. Embora tenha havido casos em que a percepção dos robôs poderia corresponder a diferentes situações de ambiente circundante, e que em alguns casos condicionou a aprendizagem, os objectivos de robustez e capacidade de aprendizagem foram alcançados. O trabalho deixou em aberto inúmeras possibilidades de desenvolvimento futuros na área da aprendizagem robótica.

Keywords Autonomous robots , reinforcement learning, modularity, sensorial redundancy.

Abstract The main goal of this project was to design, develop and test small autonomous robots managed to learn movement behaviors. The project also professed an modular approach to allow the addition of new devices.

Although the developed robots can be programmed with many learning methods, the chosen option was using reinforcement learning, which is unsupervised learning method. In this learning method it is the robot who evaluates the result of the executed actions; from the evaluation results a “reward”, that can be positive or negative, according to the result of the action. For a robot evaluate it’s executed actions it must be outfitted with devices that allows the obtainment of information of the surrounding environment. Most of the common small autonomous robots only possesses distance sensors, movement sensors, and some digital cameras for image acquisition, among other devices. However, fewer are those who have sensorial redundancy and contact sensors, and that was held in consideration in the development of this robots.

In these conditions, a robot should experiment every possible actions to be able to learn which is the best action it must take for every situation. Which means the robot may have physical contact with the surrounding environment. In that line of though it was also attempted to design robust robots.

For the learning tests, the robots were programmed with the SARSA (State-Action-Reward-State-Action) learning method, in which he main goal was to wander in an environment without any colliding.

The results were quite good, which leaded to the conclusion that the robots learned to wander in an environment without colliding. Although, had been cases in which the perception of the robots could match to different situations of the surrounding environment, and that in some cases compromised the robots’s learning, the main goals of strength and learning capability were achieved. This project left open many future developments in the area of learning in robotics.

Conteúdo

Conteúdo i

Lista de Figuras v

Lista de Tabelas ix

1 Introdução 1
- 1.1 Objectivos . . . 1
- 1.2 Enquadramento . . . 2
- 1.3 Estado da Arte . . . 3
  - 1.3.1 Robôs móveis autónomos . . . 3
  - 1.3.2 Pequenos Robôs móveis para aprendizagem . . . 9
  - 1.3.3 *Reinforcement learning* . . . 14

2 Projecto 15
- 2.1 Projecto Mecânico . . . 16
  - 2.1.1 Chassi . . . 16
  - 2.1.2 Dimensionamento dos apoios do chassi . . . 17
  - 2.1.3 Suportes . . . 19
  - 2.1.4 Sistema de pára-choques . . . 19
  - 2.1.5 Sistema de locomoção . . . 22
  - 2.1.6 Montagem do Conjunto . . . 22
- 2.2 Projecto Electrónico . . . 25

3 Software e Firmware 29
- 3.1 Arquitectura do SALbot . . . 29
- 3.2 Firmware e funcionamento dos dispositivos . . . 31
  - 3.2.1 Sensores de contacto . . . 31
  - 3.2.2 Sensores de distância infra-vermelhos . . . 31
  - 3.2.3 Sensores de distância ultra-sons . . . 33
  - 3.2.4 Servomotores . . . 34
  - 3.2.5 Memória EEPROM externa . . . 35
  - 3.2.6 XBee . . . 36
- 3.3 Protocolos de comunicação e comando . . . 38
  - 3.3.1 Sinal PW de comando dos servomotores . . . 38
  - 3.3.2 Comunicação $I^2C^{™}$ com o dispositivo de EEPROM externo . . . 39
    - 3.3.2.1 Endereçamento dos dispositivos . . . 39

3.3.2.2 Operações de escrita . . . 40
3.3.2.3 Operações de leitura . . . 40
3.3.3 Protocolo de comunicação entre o SALbot e o PC . . . 41
3.3.3.1 Mensagem de modo de operação . . . 42
3.3.3.2 Mensagem de Início/Paragem . . . 43
3.3.3.3 Mensagem de pedido de dados . . . 43
3.3.3.4 Mensagem de envio do estado dos sensores de distância . . . 44
3.3.3.5 Mensagem de envio do estado dos sensores de contacto . . . 44
3.3.3.6 Mensagem de envio de um elemento da tabela de aprendizagem . 44
3.3.3.7 Mensagem de envio do progresso de aprendizagem . . . 45
3.3.3.8 Mensagem de envio da acção executada . . . 46
3.4 Software . . . 46
3.4.1 Algoritmo de aprendizagem . . . 46
3.4.2 Acções e estados . . . 47
3.4.3 Gerador de números aleatórios . . . 50
3.4.4 Modos de funcionamento . . . 50
3.5 Interface no PC . . . 54

4 Testes e Resultados **57**
4.1 Ambiente de testes . . . 57
4.2 Resultados . . . 58
4.2.1 Teste de aprendizagem para o ambiente pequeno . . . 59
4.2.2 Teste de aprendizagem para o ambiente médio . . . 62
4.2.3 Teste de aprendizagem para o ambiente completo . . . 66
4.2.4 Comparativo entre os diferentes testes de aprendizagem . . . 69
4.2.5 Resultados em operação no modo autónomo . . . 71

5 Conclusões e trabalhos futuros **77**
5.1 Conclusões . . . 77
5.2 Trabalhos futuros . . . 79

A Algoritmos de Reinforcement Learning **81**
A.1 *Markov Decision Processes* . . . 81
A.2 Funções de Valor . . . 82
A.3 Métodos *Model-free* . . . 84
A.3.1 *Monte Carlo Control* . . . 84
A.3.2 *on-policy Monte Carlo Control* . . . 86
A.3.3 *off-policy Monte Carlo Control* . . . 88
A.3.4 TD(0) . . . 90
A.3.5 *Q-learning* . . . 90
A.3.6 SARSA - *State-Action-Reward-State-Action* . . . 92
A.4 Métodos *Model-based* . . . 93
A.4.1 *Policy Iteration* . . . 93
A.4.2 *Value Iteration* . . . 94

B Desenhos técnicos **97**
B.1 Base do chassi . . . 99
B.2 Suporte dos sensores de contacto . . . 101
B.3 Suporte dos sensores frontais . . . 103
B.4 Suporte do sensor traseiro . . . 105
B.5 Peça de fixação do suporte dos sensores frontais . . . 107
B.6 Pilar de suporte dos servos . . . 109
B.7 Pilar de suporte dos sensores de distância por infra-vermelhos laterais . . . 111
B.8 Barra de contacto frontal . . . 113
B.9 Barra de contacto lateral . . . 115
B.10 Barra de contacto lateral-frontal . . . 117
B.11 Barra de contacto traseira . . . 119
B.12 Canto frontal-lateral . . . 121
B.13 Canto traseiro-lateral . . . 123
B.14 Tela de latex frontal . . . 125
B.15 Tela de latex lateral . . . 127
B.16 Tela de latex lateral-frontal . . . 129
B.17 Tela de latex traseira . . . 131
B.18 Tela de latex para os cantos . . . 133
B.19 Suporte dos apoios esféricos . . . 135

C Placas de circuito impresso **137**
C.1 Traçado do circuito impresso das Placas . . . 137
C.2 Esquemas eléctricos . . . 139
C.2.1 Schematic SALbot Main . . . 141
C.2.2 Schematic SALbot Aux . . . 143
C.2.3 Schematic SALbot Com . . . 145
C.3 Imagens das placas . . . 147

D Imagens dos robôs **151**

Referências **153**

Lista de Figuras

1.1 Robôs exploradores de Marte: Spirit e Sojourner da NASA. 3
1.2 (a) Robô da Plustech; (b) Robô UAV (*Unmanned Air Vehicle*) Shadow 200 da Força Aérea Americana, este robô foi utilizado recentemente em missões no Iraque; (c) Robô AUV (*Autonomous Underwater Vehicle*) Odyssey do MIT; (d) Robô Versatrax 150 da Inuctun Services Ltd. 4
1.3 (a) Protótipo Cutlass da Rheinmetall AG, um robô que foi concebido para desactivar engenhos explosivos para substituir o robô do exército Britânico *Wheelbarrow*; (b) Robô Pioneer, este robô foi utilizado para uma missão em Chernobyl. 5
1.4 (a) Robô protocolar Robina da Toyota; (b) Robô AGV (*Autonomous Ground Vehicle*) da Transbotics, este robô utiliza feixes de laser para navegar autonomamente, obtendo um mapa bi-dimensional exacto do ambiente que o rodeia, podendo, assim, evitar os obstáculos; (c) Robô doméstico que trata da lida da casa, concebido pela colaboração da Toyota com o IRT (*Information and Robot Technology Research Initiative*) da Universidade de Tokyo. 6
1.5 (a) Robô ASIMO da Honda; (b) Robôs QRIO e AIBO da Sony; (c) Robô Roomba® da iRobot®. 7
1.6 (a) Robô CB^2; (b) Robô icub. 8
1.7 (a) Robô Khepera III da K-team Corporation; (b) Robô Alice desenvolvido na EPFL; (c) Robô e-puck também desenvolvido na EPFL; (d) Robô Miabot Pro da Merlin Systems Corp. Ltd. 10
1.8 (a) Robô AmigoBot™ da Mobile $Robots_{INC}$; (b) Robôs Pioneer 3-AT e 3-DX da Mobile $Robots_{INC}$. 11
1.9 (a) Robô *Starfish* da Universidade de Cornell; (b) Robôs Cyber Rodent da ATR *International*. 12

2.1 Robô MarkIII. 16
2.2 Diagrama de corpo livre para o cálculo do somatório dos momentos. 18
2.3 Zonas de impacto. 20
2.4 Possibilidades de detecção de colisão pelo sistema de pára-choques nos cantos. . . 21
2.5 Perspectiva explodida do chassi do SALbot. 23
2.6 Perspectiva explodida do SALbot. 24
2.7 Placa de circuito principal do SALbot. 26
2.8 Placa de circuito auxiliar do SALbot. 27
2.9 Placa de interface de comunicação. 27

3.1 Arquitectura do sistema do SALbot. 30

3.2 Fim de curso DB3C-B1RC. 31
3.3 Sensor de distância infra-vermelhos Sharp GP2D120. 31
3.4 Gráfico dos valores do sinal analógico em função da distância do sensor GP2D120. . 32
3.5 Gráfico do fabricante da voltagem em função da distância do GP2D120.[26] 32
3.6 Sensor de distância ultra-sons LV-MaxSonar®-EZ0™. 33
3.7 Servo Futaba® S3003. 34
3.8 Unidade de EEPROM de interface série 24LC1025 da Microchip. (a) Fotografia do 24LC1025; (b) *Pinout* do 24LC1025. 35
3.9 Módulo XBee XB24-AWI-001 da Digi®. 36
3.10 Sinal PW do servo S3003 da Futaba®. 38
3.11 Composição da sequência de transferência de dados no I^2C™. 39
3.12 Composição da sequência de *bits* do endereçamento de uma mensagem por I^2C™. . 40
3.13 Topologia da mensagem de comunicação entre o SALbot e o computador. 41
3.14 Descrição gráfica do *byte* de verificação. 42
3.15 Mensagem de alteração do modo de operação do SALbot. 42
3.16 Mensagem de Início/Paragem do SALbot. 43
3.17 Mensagem de pedido de dados. 43
3.18 Mensagem de envio do estado dos sensores de distância. 44
3.19 Mensagem de envio do estado dos sensores de contacto. 44
3.20 Mensagem de envio de um elemento da tabela de aprendizagem. 45
3.21 Mensagem de envio do progresso de aprendizagem. 45
3.22 Mensagem de envio da acção executada pelo SALbot. 46
3.23 Representação dos níveis dos sensores de distância. 48
3.24 Representação das várias acções do SALbot. 49
3.25 Geração de: (a) 1000 acções aleatórias; (b) 10000 acções aleatórias. 50
3.26 Fluxograma da rotina *Main*. 51
3.27 Fluxograma das rotinas: (a) Inicialização do SALbot; (b) Inicialização da Aprendizagem. 52
3.28 Fluxograma da rotina de Aprendizagem. 53
3.29 Fluxograma da rotina de Processamento de mensagens. 53
3.30 Secção, da interface gráfica, de monitorização dos sensores e de comando do modo de operação do SALbot2. 54
3.31 Secção, da interface gráfica, de monitorização e comando da aprendizagem do SALbot1. 55

4.1 Ambiente de testes dos SALbot. 57
4.2 (a) 1/4 do ambiente de testes; (b) 1/2 do ambiente de testes; (c) Ambiente de testes completo. 58
4.3 Representação do estado 8800 do SALbot. 59
4.4 Representação do estado 416 do SALbot. 60
4.5 Representação do estado 6402 do SALbot. 62
4.6 Representação do estado 1809 do SALbot. 63
4.7 Representação do estado 7556 do SALbot. 64
4.8 Representação do estado 592 do SALbot. 65
4.9 Representação do estado 5921 do SALbot. 66
4.10 Representação do estado 2368 do SALbot. 67
4.11 Representação do estado 64 do SALbot. 68

4.12 Comparação entre os testes dos gráficos do número de colisões ao longo das iterações. 70
4.13 Número de colisões em intervalos de 100 iterações para os testes de aprendizagem. 71
4.14 Comparação entre etapas da aprendizagem do número de colisões no modo autónomo no ambiente pequeno. 72
4.15 Comparação entre etapas da aprendizagem do número de colisões no modo autónomo no ambiente médio. 73
4.16 Comparação entre etapas da aprendizagem do número de colisões no modo autónomo num ambiente completo. 73
4.17 Número de colisões em intervalos de 100 acções para o modo autónomo no ambiente pequeno. 74
4.18 Número de colisões em intervalos de 100 acções para o modo autónomo no ambiente médio. 74
4.19 Número de colisões em intervalos de 100 acções para o modo autónomo no ambiente completo. 75

C.1 Traçado do circuito impresso da placa principal do SALbot. 137
C.2 Traçado do circuito impresso da placa auxiliar do SALbot. 138
C.3 Traçado do circuito impresso da placa de interface de comunicação. 138
C.4 Imagem foto-realista da placa principal do SALbot. 147
C.5 Imagem foto-realista da placa auxiliar do SALbot. 148
C.6 Imagem foto-realista da placa de interface de comunicação. 148
C.7 Imagem da placa de interface de comunicação. 149

D.1 Imagem foto-realista dos SALbot. 151
D.2 Imagem dos SALbot. 152
D.3 Logótipo do SALbot. 152

Lista de Tabelas

3.1 Tabela de aprendizagem do SALbot. 49

4.1 Evolução da tabela de aprendizagem para o estado 8800 do teste para 1/4 do ambiente. 60
4.2 Evolução da tabela de aprendizagem para o estado 416 do teste para 1/4 do ambiente. 61
4.3 Evolução da tabela de aprendizagem para o estado 6402 do teste para 1/4 do ambiente. 61
4.4 Evolução da tabela de aprendizagem para o estado 1809 do teste para 1/2 do ambiente. 63
4.5 Evolução da tabela de aprendizagem para o estado 7556 do teste para 1/2 do ambiente. 64
4.6 Evolução da tabela de aprendizagem para o estado 592 do teste para 1/2 do ambiente. 65
4.7 Evolução da tabela de aprendizagem para o estado 5921 do teste para a totalidade do ambiente. 67
4.8 Evolução da tabela de aprendizagem para o estado 2368 do teste para a totalidade do ambiente. 68
4.9 Evolução da tabela de aprendizagem para o estado 64 do teste para a totalidade do ambiente. 69
4.10 Comparação das tabelas de aprendizagem entre os diferentes testes de aprendizagem. 70

A.1 Pseudo-código do *Monte Carlo ES*, um algoritmo do método *Monte Carlo Control* que usa a exploração inicial.[30] 86
A.2 Tipos de estratégia de exploração *soft* 87
A.3 Pseudo-código do algoritmo do método *on-policy Monte Carlo Control*.[30] 88
A.4 Pseudo-código do algoritmo ε-*soft* do método *off-policy Monte Carlo Control*.[30] . 89
A.5 Pseudo-código do algoritmo TD(0).[30] 90
A.6 Pseudo-código do algoritmo *Q-learning*.[30] 91
A.7 Pseudo-código do algoritmo SARSA.[30] 92
A.8 Pseudo-código do algoritmo *Policy Iteration*.[30] 94
A.9 Pseudo-código do algoritmo *Value Iteration*.[30] 95

Capítulo 1

Introdução

"O único homem isento de erros, é aquele que não arrisca acertar."

Albert Einstein

1.1 Objectivos

O principal objectivo deste trabalho é projectar, desenvolver e construir pequenos robôs orientados para aprendizagem de comportamentos de locomoção.

Pretende-se implementar robôs que se possam deslocar livremente num meio confinado e que com ele possam interagir. Essa interacção será importante para a aprendizagem e, por esse motivo, procurou-se projectar uma plataforma de modo a que os robôs possam obter dados suficientes dessa interacção para desenvolver aprendizagem de forma autónoma.

Uma vez que se pretende que os robôs aprendam com os resultados de todas as suas acções, a sua robustez é importante para que não se danifiquem em caso de colisão. Por outro lado, também convém que não danifiquem o meio onde se encontram, pelo que devem possuir alguma flexibilidade.

Para além de integrar os diversos dispositivos sensoriais e de actuação, o *hardware* deve possibilitar a utilização de *software* de aprendizagem. A modularidade do hardware é também um factor importante, pois possibilita a expansão de outros dispositivos que possam ser uma mais valia para aprendizagem dos robôs, ou até para que estes possam ter outro tipo de aplicação no futuro.

1.2 Enquadramento

A inteligência artificial (IA) é, cada vez mais, uma área de interesse no ramo da robótica. A IA é uma área de pesquisa da ciência da computação que se dedica ao desenvolvimento de métodos ou dispositivos computacionais que tenham ou simulem capacidades Humanas, como, por exemplo, raciocinar, tomar opções. Existem vários sub-domínios da IA, e é muito comum, em pesquisas na área da robótica, interligar alguns desses sub-domínios. Por exemplo, existem algoritmos de *Machine learning* (aprendizagem de máquina) que recorrem a redes neuronais.

No caso deste trabalho, sendo um dos objectivos desenvolver robôs para aprendizagem, naturalmente que serão abordadas as áreas de *Machine learning*. Embora se pretenda conceber robôs autónomos móveis, não se pretende nem fará qualquer planeamento de movimento.

Nas últimas décadas, o *Machine learning* tem sido uma área muito investigada. Talvez pelo facto de a máquina aprender, as tarefas que tem de realizar em determinadas situações poder ser uma vantagem. Por exemplo, um robô com muitos sensores que tenha de realizar múltiplas tarefas em vários tipos de ambientes implica um problema muito complexo que requer uma abordagem para cada tipo de tarefa, para cada tipo de ambiente e para cada estado dos diversos sensores. Desenvolver um programa para o robô realizar essas tarefas é um trabalho árduo. Nesta situação são evidentes as vantagens do *Machine learning*: em vez de desenvolver um programa para o robô realizar determinadas tarefas, para vários tipos de situações, apenas é necessário desenvolver um programa de aprendizagem que permita um robô aprender o tipo de tarefas que tem de realizar em determinados ambientes, poupando assim tempo e trabalho em contemplar todos os estados dos sensores.

Na área de *Machine learning* existe um tipo de aprendizagem denominado de *reinforcement learning*, no qual a máquina aprende através de uma aprendizagem não supervisionada. Isto é, em vez de se fornecer exemplos à máquina, ou avaliar as acções que a máquina tomou, para aprender, é a própria máquina que avalia o resultado das acções que realiza.

O facto de ser a máquina a avaliar, por ela própria, o resultado das suas acções pode ser vantajoso. Um ser humano, enquanto criança, é ensinado pelos pais e/ou outras pessoas (aprendizagem supervisionada), no entanto, nem todos os seres humanos são iguais, uns, por exemplo, são mais altos que outros. Se uma criança fosse muito diferente de todos os outros seres humanos, a experiência desses não seria significativa, comparada com a experiência adquirida por essa criança, em relação a uma qualquer situação.

De facto, os seres humanos são diferentes e têm maneiras diferentes de agir perante certo tipo de situações; devido, em parte, pelo que lhes é ensinado, mas em grande parte, por aquilo que aprendem por eles próprios. A capacidade para aprender torna cada ser humano único, permitindo-lhes assim adaptarem-se a diferentes situações.

Extrapolando isto para os robôs, são claras as diversas aplicações que a aprendizagem pode ter: Um carro autónomo que aprende a conduzir nas estradas, um robô que aprende a socializar com as pessoas, uma casa que aprende os hábitos das pessoas que nela vivem e adapta-se a esses hábitos, comandando as luzes, a temperatura, etc..., entre outras aplicações.

Um robô móvel autónomo, orientado para a aprendizagem, deve possuir um sistema de processamento de algoritmos de aprendizagem, uma unidade de armazenamento de dados para guardar dados da aprendizagem, múltiplos sensores, um sistema de locomoção e uma fonte de energia. Embora todos estes requisitos sejam importantes, os sensores requerem uma atenção especial. Isto porque, é através dos sensores que um robô obtém uma representação do meio que o rodeia, sem eles não seria possível um robô aprender, nem sequer movimentar-se autonomamente.

Os projectos de investigação na área da robótica, de questões de alto nível cognitivo, localização, navegação, entre outras, são realizados recorrendo a robôs de plataforma standard que são adaptados ao ambiente laboratorial. No entanto, apesar de existirem alguns pequenos robôs móveis autónomos para aprendizagem, os que existem, são demasiado dispendiosos, ou não possuem uma plataforma suficientemente modular, que permita a adaptação de mais sensores, ou outro tipo de dispositivos, para permitir realizar estudos de aprendizagem.

1.3 Estado da Arte

1.3.1 Robôs móveis autónomos

Os robôs têm cada vez mais influência na vida dos seres humanos. Principalmente na indústria de produção, onde actualmente, estima-se que os robôs industriais representem cerca de um bilião e meio de euros. Executando movimentos rápidos com alta precisão, os robôs industriais podem realizar tarefas repetitivas numa linha de montagem, tais como pintar, soldar, manipular peças, etc..., permitindo elevar a cadência e eficiência da linha de produção, e a qualidade do produto final. Para além dessas vantagens, os robôs permitem libertar os operários da realização de tarefas monótonas, permitindo-lhes executar outro tipo de tarefas. Porem, este tipo de robôs têm a limitação obvia da falta de mobilidade.



Figura 1.1: Robôs exploradores de Marte: Spirit e Sojourner da NASA.

A investigação de robôs móveis começou nos finais da década de 1960, com o projecto *Shakey* desenvolvido no SRI (*Stanford Research Institute*) pelo grupo de Charlie Rosen. O artigo *"A Mobile Automation: An Application of Artificial Intelligence Techniques"* de N.J. Nilsson no IJCAI em 1969, já englobava percepção, mapeamento, planeamento de movimentos, e noções de arquitectura de controlo. Estes assuntos iriam, de facto, tornar-se o núcleo das investigações de robôs móveis nas décadas seguintes. A década de 1980 prosperou com projectos de robôs móveis. E dado que cedo se sentiu a necessidade de lidar com a realidade do mundo físico, apareceram problemas que estimularam novas direcções de pesquisa, distanciando-se actualmente do conceito original, no qual o robô era apenas uma aplicação de técnicas de IA.[27]

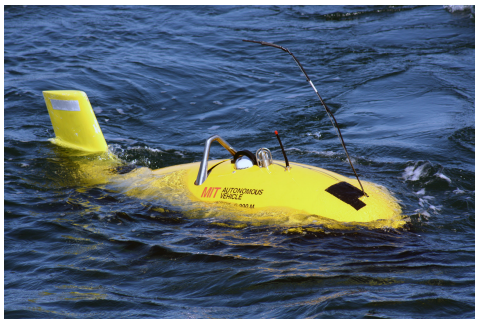
Um robô móvel implica que o mesmo possua um sistema de locomoção, normalmente recorrendo a motores eléctricos ou outro tipo de actuadores. Ao contrário da maior parte dos manipuladores industriais, que estão fixos num determinado local, os robôs móveis têm a capacidade de poder movimentar pelo meio em que se encontram, podendo executar as suas tarefas em vários locais.



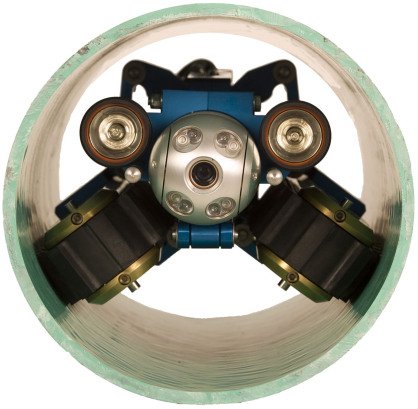
(a)



(b)



(c)



(d)

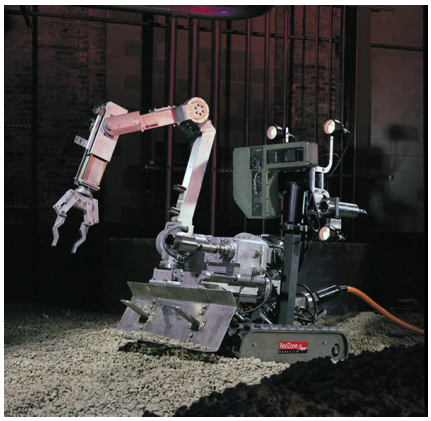
Figura 1.2: (a) Robô da Plustech; (b) Robô UAV (*Unmanned Air Vehicle*) Shadow 200 da Força Aérea Americana, este robô foi utilizado recentemente em missões no Iraque; (c) Robô AUV (*Autonomous Underwater Vehicle*) Odyssey do MIT; (d) Robô Versatrax 150 da Inuctun Services Ltd.

A necessidade da utilização de dispositivos robóticos em ambientes hostis, como por exemplo em Marte, impulsiona a implementação de sistemas de locomoção cada vez mais incomuns (ver figura 1.1). Os robôs teleoperados têm ganho popularidade em terrenos perigosos e/ou inabitáveis (figuras 1.2b, 1.2c, 1.2d, 1.3a e 1.3b). Nalguns casos, o elevado nível de complexidade do sistema de locomoção do robô impossibilita o controlo directo por um operador humano. Por exemplo, no caso do robô da Plustech (figura 1.2a), o operador apenas tem de indicar a direcção para a qual

deseja que o robô se desloque, enquanto que o robô possui um controlo automático da coordenação das pernas, que permite movimentar o robô na direcção desejada pelo operador. Na figura 1.2c pode ver-se um robô aquático que, apesar de possuir motores que lhe permitem uma estabilização autónoma debaixo de água, um operador pode escolher o destino que o submarino deve alcançar. Robôs como os da figura 1.3a e 1.3b, permitem desactivar dispositivos explosivos ou executar outro tipo de tarefas em locais perigosos, sem pôr em risco vidas Humanas.



(a)



(b)

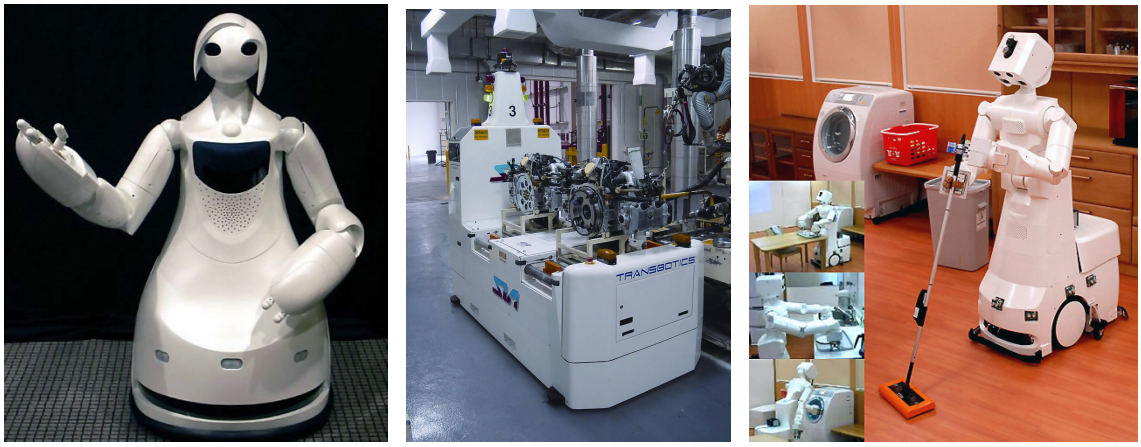
Figura 1.3: (a) Protótipo Cutlass da Rheinmetall AG, um robô que foi concebido para desactivar engenhos explosivos para substituir o robô do exército Britânico *Wheelbarrow*; (b) Robô Pioneer, este robô foi utilizado para uma missão em Chernobyl.

Não existem, no entanto, apenas robôs para serem operados em locais onde o Homem não pode ir. Existem outros que partilham o mesmo espaço habitado pelos humanos (figuras 1.4a, 1.4c, 1.5a, 1.5b e 1.5c). Estes robôs não são úteis por causa da sua mobilidade, mas sim por causa da sua autonomia. De facto, nestes robôs, a capacidade de localização e de navegação sem intervenção humana é superior nestes robôs.

Segundo a definição, um robô é autónomo quando é auto-suficiente, isto é, possui um controlador próprio, não dependendo de um computador central para ser comandado. Um robô autónomo, executa as suas tarefas por ele próprio. Quanto mais complexas forem as tarefas que um robô tenha que executar, maior pode ser a sua autonomia e, normalmente, recorrem ao uso de inteligência artificial para a execução das mesmas.

A autonomia de um robô pode ser uma grande vantagem; se um robô executar as tarefas por ele próprio num sistema e se outras partes do sistema falharem, o robô continuará a executar as suas tarefas. No entanto, num sistema com muitos robôs idênticos, a autonomia pode não ser eficaz. Neste caso, é melhor, economicamente, colocar um programa central a controlar todos os robôs. É desta forma que, normalmente, funcionam os robôs insecto.[13]

Robôs AGV (*Autonomous Ground Vehicle*), distribuem peças por vários pontos de uma linha de montagem autonomamente. Este tipo de robôs tem ganho uma enorme importância na industria de produção. Inicialmente era necessário preparar os pavimentos com fios eléctricos ou magnetos, sobre os quais se guiavam esses robôs. Hoje em dia, existem robôs AGV com sistema de navegação por laser (ver figura 1.4b). Tais robôs recorrem à emissão de feixes laser para navegarem autonomamente, para isso, é apenas necessário colocar reflectores nas paredes, colunas ou máquinas da fábrica, permitindo a reflexão dos fixes laser de volta para o robô, dando-lhe, assim, a posição dos obstáculos.



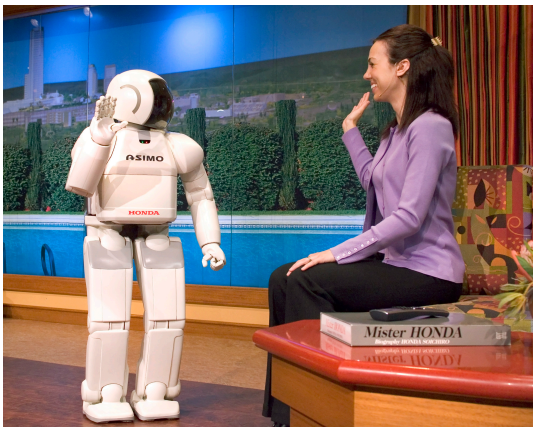
(a) (b) (c)

Figura 1.4: (a) Robô protocolar Robina da Toyota; (b) Robô AGV (*Autonomous Ground Vehicle*) da Transbotics, este robô utiliza feixes de laser para navegar autonomamente, obtendo um mapa bi-dimensional exacto do ambiente que o rodeia, podendo, assim, evitar os obstáculos; (c) Robô doméstico que trata da lida da casa, concebido pela colaboração da Toyota com o IRT (*Information and Robot Technology Research Initiative*) da Universidade de Tokyo.

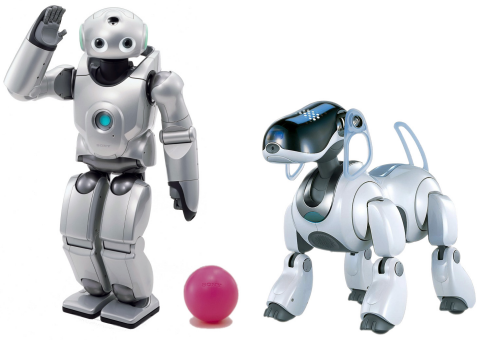
Nos últimos anos, têm vindo a ser desenvolvidos, cada vez mais, robôs para ajudar os humanos nas tarefas domésticas (figuras 1.4c e 1.5c). De facto, o Roomba® da iRobot® tem sido um enorme sucesso de vendas, tendo sido vendidas mais de três milhões de unidades desde 2002, ano em que foi introduzido no mercado. Isto significa que, um em cada quinhentos lares já é limpo pelo Roomba®. Este robô aspirador, aspira, por vácuo, o chão de uma casa autonomamente até quatro divisões por carga. O utilizador apenas tem de indicar a área que o robô tem de limpar por dia e o robô, depois de limpar, ou quando precisa de recarregar as baterias, volta à sua base.

Robôs de estudo como o ASIMO (figura 1.5a), o Robomaid (figura 1.4c) ou a Robina (figura 1.4a), dão uma ideia de como serão os robôs no futuro. Mas é também graças a estes e outros protótipos, que a robótica vem evoluindo ao longo dos anos. De facto o ASIMO, actualmente o mais avançado robô humanoide do mundo, deu um "passo" enorme na área da robótica, ou não fosse o seu nome *Advanced Step in Innovative MObility*. Isto porque, foi o primeiro robô a andar dinamicamente como os humanos, caminha para a frente e para trás, vira-se enquanto caminha, e até sobe e desce escadas. Alem disso possui avançadas tecnologias de inteligência, nomeadamente reconhecimento facial e vocal, mapeamento digital do ambiente que o rodeia, reconhecimento de postura e gestos, e ainda ligação à internet.

Mas os robôs não servem só para executar tarefas quotidianas, industriais, ou perigosas. Existem robôs autónomos, muito populares hoje em dia, que têm como função entreter miúdos e graúdos. Os robôs QRIO e AIBO da Sony (figura 1.5b), são um bom exemplo disso. O QRIO (lê-se [quiuriô] *Quest for cuRIOsity*, originalmente designado de *Sony Dream Robot* ou SDR) reconhece vozes e faces, permitindo-o recordar as pessoas como também os seus gostos. Além de ser um robô humanoide bípede autónomo, o QRIO foi considerado, pelo *Guinness World Records* (edição de 2005), como sendo o primeiro robô bípede com a capacidade de correr (andar enquanto, periodicamente, ambas as pernas estão acima do chão simultaneamente), conseguindo correr a 0.83 Km/h. Foram ainda feitas várias experiências da interacção do QRIO com crianças, das quais



(a)



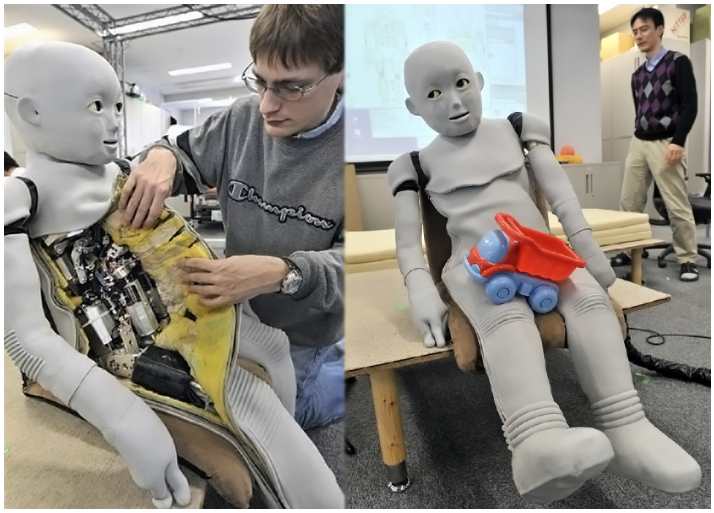
(b)



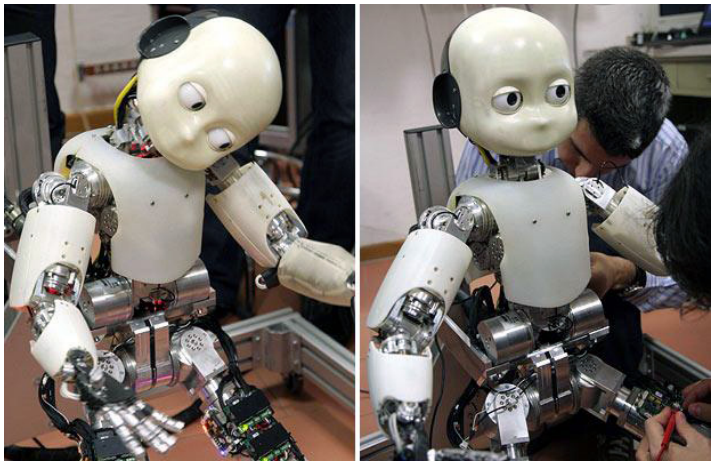
(c)

Figura 1.5: (a) Robô ASIMO da Honda; (b) Robôs QRIO e AIBO da Sony; (c) Robô Roomba® da iRobot®.

se obtiveram várias conclusões. A experiência conduzida por Fumihide Tanaka, Aaron Cicourel e Javier Movellan da Universidade da Califórnia San Diego, nos Estados Unidos, foi o primeiro estudo da interacção entre crianças e robôs de longo período.[31] Esta experiência consistiu em colocar um QRIO numa sala com crianças, com idades entre os dezoito meses e os dois anos, durante cinco meses. Durante a experiência, esperava-se que o entusiasmo das crianças, com o robô, desaparecesse ao fim de poucas horas, que era o que se passava com outros robôs. Mas, o que se verificou, foi que as crianças se apegaram ao robô durante várias semanas, eventualmente, interagindo com o QRIO do mesmo modo que o faziam com as outras crianças. Um dos problemas com os robôs anteriores, era que as pessoas rapidamente se aborreciam com eles. E, como este estudo comprovou, o QRIO manteve o interesse das crianças, o que possibilita, a este robô, ou outros com as mesmas características, vários tipos de aplicações, como, por exemplo, ajudar crianças autistas. Muito embora a Sony tenha parado o desenvolvimento do QRIO e descontinuado o AIBO, estes deixaram uma marca importantíssima para o desenvolvimento de robôs, para crianças, no futuro.



(a)



(b)

Figura 1.6: (a) Robô CB^2; (b) Robô icub.

Robôs como o CB^2 (figura 1.6a), iCub (figura 1.6b), e outros que serão abordados na secção seguinte, são muito utilizados para pesquisa na robótica em áreas de alto nível como aprendizagem, navegação autónoma, entre outras. O CB^2 é um robô de investigação desenvolvido pelo *Socially-Synergistic Intelligence* (liderado pelo professor Hiroshi Ishiguro da Universidade de Osaka), um grupo do projecto *JST ERATO Asada*. O objectivo do projecto *Asada* é compreender o processo de desenvolvimento humano de funções cognitivas, através do *feedback* mútuo entre aproximações científicas e sintéticas, construindo um robô humanoide que possui um modelo de desenvolvimento computacional.[20] O CB^2 (Child-Robot with Biomimetic Body) foi concebido com o fim de estabelecer, e manter, uma interacção entre o ser humano e o robô. Além de reconhecer caras, expressões e gestos, nos últimos dois anos o CB^2 tem vindo a aprender, com ajuda de humanos, a andar e falar. O CB^2 está dotado de 51 actuadores pneumáticos, 5 motores eléctricos e está coberto por uma pele de silicone que contém 197 sensores tácteis. A característica mais relevante é a sua pele de silicone, pois para além dos múltiplos sensores que lhe permitem saber exactamente onde está a ser tocado, permite também, protecção às pessoas que interajam com ele. Ainda na área de

investigação do desenvolvimento cognitivo, o RobotCub, um consórcio de diversas universidades europeias, desenvolveu o iCub[32]. Este projecto teve inicio em 2004 com o objectivo de ajudar os investigadores a entenderem como os humanos aprendem e evoluem. Com um peso de 23 Kg e uma altura de 90 cm, o iCub tal como o CB^2 também reconhece faces, expressões e objectos, mas a sua maior vantagem é a multiplicidade de movimentos que lhe permite replicar o processo de aprendizagem de uma criança, embora os seus graus de liberdade (30 dof) estejam um pouco abaixo dos do CB^2, o iCub possui mãos com cinco dedos, ao contrário do CB^2 que possui mãos de apenas três dedos. E uma criança para aprender precisa, sobretudo, de tocar e manipular.

Quanto à natureza dos robôs móveis, estes podem ser classificados por duas áreas distintas: ambiente de operação e tipo de locomoção.

Relativamente à classificação por ambiente de operação, estes podem ser:

- Terrestres;

  Exteriores (figuras 1.1, 1.2a, 1.2d, 1.3a e 1.3b);

  Interiores (figuras 1.4a, 1.4b, 1.5a, 1.4c, 1.5b, 1.5c, 1.6a, 1.6b e 1.7a);

- Aéreos, também designados por UAV - *Unmanned Aerial Vehicles* (figuras 1.2b);

- Subaquáticos, também designados por AUV - *Autonomous Underwater Vehicles* (figura 1.2c);

Em relação ao tipo de locomoção, no caso dos robôs terrestres, estes podem ser movidos por:

- Rodas (figuras 1.1, 1.4a, 1.4b, 1.4c, 1.5c e 1.7a);

- Pernas (figuras 1.2a, 1.5a, 1.5b, 1.6a e 1.6b);

- Lagartas (figuras 1.2d, 1.3a e 1.3b);

1.3.2 Pequenos Robôs móveis para aprendizagem

Não sendo tão avançados, a nível de *hardware*, como os tipos de robôs da área da aprendizagem anteriormente mencionados, os pequenos robôs para aprendizagem serão sempre utilizados para investigação nesta área. Pois, não sendo estes robôs tão complexos como os outros, é mais fácil testar e validar algoritmos de aprendizagem, que depois poderão ser utilizados em robôs mais complexos. Além disso, é possível trabalhar com eles em cima de bancadas laboratoriais ou em espaços mais pequenos, devido às suas reduzidas dimensões.

Existem várias plataformas *standard* deste tipo de robôs prontas para programar que, normalmente, são adquiridas para investigação na área. Como por exemplo o Khepera (figura 1.7a), o Alice (figura 1.7b), o e-puck (figura 1.7c), o Miabot Pro (figura 1.7d), o AmigoBot™ (figura 1.8a) e o pioneer 3-DX (figura 1.8b). Os dois últimos, significativamente maiores que os primeiros, não podendo embora, ser utilizados em cima de bancadas, podem, ainda assim, ser utilizados em ambientes pequenos. Estas plataformas *standard* são muitas vezes modificadas, para que os robôs estejam mais adaptados e capacitados para o tipo de estudo em causa.

O Khepera é dos mais conhecidos desta categoria de robôs. Foi desenvolvido em 1992 no LAMI (*Laboratoire de Microinformatique*) da EPFL (*École Polytechnic Fédérale de Lausane*) e lançado em 1995. Entre 1995 e 1999, venderam-se cerca de 700 Khepera da primeira geração.[21]

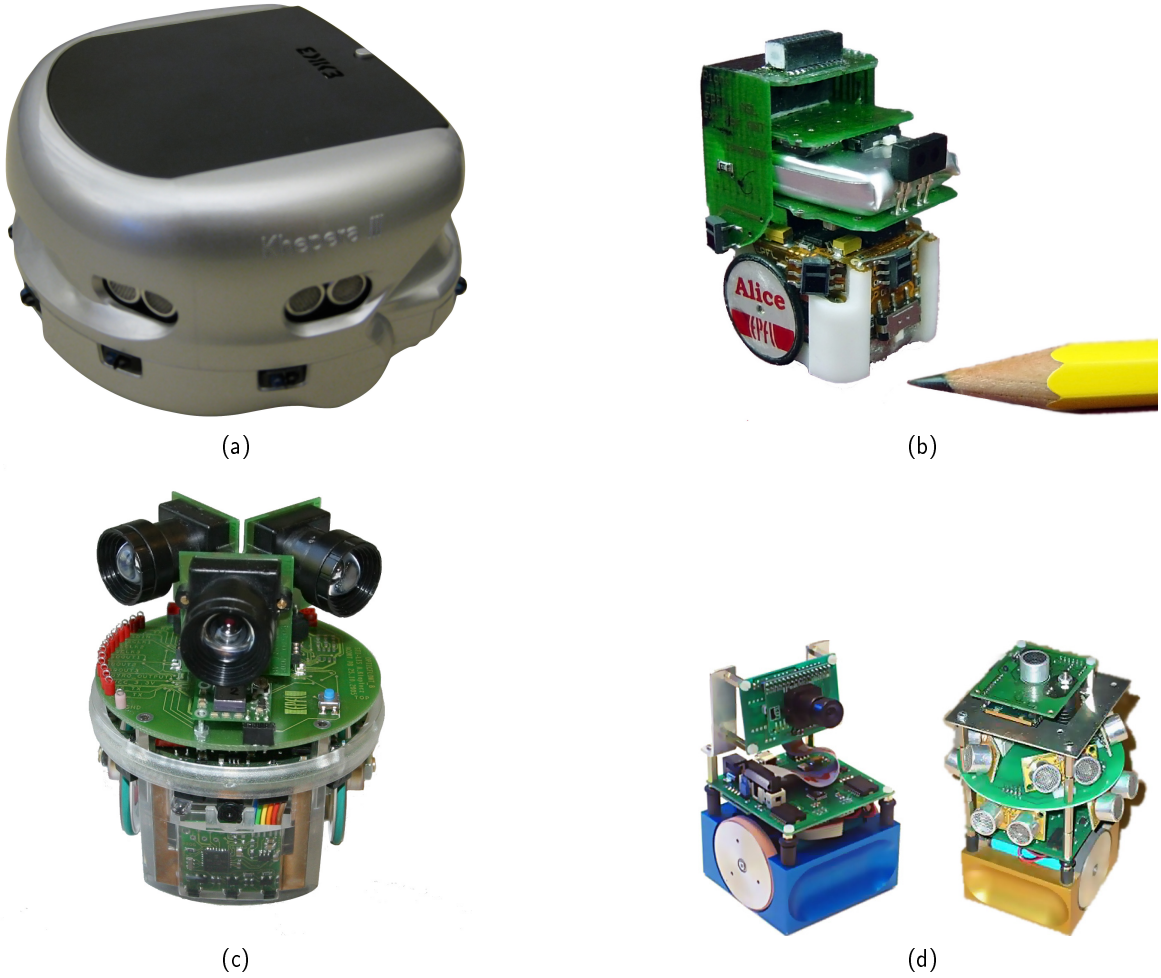


Figura 1.7: (a) Robô Khepera III da K-team Corporation; (b) Robô Alice desenvolvido na EPFL; (c) Robô e-puck também desenvolvido na EPFL; (d) Robô Miabot Pro da Merlin Systems Corp. Ltd.

Hoje já vai na terceira geração, com maior poder de processamento e mais sensores. O Khepera III (figura 1.7a) possui um microcontrolador dsPIC30F5011 da Microchip, com 4 KB de memória RAM (*Random Access Memory*) e 66 KB de memória *flash*, dois servo-motores, controlados por dois microcontroladores PIC18F4431 da Microchip com dois *encoders* incrementais acoplados ao veio dos servos. A nível de sensores, está equipado com onze sensores de infra-vermelhos, nove de proximidade e dois para detectar linhas no chão, e cinco sensores de ultra-sons. Mede à volta de 130 mm de diâmetro, 70 mm de altura e pesa 690 g. Além disso, podem-se acopular módulos de expansão Korebot ao Khepera, permitindo que este tenha maior capacidade de processamento, mais e variados tipos de sensores, maior capacidade de armazenamento de dados, entre outras características.

O Alice (figura 1.7b) é o robô mais pequeno deste tipo de robôs. Desenvolvido no ASL (*Autonomous Systems Lab*) da EPFL mede 22 mm de comprimento, 21 mm de largura e 20 mm de altura. Está equipado com um microcontrolador PIC16LF877 da Microchip, dois motores Swatch e quatro sensores infra-vermelhos de proximidade. Tem vários módulos de expansão, nomeadamente,

uma câmara digital linear de 102 pixels, sensores tácteis, ZigBee®, uma garra e um painel solar.

Criado em 2004 no ASL da EPFL, pelos mesmos criadores do Khepera, o e-puck (figura 1.7c) foi desenvolvido para poder operar em cima de uma bancada de laboratório, medindo apenas 70 mm de diâmetro por 50 mm de altura (sem módulos de expansão), em diversas áreas de desenvolvimento.[22] O e-puck está equipado com um microcontrolador dsPIC30F6014A da Microchip, com 8 KB de memória RAM e 144 KB de memória *flash*, dois motores passo-a-passo com uma resolução de 1000 passos por revolução, oito sensores de proximidade de infra-vermelhos, um acelerómetro de três eixos, três microfones, uma câmara digital CMOS a cores com uma resolução de 640 x 480 *pixels*, um altifalante conectado a um descodificador audio, e vários LEDs verdes e vermelhos para interface visual. Possui ainda um receptor de infra-vermelhos, uma interface RS232 e um dispositivo de interface Bluetooth® com o qual é possível comunicar com um PC ou com outros e-puck até um máximo de sete.

A estes dispositivos, podem-se juntar outros, através dos módulos de expansão, como por exemplo interface ZigBee®, um sistema de visão omnidireccional, um anel de LEDs, entre outros. Estes módulos de expansão, podem ser montados uns em cima dos outros, o que proporciona ao e-puck uma maior flexibilidade em relação às possíveis áreas de desenvolvimento.

O Miabot Pro (figura 1.7d) criado pela Merlin Systems Corp. Ltd é um robô com um formato cúbico, medindo 75 mm x 75 mm x 75 mm, está equipado com um microprocessador ATMega64 da Atmel, com 4 KB de memória RAM, 64 KB de memória *flash* e 2 KB de memória EEPROM (*Electrically-Erasable Programmable Read-Only Memory*), dois servo-motores com *encoders* e uma interface de comunicações por Bluetooth®.[18]

Embora tenha uma porta de expansão de oito pinos I/O (ou analógicos de 10 *bits*), permitindo que seja possível a introdução de sensores para aprendizagem, o Miabot é muito inferior ao nível de *hardware* em relação aos outros robôs. No entanto, o Miabot possui um algoritmo de um controlador PID *onboard*, possibilitando movimentos rápidos e livres de derrapagens ou deslizamentos, e uma função de auto-programação, o que é uma vantagem em relação aos outros robôs.



(a) (b)

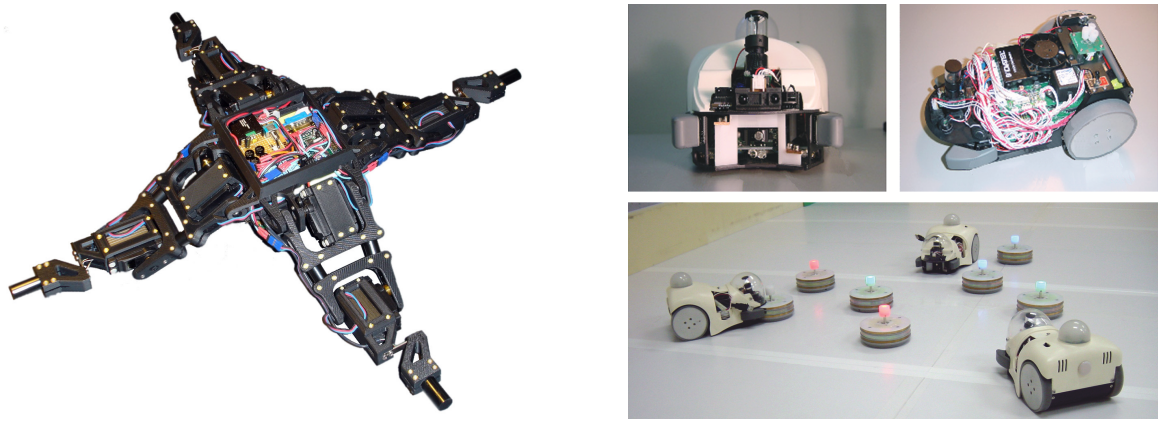
Figura 1.8: (a) Robô AmigoBot™ da Mobile Robots$_{INC}$; (b) Robôs Pioneer 3-AT e 3-DX da Mobile Robots$_{INC}$.

A Mobile Robots$_{INC}$ é uma empresa especializada em desenvolver e comercializar robôs móveis autónomos, muitos dos quais para pesquisa e desenvolvimento para Universidades. Desde que lançou o primeiro robô Pioneer, em 1995, que a Mobile Robots tem vindo a crescer no ramo da robótica, acumulando cada vez mais conhecimentos na área de navegação autónoma de robôs.[3] Isto implica que os seus robôs estejam equipados com bons materiais e *hardware* apropriado para a navegação autónoma. Os robôs mais pequenos que esta empresa disponibiliza são o AmigoBot™

e o Pioneer 3-DX. Como já foi referido anteriormente, estes robôs são significativamente de maiores dimensões, comparativamente aos restantes já descritos, no entanto não deixam de ser uma referência para esta área de pesquisa já que possuem dimensões suficientes para poderem operar em espaços como o de um laboratório de pesquisa robótica.

O AmigoBot™ (figura 1.8a) está equipado com um microprocessador de 32 *bits* SH7144 da Renesas, com 16 KB de memória RAM interna, 1 MB de memória RAM externa e 64 KB de memória *flash*, dois servo-motores (com *encoders* com uma resolução de $5 \times 10^{-5}\,^{\circ}$), um altifalante, oito sensores de ultra-sons (seis para a frente e dois para trás) e duas portas de comunicação série. Mede 330 mm de comprimento, 280 mm de largura e 150 mm de altura. Este robô apresenta uma fraca expansibilidade pois, embora o microcontrolador possua um barramento de oito pinos I/O e um pino analógico de 10 *bits*, só tem capacidade para receber dados de mais oito dispositivos, uma vez que já estão integrados os oito sensores de ultra-sons. A grande desvantagem deste robô, para além do seu tamanho e da sua pouca expansibilidade, é o facto de ter sido concebido para ser comandado à distância, sendo necessário conectá-lo a um computador externo pela porta série (por cabo ou *wireless*) para o tornar autónomo, impossibilitando-o de ser completamente autónomo.

Um pouco maior ainda, o Pioneer 3-DX (figura 1.8b), que mede 440 mm de comprimento, 380 mm de largura e 220 mm de altura, possui um microprocessador SH2 RISC (*Reduced Instruction Set Computer*) de 32 *bits* da Renesas, com 32 KB de memória RAM interna, 1 MB de memória externa e 128 KB de memória *flash*, dois servo-motores com *encoders* com uma resolução de $5 \times 10^{-5}\,^{\circ}$, dois conjuntos de sensores de ultra-sons compostos por oito sensores cada um, um orientado para a frente do robô e outro, opcional, orientado para trás do robô, que lhe confere uma percepção de $360\,^{\circ}$. Possui ainda a possibilidade de integrar um computador, o que torna o Pioneer 3-DX completamente autónomo. A maior vantagem deste robô é a sua capacidade de carga, até 25 kg, podendo ser equipado com sensores mais sofisticados, como por exemplo um sensor de laser industrial ou câmaras de video digitais de maior resolução.



(a) (b)

Figura 1.9: (a) Robô *Starfish* da Universidade de Cornell; (b) Robôs Cyber Rodent da ATR *International*.

No entanto, existem alguns robôs que são concebidos de raiz para certos estudos. Como é o caso do Cyber Rodents (figura 1.9b) concebido para o *Cyber Rodent Project* da ATR (*Advanced Telecommunications Research Institute International*), e do robô *Starfish* (figura 1.9a) do CCSL (*Cornell Computational Synthesis Laboratory*) da Universidade de Cornell. Estes robôs, embora não sejam muito flexíveis, relativamente às áreas de desenvolvimento, possuem as características mais indicadas para a área, para a qual foram desenvolvidos. O que proporciona aos investigadores, um desenvolvimento mais profundo dessa área.

Como já foi dito anteriormente, o Cyber Rodent foi desenvolvido no âmbito do *Cyber Rodent Project* da ATR dirigido por Mitsuo Kawato. O principal objectivo deste projecto é compreender os mecanismos adaptativos necessários para agentes artificiais, que contêm as mesmas restrições que agentes biológicos: a auto-preservação e a auto-reprodução. Para isso, houve a necessidade de desenvolver uma colónia de robôs que tivesse a capacidade para procurar uma fonte de geração de energia, para recarregarem as suas baterias, e um meio de comunicação *wireless* que possibilite a cópia de programas entre robôs.[11]

O Cyber Rodent é um robô, movido por duas rodas, com um comprimento de 250 mm e está equipado com um processador SH-4 da Hitashi com 32 MB de memória, uma câmara digital CCD omnidireccional, um processador gráfico FPGA (*Field-Programmable Gate Array*) para captura de video e processamento de imagem a 30 Hz, um sensor de ultra-sons, sete sensores de infra-vermelhos (cinco orientados para a frente, um para a retaguarda e outro orientado para o chão), um acelerómetro de três eixos, um giroscópio de dois eixos, três LEDs para interface visual (verde, vermelho e azul), um altifalante e dois microfones para comunicação acústica, uma porta de comunicações por infra-vermelhos, para a comunicação entre dois robôs próximos, uma interface *wireless LAN* e uma porta USB para comunicar com um computador.[12]

No âmbito da pesquisa na área de auto-modelação robótica na Universidade de Cornell, um grupo de investigadores necessitou de um robô para validar algoritmos que ainda só tinham sido testados computacionalmente. A pesquisa consiste no desenvolvimento de algoritmos que permitem a um robô criar um modelo da sua própria topologia, através da sua interacção com o meio que o rodeia, para depois aprender a movimentar-se nesse meio, através da exploração de comportamentos de movimento do seu próprio modelo.[15]

O robô *Starfish* é um robô quadrúpede, que mede 140 mm de largura por 140 mm de comprimento e 85 mm de altura (na sua posição normal), equipado com um computador PC-104 *onboard* com um módulo A/D DMM-32X-AT, com trinta e duas entradas de 16 *bits*, e um módulo de controlo de servos SV-203B, comandado por porta série. O robô consiste num corpo rectangular, onde está posicionado o computador, e quatro pernas articuladas por dois servos Airtronics 94359 que lhe conferem oito graus de liberdade. Além disso, o robô está também equipado com uma variedade de sensores, entre os quais: um acelerómetro *onboard*, extensómetros colocados na parte distal das pernas e um sensor de distância óptico na parte inferior do robô.

Não existe uma definição concreta do que é um pequeno robô móvel para aprendizagem, no entanto, existem algumas características necessárias para este tipo de robôs. Nomeadamente: o robô deve possuir vários sensores que lhe forneçam dados relativos ao ambiente que o rodeia, pelo menos um sistema de locomoção e um sistema de alimentação do circuito electrónico; um controlador (microcontrolador ou computador) para processar os dados recebidos dos sensores e comandar o robô; e memória, quer volátil (como por exemplo RAM) para o armazenamento de dados durante o processamento da aprendizagem, e não volátil (como por exemplo EEPROM ou *flash*) para guardar os dados da aprendizagem.

1.3.3 *Reinforcement learning*

O *Reinforcement Learning* estuda o modo como os sistemas naturais e artificiais aprendem a prever as consequências das suas acções e optimizar os seus comportamentos em ambientes, cujas acções transportam esses sistemas de uma situação para outra distinta, levando a recompensas ou castigos.[8]

Genericamente, no *Reinforcement Learning* um agente observa, repetidamente, o estado do ambiente que o rodeia, e, depois, escolhe uma acção para executar através de uma política de aprendizagem. Uma vez executada a acção, o estado do ambiente muda e o agente recebe uma recompensa pela acção executada. Estas recompensas podem ser positivas, boas recompensas, ou negativas, castigos. Assim, o agente aprende a escolher acções por forma a maximizar a soma das recompensas que irá receber no futuro.[7]

O *Reinforcement Learning* é, de todas as abordagens de *Machine Learning*, a que mais se foca na aprendizagem por objectivos, a partir da interacção com o meio. Ao invés de ser dito a um agente qual a acção que deve executar, tal como na maioria dos vários métodos de *Machine Learning*, é o próprio agente que deve "descobrir" quais as acções, que em determinados estados, garantem maiores recompensas, executando-as e explorando o meio que o rodeia. As acções, não só afectam as recompensas imediatas, como também os próximos estados e, consequentemente, as recompensas seguintes. De facto, as características mais evidentes que distinguem o *Reinforcement Learning* de outras formas de aprendizagem são a tentativa-erro e a recompensa retardada.[30]

Um dos maiores desafios que aparecem no *Reinforcement Learning*, que não existe noutros tipos de aprendizagem, é a decisão entre explorar novas acções (*exploration*) e promover acções que já realizou no passado (*exploitation*). De facto, um agente, para maximizar as recompensas, deve executar acções que executou no passado e que descobriu serem as que maior recompensas dão. Por outro lado, para descobrir tais acções o agente tem de executar acções que ainda não tenham sido escolhidas. O agente tem de promover o que já conhece, por forma a obter a melhor recompensa, mas também tem de explorar, para poder escolher melhores acções no futuro.[30]

Existem dois tipos distintos de métodos de *Reinforcement Learning*: os baseados em modelos (*model-based*) e os livres de modelos (*model-free*). O método *model-based* usa a experiência para construir um modelo interno das transições entre os estados e as consequências imediatas. Depois, é escolhida uma acção, através da procura ou planeamento, a partir desse modelo interno. Por outro lado, o método *model-free* usa a experiência para aprender directamente politicas ou valores de estado-acção, sem recorrer a estimativas ou modelos de ambientes.[8]

No anexo A estão descritos alguns métodos de *Reinforcement Learning* mais conhecidos.

Capítulo 2

Projecto

"O único homem que nunca comete erros é aquele que nunca faz coisa alguma. Não tenha medo de errar, pois aprenderá a não cometer duas vezes o mesmo erro."

Theodore Roosevelt

No capítulo anterior, quando se aborda os pequenos robôs para aprendizagem, nota-se que em nenhum dos robôs descritos existe um dispositivo de pára-choques, à excepção do pioneer 3-DX que possui um módulo de expansão. Para um robô aprender um comportamento de locomoção necessita de um *feedback* do ambiente, que pode ser dado por diversos sensores. Um dispositivo de pára-choques não é imperativo para um robô poder aprender. No entanto, um sensor de contacto físico, como é um sensor passivo, tem uma fiabilidade, normalmente, superior à de qualquer tipo de sensor activo. Isto porque, um sensor de contacto é menos susceptível a factores externos que possam interferir com a sua mecânica, ao contrário da maioria dos sensores activos, onde, por vezes, a temperatura, campos electromagnéticos ou a humidade, podem interferir com a mecânica de percepção desses sensores.

Ainda assim, os sensores de contacto fornecem apenas a informação de um robô ter ou não embatido, pelo que são necessários outro tipo de sensores que possibilitem ao robô ter uma percepção do ambiente que o rodeia, nomeadamente sensores de distância, como infra-vermelhos ou ultra-sons, que permitem ao mesmo saber a distância a que se encontram obstáculos.

Numa primeira abordagem deste trabalho, a ideia era melhorar o equipamento de um robô já existente, para poder aprender. No entanto, os robôs existentes no mercado ou são muito dispendiosos, ou são muito limitados no que diz respeito à expansão de *hardware*. Por exemplo, cada Khepera III custa cerca de 2500 €, cada Miabot 950 €, e mesmo o e-puck (figura 1.7c) custa 600 €. Estes preços incluem apenas o equipamento base dos robôs. Existem robôs mais baratos, como o 3pi da Polulu que custa cerca de 86 €, porém, além da fraca expansibilidade que este robô apresenta, seria necessário introduzir mais sensores e uma nova estrutura para suportar esses sensores, o que implicaria mais gastos.

Então, optou-se por conceber um robô novo de raiz, de baixo custo, usando, como inspiração, um projecto *open source* de um robô concebido para competições de mini-sumo, o Mark III[2].

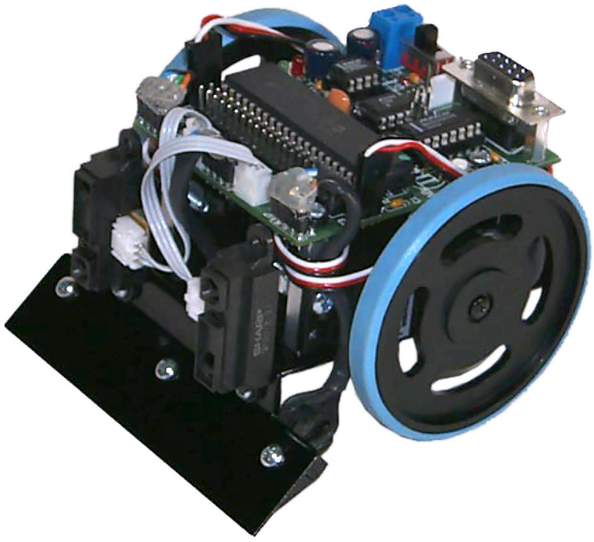


Figura 2.1: Robô MarkIII.

O Mark III (figura 2.1) é um robô concebido para lutas entre robôs mini-sumo, em que o objectivo principal é colocar o robô adversário fora da área de luta. Uma vez que se trata de lutas entre robôs, este apresenta um chassi robusto, um dos factores importantes para um robô de aprendizagem por reforço. Este robô também possui hardware electrónico expansível, factor igualmente importante para a escolha deste robô como base para o desenvolvimento do robô de aprendizagem.

O Mark III possui um microcontrolador PIC16F877 da Microchip, pré-programado, com 368 bytes de memória RAM, 14 KB de memória *flash* e 256 bytes EEPROM, dois servos, dois sensores de proximidade de infra-vermelhos e três sensores infra-vermelhos para a detecção de linhas no chão.

Concebeu-se assim, um robô com um chassi robusto, um sistema de pára-choques, que contém dez sensores de contacto, robusto e flexível, quatro sensores infra-vermelhos de distância, um sensor ultra-sons, dois servos e um módulo de comunicação *wireless*. Ao robô concebido foi dado o nome SALbot (*Small Autonomous Learning Robot*). Foram fabricados e montados dois SALbot para testar e comparar algoritmos de aprendizagem. Um pouco maior que o Khepera III, o SALbot mede 148 mm de comprimento por 172 mm de largura e 102 mm de altura, o que possibilita a operação do robô em bancadas de laboratório.

O SALbot é um robô do tipo diferencial de duas rodas com dois pontos adicionais de contacto. Isto é, o SALbot possui quatro rodas, sendo duas, accionadas pelos servos S3003 da Futaba® no centro do chassi do robô, e duas rodas de apoio que correspondem a dois apoios esféricos livres. O SALbot possui também um micro controlador PIC18F4620 da Microchip, com 3968 bytes de memória RAM, 64 KB de memória *flash* e 1024 bytes EEPROM.

2.1 Projecto Mecânico

2.1.1 Chassi

Como já foi referido, foram concebidos um chassi e um sistema de pára-choques suficientemente robustos para que a aprendizagem possa ser realizada eficientemente. A base do chassi foi

fabricada a partir de uma chapa de 1 mm de espessura de alumínio AW1050-H111, que, embora não possua uma elevada resistência mecânica, apresenta uma boa conformabilidade, sendo que essa característica foi tida em conta devido ao facto de ser necessário conformar a chapa, para esta adquirir as formas do chassi. Para colmatar a fraca resistência mecânica do alumínio, foi concebida uma peça em acrílico que, para além de aumentar a resistência mecânica do chassi, também aloja os sensores de contacto que fazem parte do sistema de pára-choques. O chassi é composto também por quatro pilares que suportam os dois servo motores e a placa de circuito impresso; uma peça onde encaixa o suporte para os sensores de ultra-sons e infra-vermelhos de distância frontais; dois pilares que suportam, cada um, um sensor de distância infra-vermelho; e dois apoios esféricos livres.

Como é possível ver na figura 2.5, a base do chassi tem uma área plana grande no centro, onde são alojados os servos. Nas extremidades laterais estão dois "braços" onde são colocadas as peças de acrílico, e alojados os sensores de contacto do sistema de pára-choques. Na parte de trás, a chapa é dobrada em forma de "U", para dar mais resistência mecânica ao chassi e para a colocação de um sensor de distância de infra-vermelhos na traseira. Nas extremidades frontal e traseira a chapa é dobrada para alojar dois apoios esféricos livres.

2.1.2 Dimensionamento dos apoios do chassi

Para saber quantos apoios seriam necessários no SALbot, determinou-se o centro de massa e calculou-se o momento que o peso do robô exerce no eixo de rotação dos servos, comparando-o com o binário máximo dos mesmos. Devido à complexa distribuição de massas dos diversos componentes do SALbot, o centro de massa foi determinado a partir do modelo 3D, que, como é possível verificar na figura 2.2, se encontra na parte posterior em relação ao eixo de rotação das rodas, o que implica a necessidade de um apoio na parte frontal do chassi. Das características do servo da Futaba® S3003 retiraram-se os seguintes valores:

$$\textbf{Velocidade:} \begin{cases} 0.23\ \text{s}/60^\circ & @ \quad 4.8\ \text{V}; \\ 0.19\ \text{s}/60^\circ & @ \quad 6\ \text{V}. \end{cases}$$

$$\textbf{Binário:} \begin{cases} 3.2\ \text{kgf}\cdot\text{cm} & @ \quad 4.8\ \text{V}; \\ 4.1\ \text{kgf}\cdot\text{cm} & @ \quad 6\ \text{V}. \end{cases}$$

Para verificar se o momento criado pelo peso do robô é suficiente para anular o binário dos servos, foi necessário calcular-se o binário dos dois servos e compará-lo ao momento criado pelo peso do próprio robô. Uma vez que os servos operam a 5V, usou-se interpolação linear para calculo:

$$\begin{aligned} \vec{M}_1 &= 2 \times \left(\frac{6-5}{6-4.8} \times 3.2 + \frac{5-4.8}{6-4.8} \times 4.1 \right) \\ &\cong 6.7 \quad \text{kgf}\cdot\text{cm} \end{aligned}$$

e como,

$$\begin{aligned} 1 \quad \text{kgf}\cdot\text{cm} &= \frac{1}{100} \times 9.8 \quad \text{N}\cdot\text{m} \\ &= 9.8 \times 10^{-2} \quad \text{N}\cdot\text{m} \end{aligned}$$

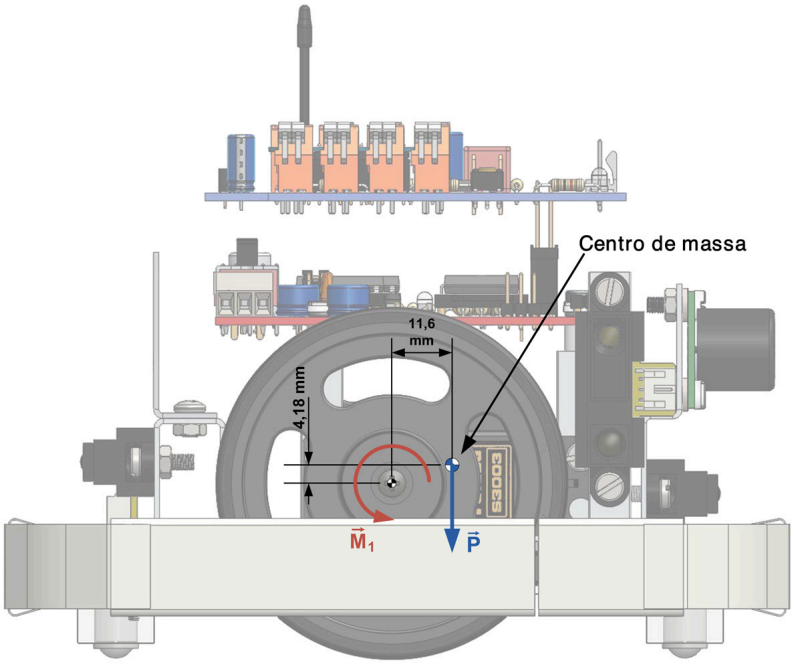


Figura 2.2: Diagrama de corpo livre para o cálculo do somatório dos momentos.

então,

$$\begin{aligned} \vec{M}_1 &= 6.7 \times 9.8 \times 10^{-2} \\ &= 65.6 \times 10^{-2} \quad \text{N} \cdot \text{m} \end{aligned}$$

A massa do robô foi obtido a partir do modelo 3D com uma majoração de 50%, e calculou-se o momento criado no eixo de rotação dos servos:

$$\begin{aligned} \vec{P} &= m \times \vec{g} \\ \vec{P} &= 0.462 \times 9.8 \\ &= 4.5 \text{ N} ; \\ \vec{M}_P &= \vec{P} \times \text{d} \\ &= 4.5 \times 11.6 \times 10^{-3} \\ &= 5.2 \times 10^{-2} \text{N} \cdot \text{m} \end{aligned}$$

Como $\vec{M}_1 > \vec{M}_P$, é necessário então um apoio na parte traseira do robô para que este não levante no arranque para trás.

Os apoios esféricos livres são compostos por uma esfera de aço de ∅ 7 mm, uma mola e uma estrutura cilíndrica com ∅ 8 mm. A mola pressiona a esfera contra a estrutura, para que esta apenas ceda a uma força que vença a força da mola, sem, no entanto, impedir que esta rode livremente. Cada apoio esférico é fixado, por aperto veio-furo, a uma peça de acrílico que por sua vez é fixa, através de uma ligação colada, à base do chassi.

2.1.3 Suportes

Tanto os pilares que suportam os servos, como os que suportam os sensores infra-vermelhos de distância, foram fabricados a partir de uma vara de perfil quadrado de 8*x*8 mm de alumínio corrente, para garantir uma fixação rígida entre os servos e o chassi, e também, para suportar as placas de circuito impresso. Os pilares, como se pode ver na figura 2.5, são fixados à base do chassi por uma ligação aparafusada; cada pilar possui um furo roscado no qual, um parafuso M3 garante a fixação do pilar à base do chassi. Nos pilares dos sensores existem, também, mais dois furos roscados que permitem a fixação do sensor com dois parafusos M3. Enquanto que, nos pilares que suportam os servos, além de um furo passante para introduzir um parafuso para a fixação do servo, possui, também, outro furo roscado para a fixação da placa de circuito impresso. Cada pilar que suporta um sensor de distância infra-vermelhos pode ser fixado por forma a colocar o sensor numa determinada orientação desejada. O que permite várias configurações diferentes do SALbot que podem ser testadas, em vários tipos de aprendizagem ou vários tipos de ambiente, com o fim de descobrir a orientação que mais se adequa ao tipo de aprendizagem ou ambiente.

A peça onde encaixa o suporte dos sensores de ultra-sons e infra-vermelhos de distância frontais, além de permitir a fixação do suporte dos sensores frontais, garantida por dois furos roscados onde apertam dois parafusos M3, permite também a fixação dos servos aos pilares frontais do chassi, através de dois furos roscados onde apertam dois parafusos M3, sendo que, cada parafuso aperta o servo ao pilar. Esta peça foi concebida em acrílico, uma vez que é leve e suficientemente resistente para a sua função.

Existe ainda um suporte traseiro para fixar um sensor de distância por ultra-sons; prescindiu-se no entanto da implementação desse sensor, pelo que, posteriormente, esse suporte passou a suportar a pilha que alimenta o circuito electrónico das placas.

2.1.4 Sistema de pára-choques

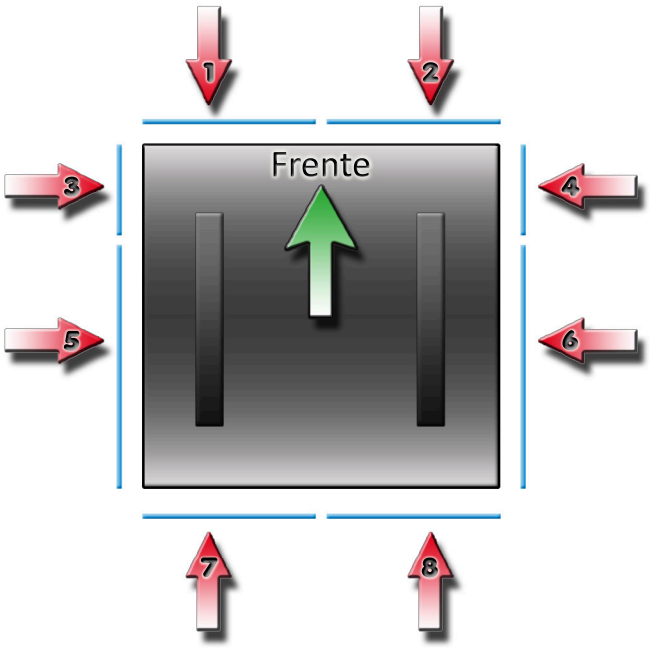
O sistema de pára-choques foi concebido tendo em conta a necessidade de robustez e flexibilidade, mas também de forma a permitir alguma distinção em relação ao local de embate. A ideia foi distinguir oito zonas distintas de contacto, nomeadamente frente-esquerda, frente-direita, lateral-esquerda-frente, lateral-direita-frente, lateral-esquerda, lateral-direita, trás-esquerda e trás-direita do robô, como se pode ver na figura 2.3.

Para isso, projectou-se o SALbot com dez sensores de contacto, um para cada zona de contacto, excepto nas zonas laterais esquerda e direita, que contêm dois em paralelo, para uma maior estabilidade. Esses sensores de contacto, são fins-de-curso com uma haste que contém um rolete na ponta para poderem deslizar na face de contacto. Além desses sensores, o sistema de pára-choques é constituído por barras de contacto, que fazem a transferência de um impacto exterior com os sensores de contacto e representam cada uma das zonas de impacto, chapas arredondadas para os cantos do robô, que permitem a transferência de um impacto exterior nos cantos para os sensores de contacto para uma ou duas zonas de impacto vizinhas desse canto.

As barras de contacto foram concebidas em Teflon® por forma a permitirem um deslizamento quer das rodas dos sensores de contacto, quer da chapa de alumínio dos cantos, com pouco atrito. As barras possuem um perfil em forma de um duplo "T", para que a chapa de alumínio possa deslizar em duas direcções, longitudinal e transversalmente, com alguma estabilidade, principalmente quando desliza transversalmente ao perfil. Cada barra de contacto apoia na extremidade da haste de dois sensores de contacto, excepto as barras lateral-esquerda-frente e lateral-direita-frente, que apoiam apenas na extremidade da haste de um único sensor. As barras são fixadas, por um adesivo,

a uma tela fina de latex na face de contacto, que por sua vez é fixada ao chassi do robô através de parafusos e porcas M2 e M3. Esses parafusos e porcas apertam numas tiras de PVC transparente que fixam uniformemente as telas de latex ao chassi.

Por serem flexíveis, as telas de latex permitem que as barras de contacto se movam quando existe uma colisão, mas também garantem a posição das barras de contacto quando estas não são solicitadas.



1 - Frente-esquerda;

2 - Frente-direita;

3 - Lateral-esquerda-frente;

4 - Lateral-direita-frente;

5 - Lateral-esquerda;

6 - Lateral-direita;

7 - Trás-esquerda;

8 - Trás-direita

Figura 2.3: Zonas de impacto.

Para que um robô "saiba" quando bateu necessita de um dispositivo de detecção que distinga um embate em qualquer direcção; por isso concebeu-se o sistema de pára-choques para que quando exista um contacto nos cantos, este, seja distinguido independentemente da direcção do contacto. Isso foi conseguido com um sistema que consiste numa chapa dobrada a 90°, que forma o canto, que desliza entre as barras de contacto, que estão próximas do canto, e as rodas das hates dos sensores de contacto. Na figura 2.4 é possível ver as várias situações de embate que podem ocorrer nos cantos do sistema de pára-choques. Quando o choque se dá na zona das barras de contacto, quer a barra de contacto, onde se dá o contacto, quer o canto são deslocados e o sensor de contacto (fim de curso) é activado. Se o choque se der exclusivamente na zona do canto, apenas este é deslocado, activando um ou ambos os sensores de contacto, dependendo da direcção do embate. No caso de existir um duplo contacto, um na zona de uma das barras de contacto e outro na zona do canto, a barra de contacto e o canto são deslocados na direcção do embate na zona da barra de contacto, activando um dos sensores de contacto, e o canto é, também, deslocado na direcção do segundo contacto, activando o outro sensor de contacto. Além disso, nas partes laterais a distância entre as barras de contacto, laterais-frontais e laterais, é de 1 mm, o que permite a detecção de uma colisão em qualquer zona lateral. Assim, o sistema de pára-choques do SALbot permite a detecção de um embate qualquer que seja a direcção do mesmo.

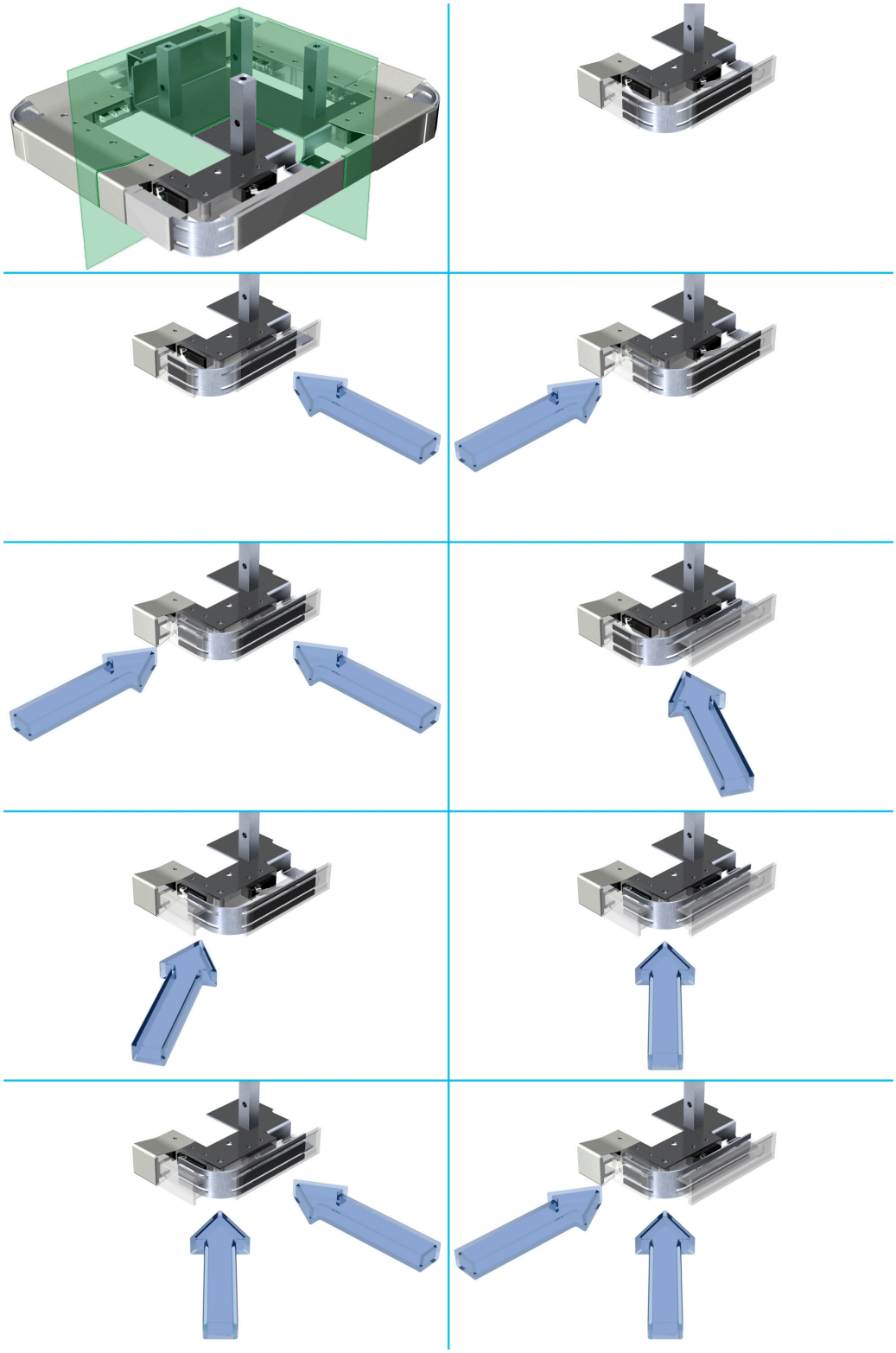


Figura 2.4: Possibilidades de detecção de colisão pelo sistema de pára-choques nos cantos.

Os cantos foram concebidos em chapa de 1 mm de espessura de alumínio AW1050-H111, dobrada com um raio médio de 16.5 mm e com dois rasgos de cada extremidade, por forma a que encaixem nas barras de contacto onde deslizam. Os cantos são fixados, por um adesivo, a uma tira de latex que é também fixada na extremidade de duas barras que formam um canto, também por um adesivo. Esta tira de latex vai garantir quer o retorno correcto do canto ao estado inicial após um choque, quer a correcta movimentação quando existe um choque, garantindo o paralelismo em relação às barras des contacto.

2.1.5 Sistema de locomoção

Em cada servo está fixada uma roda, de ∅ 65 mm, por um parafuso. Cada roda tem um elástico de borracha à volta, o que permite uma melhor aderência ao piso. Uma vez que os servos S3003 da Futaba® apenas têm um amplitude de rotação de 90° para cada lado, foi necessário a modificação dos mesmos para poderem rodar continuamente. Para tal, houve a necessidade de abrir cada servo, cortar o batente da roda dentada, retirar o potenciómetro, que limita a amplitude de rotação do servo, e colocar, no seu lugar, duas resistências de 2.2 $k\Omega$ em paralelo, o que faz com que o controlador do servo "pense" que o servo se encontra sempre a meio da sua amplitude de rotação. Assim, para que o servo rode numa determinada direcção à velocidade máxima, basta enviar o comando para uma das posições extremas. Para o parar basta enviar o comando para o centro da amplitude de rotação. O controlo dos servos é descrito mais detalhadamente na secção 3.3.1. Como já foi referido, cada servo é fixado aos pilares do chassi que os suportam. Esta fixação é garantida por dois parafusos M3, na parte frontal o parafuso é apertado à peça de acrílico onde é também fixado o suporte dos sensores frontais, na parte traseira o parafuso é apertado a uma porca M3.

2.1.6 Montagem do Conjunto

Na figura 2.6 é possível ver a perspectiva explodida do SALbot e na figura 2.5 a perspectiva explodida do chassi. A maior parte da montagem dos componentes é feita com o recurso a parafusos. No entanto, os suportes dos apoios esféricos são fixados com uma fita adesiva de dupla face e as baterias para alimentação dos servos e das placas de circuito impresso são fixadas com tiras adesivas de velcro, para poderem ser retiradas e substituídas facilmente para o recarregamento, por exemplo.

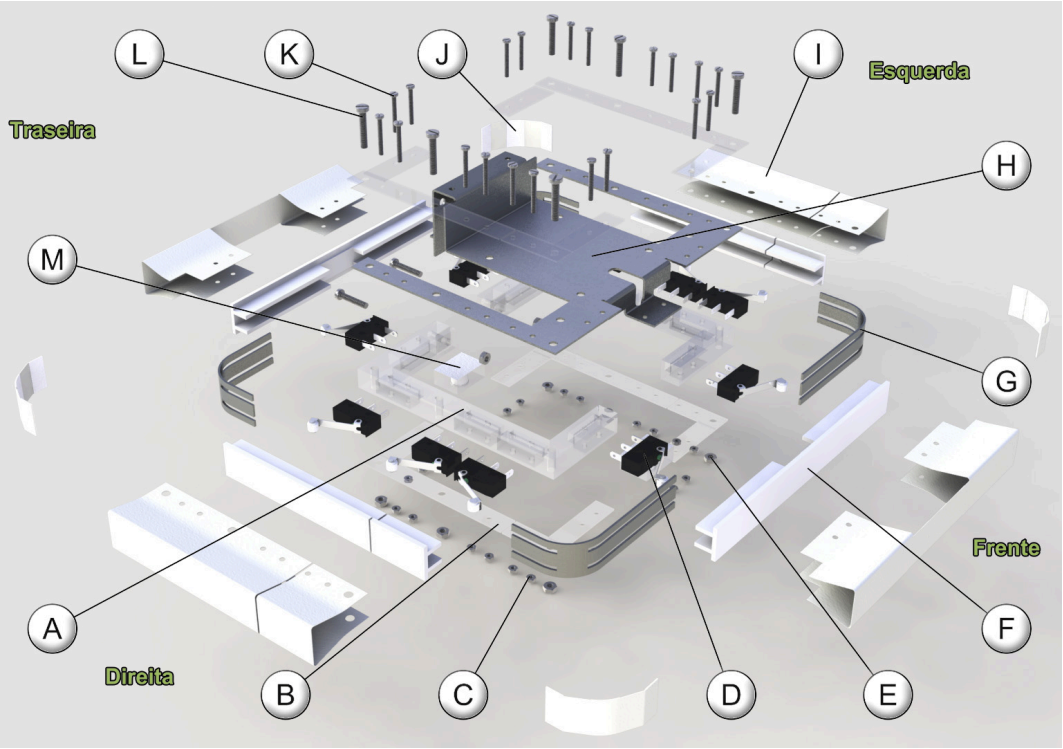
A sequência de montagem para o chassi é a seguinte:

1. Fixar as peças que suportam os sensores de contacto (A) à base do chassi (H), através de dois parafusos (L) e duas porcas M3 (E) na parte traseira;

2. Fixar os apoios esféricos nos suportes respectivos (M) e fixá-los na base do chassi (H);

3. Colocar cada sensor de contacto (D) em cada caixa, das duas peças que suportam os sensores (A), para o efeito;

4. Colar cada tela de latex (I) à barra de contacto (F) correspondente;



A- Suporte dos sensores de contacto; B- Tiras de PVC; C- Porcas M2; D- Sensores de contacto; E- Porcas M3; F- Barras de contacto; G- Cantos do pára-choques; H- Base do chassi; I- Telas de Latex; J- Telas de Latex para os cantos; K- Parafusos M2; L- Parafusos M3; M- Apoios esféricos com suporte.

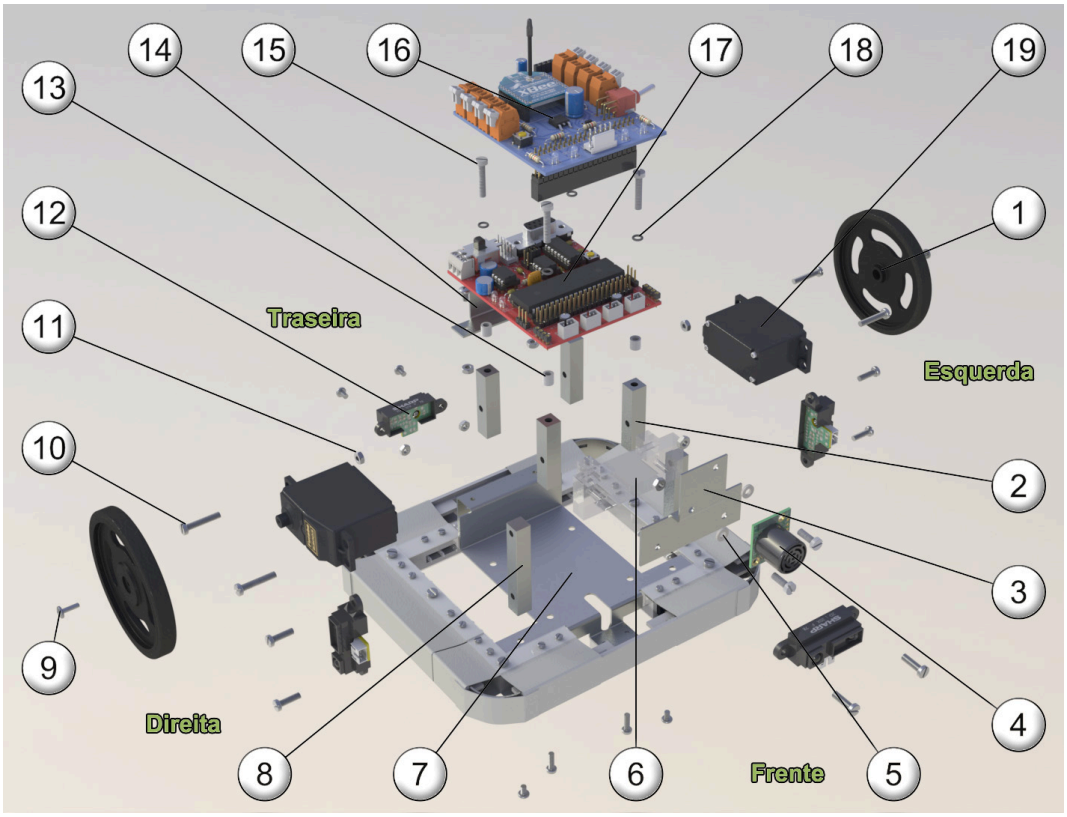
Figura 2.5: Perspectiva explodida do chassi do SALbot.

5. Orientar os furos de cada tela de latex (I), da parte superior, com os furos dos "braços" da base do chassi (H), colocar as respectivas tiras de PVC (B) orientando-as pelos furos e aplicar os parafusos M3 (L) e M2 (K);

6. Orientar os furos de uma tela de latex (I), da parte inferior, de cada lado do robô (no caso da parte da frente e de trás apenas um dos lados da tela) com os furos dos suportes dos sensores de contacto (A), colocar as respectivas tiras de PVC (B) orientando-as pelos furos e aplicar as porcas M3 (E) e M2 (C);

7. Encaixar os cantos (G) nas barras de contacto (F) já fixadas;

8. Orientar os furos das restantes telas de latex (I), da parte inferior, com os furos dos suportes dos sensores de contacto (A), encaixando as respectivas barras de contacto (F) nos cantos (G), colocar as respectivas tiras de PVC (B) orientando-as pelos furos e aplicar as porcas M3 (E) e M2 (C).



1- Rodas ; 2- Pilares de suporte dos servos ; 3- Suporte dos sensores de distância frontais ; 4- Sensor de distância por ultra-sons ; 5- Espaçador de Nylon ; 6- Peça de fixação do suporte dos sensores de distância frontais ao chassi; 7- Chassi; 8- Pilares de suporte dos sensores de distância por infra-vermelhos laterais; 9- Parafusos de fixação das rodas aos servos; 10- Parafusos M3; 11- Porcas M3; 12- Sensores de distância por infra-vermelhos; 13- Espaçador de plástico; 14- Suporte para sensor de distância por ultra-sons traseiro; 15- Parafusos M3 de Nylon; 16- PCB auxiliar; 17- PCB principal; 18- *O-rings*; 19- Servos.

Figura 2.6: Perspectiva explodida do SALbot.

Depois da montagem do chassi o robô é montado da seguinte forma:

1. Fixar cada servo (19) aos pilares (2) traseiros com um parafuso (10) e porca (11) M3, no furo superior que se encontra mais próximo do centro de rotação do servo;

2. Fixar cada servo (19) aos pilares (2) frontais com um parafuso (10) apertando-o à peça de fixação do suporte dos sensores de distância frontais (6);

3. Fixar as rodas (1) nos servos (19) com os parafusos (9);

4. Ligar os fios aos sensores de contacto que estão no chassi, que posteriormente serão ligados à placa auxiliar (16);

5. Fixar os pilares (2), já fixados aos servos (19) e à peça de fixação do suporte dos servos frontais (6), à base do chassi (7) com parafusos M3 (10);

6. Fixar os sensores de infra-vermelhos (12) nos pilares (8) respectivos para o efeito, com parafusos M3 (10), e o sensor de ultra-sons frontal (4) na chapa de suporte dos sensores frontais (3), com dois parafusos M3 de nylon (15), duas porcas M3 (11) e com um espaçador de nylon (5) entre o sensor (4) e a chapa de suporte (3);

7. Fixar o sensor de infra-vermelhos (12) traseiro na parte de trás do chassi com dois parafusos (10) e porcas M3 (11), fixar os pilares (8) de suporte dos sensores de infra-vermelhos à base do chassi (7), com a orientação desejada, e fixar o sensor de infra-vermelhos (12) frontal à chapa de suporte dos sensores frontais (3), fixando, esta, à peça de fixação do suporte dos sensores frontais (6);

8. Fixar a chapa de suporte traseira (14) com dois parafusos (10) e porcas M3 (11) ao chassi (7);

9. Fixar a bateria de alimentação dos servos por baixo da base do chassi (11) e a bateria de alimentação das placas de circuito impresso na chapa de suporte traseira (14), utilizando fitas adesivas de velcro;

10. Fixar a placa de circuito impressa principal (17) aos pilares (2) com parafusos M3 de nylon (15), colocando um espaçador de plástico (13) entre cada pilar e a placa, e um o-ring (18) de borracha entre a cabeça de cada parafuso e a placa;

11. Fazer as ligações dos servos (19), baterias, sensores de infra-vermelhos (12) e ultra-sons (4) com a placa de circuito impresso principal (17);

12. Encaixar a placa de circuito auxiliar (16) na placa de circuito impresso principal (17) e ligar os fios dos sensores de contacto na placa auxiliar.

2.2 Projecto Electrónico

Nesta secção é descrito o desenvolvimento das placas de circuito impresso utilizadas no SALbot. Foram concebidos três tipos de placas de circuito impresso: uma placa de circuito principal (figura 2.7) para cada SALbot; uma placa de circuito auxiliar (figura 2.8) para cada SALbot; e uma placa de interface de comunicação *wireless* (figura 2.9) para o computador comunicar com o SALbot. A placa principal é responsável pelo controlo do robô e da aquisição dos sensores de distância infra-vermelhos e ultra-sons. Enquanto que a placa auxiliar é responsável pela comunicação *wireless*, aquisição dos sensores de contacto e interface de programação do microcontrolador. A placa de interface de comunicação *wireless* serve unicamente para a comunicação entre o computador e o SALbot.

A placa de circuito principal foi desenvolvida com base na placa do Mark III. Esta contém o microcontrolador que controla o robô; a interface de porta série RS-232, controlado por um MAX232 ACPE+; os terminais de alimentação da placa e dos servos; as portas dos servos, dos sensores de infra-vermelhos e de ultra-sons; a porta de expansão de 40 pinos da placa; uma unidade de memória EEPROM 24LC1025 com 1 Mbit; uma porta de expansão da interface $I^2C^{™}$, que é configurada por 6 pinos; um botão de *reset*; e um botão *on-off*. Contém ainda dois *LEDs*: um verde que indica se o circuito da placa está ligado e um vermelho que indica que a bateria de alimentação do circuito está fraca. O *LED* vermelho é controlado pelo regulador de tensão MAX667, que, recorrendo a um divisor resistivo, acende o *LED* quando a bateria baixa para perto dos 5V.

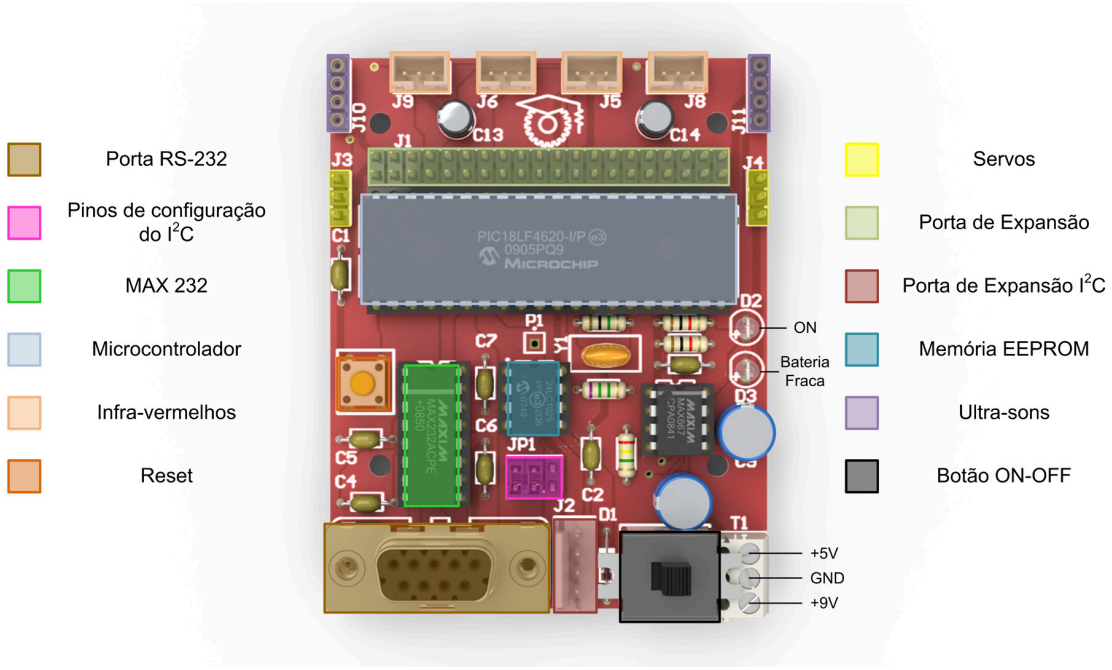


Figura 2.7: Placa de circuito principal do SALbot.

A placa de circuito auxiliar foi desenvolvida de raiz para a interface de comunicação *wireless* e aquisição dos sinais dos sensores de contacto. Esta placa contém um dispositivo XBee, responsável pela comunicação *wireless*, que é descrito detalhadamente no próximo capítulo; oito terminais duplos, para a aquisição dos sinais dos sensores de contacto de cada zona de embate; uma porta de expansão de 40 pinos, que faz a ligação entre a placa de circuito principal e esta placa; um botão de on-off da comunicação *wireless*; um botão de reset do XBee; um regulador de tensão LM1117 DT-3.3, que regula a tensão de 5V vinda da placa de circuito principal para 3.3V para a alimentação do XBee; e, ainda, quatro *LEDs* de interface visual do estado da comunicação *wireless*. Um *LED* amarelo para indicar a recepção de dados, um vermelho para indicar o envio de dados, um azul que indica a potência do sinal recebido, ou RSSI (*Received signal strength indication*), e um verde que indica se o circuito de comunicação *wireless* está ligado.

A placa de interface de comunicação *wireless* foi também desenvolvida de raiz para a comunicação entre o computador e o SALbot. O computador recebe e envia informações para a placa a partir de uma porta série DB-9 implementada na placa. A comunicação série entre o computador e a placa é controlada por um MAX3232 CPE. A placa possui também um dispositivo XBee, responsável pela comunicação *wireless*, e um terminal de ligação da alimentação da placa, que é alimentada por uma pilha de 9V. À semelhança da placa de circuito auxiliar do SALbot, também a placa de interface de comunicação *wireless* para o computador, integra quatro *LEDs* de interface visual do estado da comunicação *wireless*. Um *LED* amarelo para indicar a recepção de dados, um vermelho para indicar o envio de dados, um azul que indica a potência do sinal recebido, ou RSSI (*Received signal strength indication*), e um verde que indica se o circuito de comunicação *wireless* está ligado.

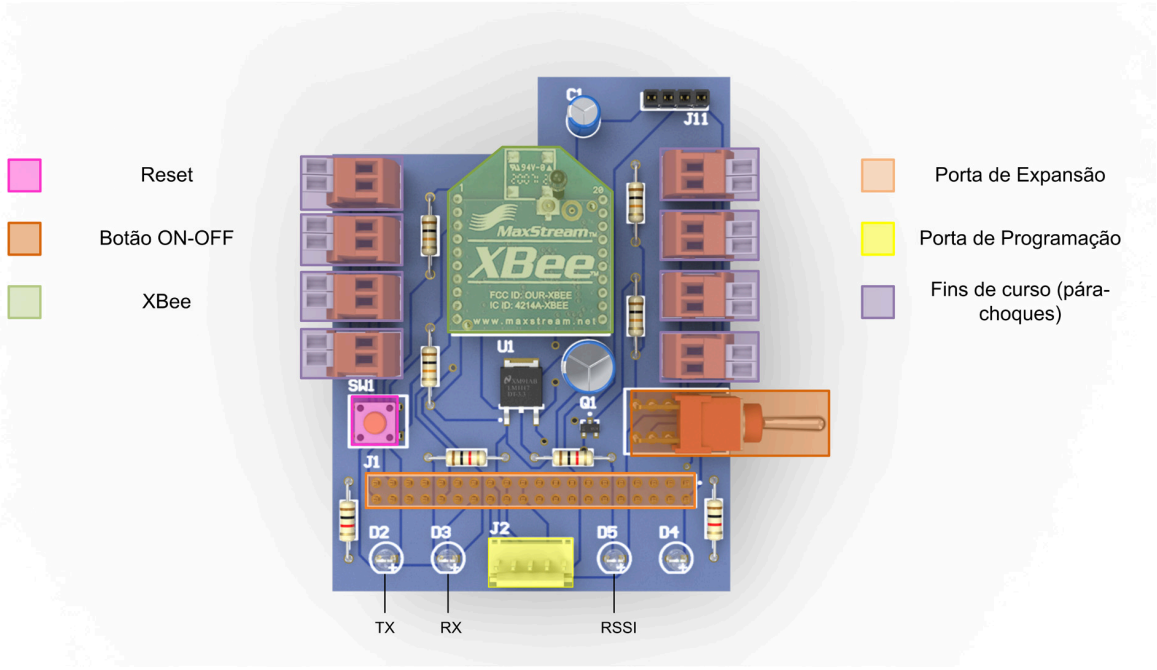


Figura 2.8: Placa de circuito auxiliar do SALbot.

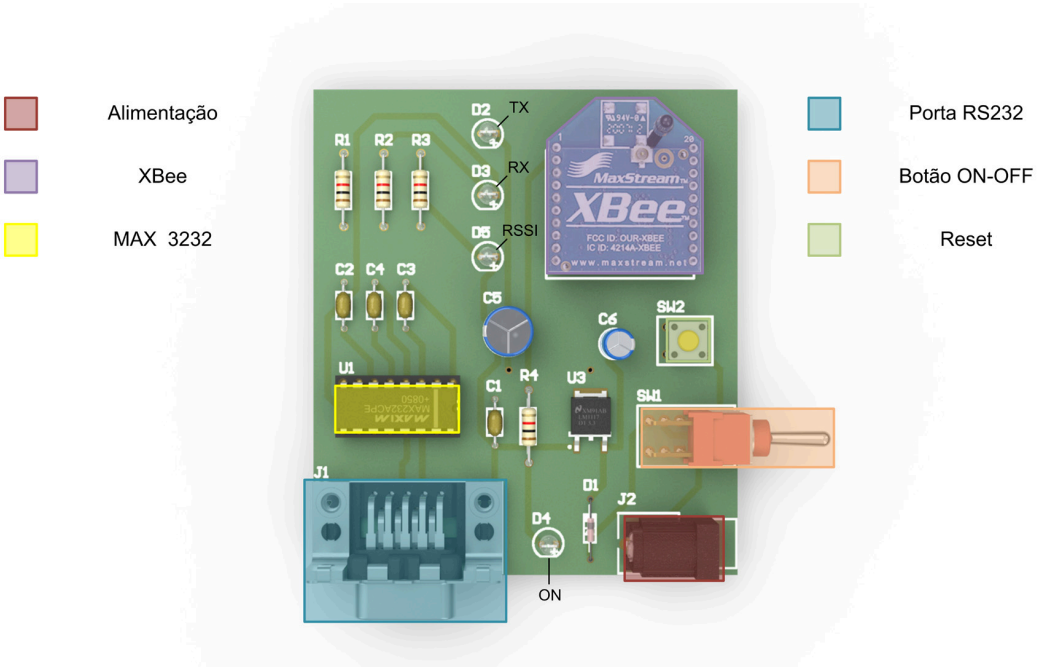


Figura 2.9: Placa de interface de comunicação.

Capítulo 3

Software e Firmware

"É fazendo que se aprende a fazer aquilo que se deve aprender a fazer."

Aristóteles

Neste capítulo é descrito o software de aprendizagem desenvolvido no SALbot, assim como o firmware dos dispositivos e os protocolos de comunicação utilizados. Primeiro, é descrita a arquitectura do sistema dos robôs, isto é, de que forma os dispositivos estão interligados e como são feitas as comunicações. Seguidamente, é abordado o firmware e o funcionamento dos diversos dispositivos, ou seja, a forma como os dispositivos estão programados para interagir com outros dispositivos e de que forma é realizada essa interacção. Depois, são descritos os protocolos de comunicação utilizados, nomeadamente o protocolo de comunicação desenvolvido para o envio e recepção de dados entre o SALbot e o computador, o $I^2C^{™}$, utilizado entre o microcontrolador e uma unidade EEPROM externa, e o sinal PW, utilizado para controlar os servos. Por fim, é descrito o software implementado no microcontrolador, para a aprendizagem dos robôs.

3.1 Arquitectura do SALbot

Em primeiro lugar, é importante explicar o funcionamento da arquitectura do sistema do SALbot (figura 3.1). Como já foi referido no capítulo anterior, o SALbot possui duas placas de circuito impresso: a placa de circuito principal e a auxiliar, sendo a última uma expansão da principal. Na placa de circuito principal está implementado um microcontrolador programado com um algoritmo de aprendizagem. Além dos cálculos específicos do algoritmo de aprendizagem, o microcontrolador é responsável por comandar os servos que fazem mover o robô, ler os valores das distâncias dos sensores de distância e o estado dos sensores de contacto (que estão ligados à placa de circuito auxiliar) necessários para o algoritmo de aprendizagem, escrever dados relativos à aprendizagem na memória EEPROM externa e enviar, e receber, dados pela porta série, nomeadamente enviar dados, relativos à aprendizagem ou ao estado do robô, e receber ordens. Os dados são enviados e recebidos pela porta série do microcontrolador pelos pinos TX e RX. Quando os dados são enviados, o microcontrolador envia-os para o módulo XBee que se encontra na placa de circuito auxiliar. Depois, o módulo XBee envia os dados, em ondas rádio moduladas usando o protocolo IEEE 804.15.4, para outro módulo XBee que se encontra na placa de interface de comunicação, a partir do qual, os mesmos dados são enviados para um computador pela porta série usando o protocolo de comunicação RS-232. Relativamente ao envio de dados do computador para o SALbot,

é invertido o processo descrito anteriormente. Isto é, os dados são enviados pelo computador para o XBee na placa de interface de comunicação, que são enviados para o XBee na placa de circuito auxiliar e, por fim, chegam ao microcontrolador.

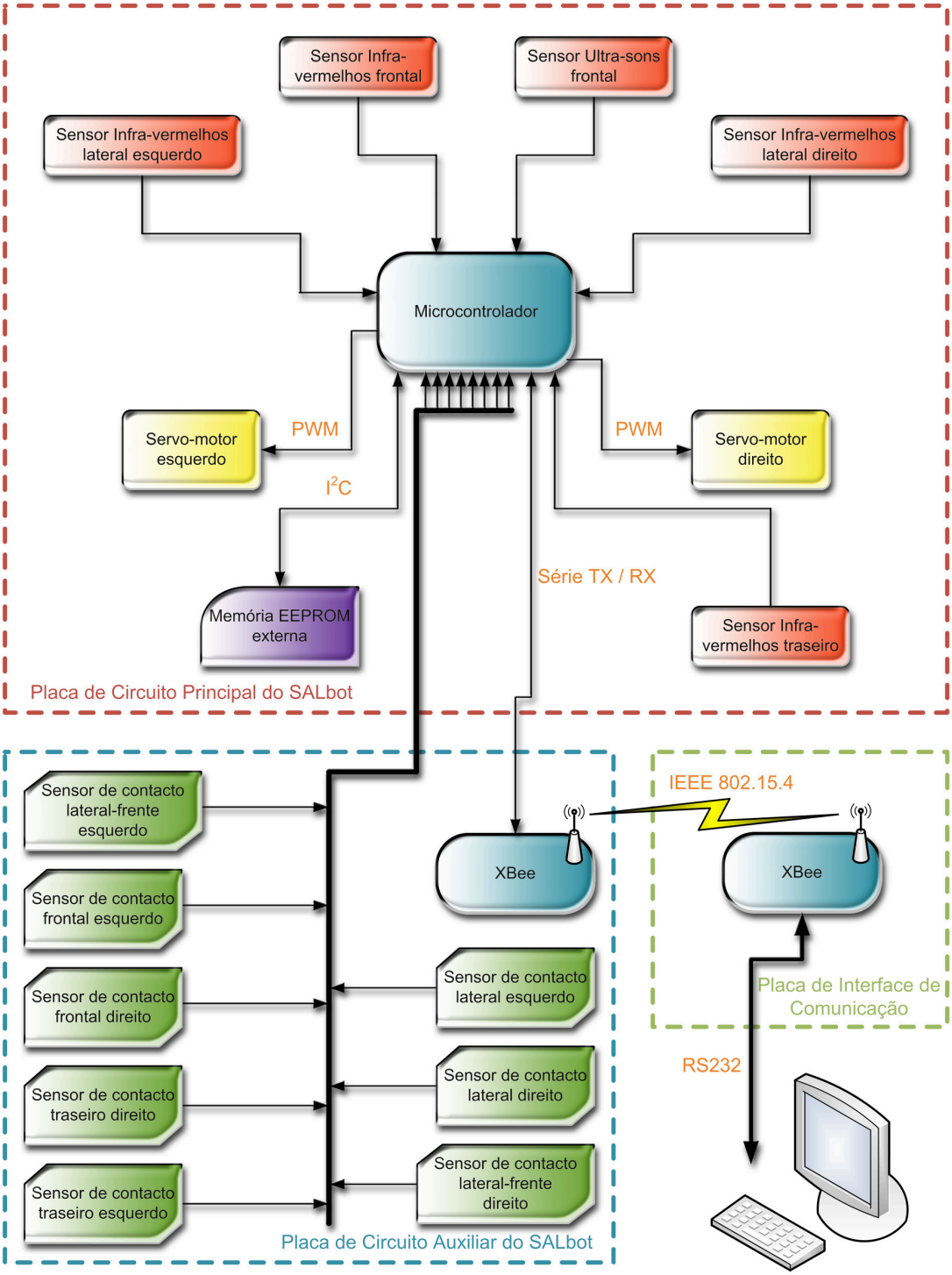


Figura 3.1: Arquitectura do sistema do SALbot.

3.2 Firmware e funcionamento dos dispositivos

3.2.1 Sensores de contacto

Os sensores de contacto implementados no sistema de pára-choques foram os DB3C-B1RC da Cherry. Estes sensores de contacto não são mais do que detectores de fim de curso, accionados por uma haste com uma roda na extremidade. Tais dispositivos funcionam como um interruptor normalmente aberto, entre os 5V de alimentação e o pino do microcontrolador do respectivo sensor, que é fechado quando o sensor é activado. Assim, os 5V são enviados para o pino do microcontrolador, indicando a existência de um contacto. Para evitar o flutuamento da tensão nos pinos do microcontrolador, o que induziria o microcontrolador em erro, foram implementados "*pull-downs*" nos pinos do microcontrolador relativos aos sensores de contacto. Isto é, cada pino tem uma resistência de $1k\Omega$ ligada à massa.



Figura 3.2: Fim de curso DB3C-B1RC.

3.2.2 Sensores de distância infra-vermelhos

Os sensores de infra-vermelhos utilizados foram os GP2D120 da Sharp, que permitem medir distâncias entre 30 mm e 400 mm. Cada um destes sensores possui três tipos de ligações: uma de alimentação a 5V, a ligação à massa e o sinal, que é enviado para o microcontrolador. Esse sinal é analógico, variando entre 0V e 5V, e representa a distância medida pelo sensor.



Figura 3.3: Sensor de distância infra-vermelhos Sharp GP2D120.

Basicamente, este tipo de sensores determina a distância a um objecto utilizando triangulação. Um emissor envia um feixe de luz infra-vermelha, que, caso encontre um objecto, é reflectido de volta, e é captado por um pequeno *array* linear CCD (*charge-coupled device*), formando um triângulo. O feixe infra-vermelhos, antes de chegar ao CCD, passa por uma lente que distribui o feixe infra-vermelho por diversos pontos do CCD, dependendo do ângulo da base do triângulo. Sabendo o ângulo da base do triângulo, dado pelo CCD, é então possível calcular a distância do sensor ao objecto.[9][28][14]

GP2D120X

Sinal analógico (V)	valor (contagens)	distância (mm)
0,325	67	400
0,375	77	350
0,425	87	300
0,525	108	250
0,65	133	200
0,735	151	180
0,8125	166	160
0,925	189	140
1,05	215	120
1,25	256	100
1,4	287	90
1,575	323	80
1,775	364	70
2,025	415	60
2,35	481	50
2,75	563	40
3,5	717	30

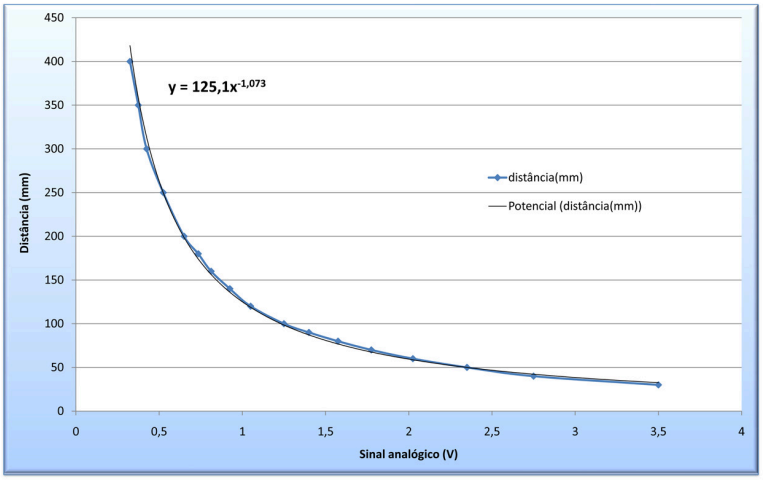


Figura 3.4: Gráfico dos valores do sinal analógico em função da distância do sensor GP2D120.

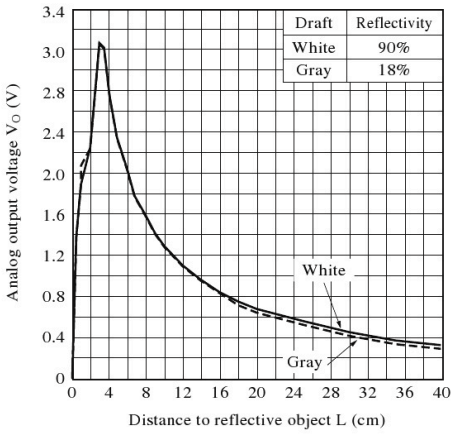


Figura 3.5: Gráfico do fabricante da voltagem em função da distância do GP2D120.[26]

Apesar deste tipo de sensores ser quase imune à interferência da luz ambiente e não depender da cor do objecto detectado, é extremamente difícil detectar objectos transparentes. O sinal analógico enviado pelo sensor não é linearmente proporcional à distância (figura 3.5) e o valor do ângulo medido pelo CCD varia com a temperatura ambiente e com o nível de reflectividade do objecto detectado. Assim, para uma melhor precisão das distâncias obtidas por estes sensores, retiraram-se vários valores do sinal enviado pelo sensor para diversas distâncias medidas, a partir dos quais, se

construiu um gráfico do valor do sinal analógico em função da distância medida, que é possível ver na figura 3.4. Com esses valores, e usando uma regressão exponencial, que é a que mais se adequa aos valores obtidos, obteve-se uma função aproximada da distância medida, em função do sinal analógico enviado pelo sensor.

Embora a distância precisa, em milímetros, da distância entre o robô e um objecto não seja fundamental para o algoritmo de aprendizagem, sentiu-se a necessidade de fazer corresponder os valores analógicos, lidos pelo microcontrolador, a medidas de distância, para serem visualizadas na interface gráfica. Dando, assim, uma noção ao utilizador das distâncias detectadas pelos sensores.

3.2.3 Sensores de distância ultra-sons

Os sensores de ultra-sons instalados nos robôs foram os LV-MaxSonar®-EZ0™, que permitem a medição de distâncias entre 150 mm e 6450 mm. Cada sensor possui vários tipos de interface para a leitura da distância medida, nomeadamente porta série, PWM e sinal analógico. Por ser de mais fácil implementação e de serem necessários menos ligações ao sensor, utilizou-se o sinal analógico para a leitura da distância medida pelo sensor. Este tipo de interface de medição é semelhante ao do GP2D120; o sensor tem apenas três terminais: um de alimentação a 5V, a ligação à massa e o sinal analógico. Porém, ao contrário do GP2D120, o sinal analógico vai dos 0V até à tensão de alimentação V_{CC}, neste caso 5V, proporcionalmente à distância medida. Assim, o sinal analógico apresenta uma resolução de $3.845 \times 10^{-4}\ V \cdot mm^{-1}$.



Figura 3.6: Sensor de distância ultra-sons LV-MaxSonar®-EZ0™.

Nos sensores de distância por ultra-sons, a distância a que este se encontra de um objecto é calculada através da medição do tempo de voo de um feixe de sons ultra-sónicos. No sensor, existe um emissor que emite um feixe direccional de ultra-sons, que não são mais do que treze ondas mecânicas com uma frequência elevada de 42kHz, que não são detectadas pelos ouvidos humanos. Essas ondas mecânicas viajam pelo ar à velocidade do som (cerca de 343 $m \cdot s^{-1}$ em ar seco a 20°) e, quando encontram um obstáculo no seu caminho, são reflectidas de volta para o sensor onde são captadas pelo receptor; as ondas reflectidas designam-se por eco. Assim, o sensor emite um feixe de ultra-sons e fica à espera do seu eco. Quando recebe o eco, o sensor determina o tempo que passou desde o envio do feixe até à recepção do eco e, sabendo a velocidade do som, a distância, do sensor ao objecto que reflectiu o som, é também facilmente determinada. O sensor envia, a cada 49 ms, um feixe de ultra-sons e coloca o PWM no nível lógico 1, que se mantém durante 37.5 ms se não for recebido um eco, caso seja recebido um eco o PWM é colocado no nível lógico 0. Do restante tempo, menos de 4.7 ms são usados para corrigir o nível do sinal analógico e os últimos 4.7 ms são para enviar dados pela porta série.[28]

Este sensor pode ser auto-calibrado, para tal, apenas é necessário que passados 250 ms, após a inicialização do sensor, o pino RX esteja aberto ou seja colocado no nível lógico 1. Se isso acontecer o sensor irá correr um ciclo de calibração de 49 ms e, seguidamente, uma leitura da distância, demorando assim aproximadamente 100 ms.

Para objectos entre os 0 mm e 150 mm, o sensor envia sempre o valor correspondente a 150 mm. Outra desvantagem é o facto do sensor não realizar uma reajustamento da velocidade do som em cada medição, relativamente à temperatura e humidade, que a influenciam. Em vez disso, a calibração do sensor consiste na compensação em relação ao padrão de *ringdown* do sensor, para limitar a distância mínima de medição. Se a temperatura e/ou humidade se alterarem durante a operação do sensor, este pode apresentar medidas erróneas, pelo que será necessário recalibrar-lo para este se ajustar ao novo padrão de *ringdown*. Este padrão de *ringdown* tem a ver com o padrão de recepção do eco. Como o dispositivo, do sensor, que envia as ondas de ultra-sons é o mesmo que detecta a sua recepção, enquanto o sensor envia o feixe de ultra-sons, está, ao mesmo tempo, a receber, daí o termo *ringdown*. Assim, o sensor mede o tempo que demora a enviar o feixe de ultra-sons, e com esse tempo é definido a distância mínima de medição.

3.2.4 Servomotores

Para a movimentação do SALbot, instalaram-se dois servos com a configuração diferencial de duas rodas. Isto é, os servos estão fixados no centro do robô, o que permite que este se mova num sentido movendo as rodas no mesmo sentido, ou que gire em torno do seu próprio eixo, movendo as rodas em sentidos opostos. Além disso, várias combinações de velocidades e direcções de rotação em cada roda permitem ao robô uma infinidade de movimentos.



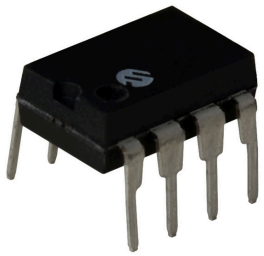
Figura 3.7: Servo Futaba® S3003.

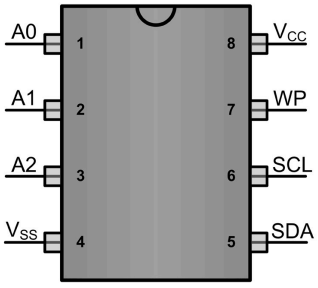
Os servos usados, como anteriormente já foi referido, foram os S3003 da Futaba®. Estes servos estão preparados para uma amplitude de movimentos de 90° para cada lado do ponto central, pelo que, para permitir uma rotação contínua, foi necessário alterar o hardware dos servos, descrito no capítulo anterior.

Cada servo é controlado pelo microcontrolador através de um sinal PW (*Pulse-Width*), que é abordado na secção de "Protocolos de comunicação e comando" (3.3). Devido à alteração realizada para a rotação contínua dos servos, para que estes rodem numa direcção à velocidade máxima, basta enviar o sinal que corresponderia à movimentação deste para a posição máxima dessa direcção. Variando o sinal de comando para uma posição duma direcção, varia-se a velocidade de rotação do servo.

3.2.5 Memória EEPROM externa

Para que, um robô autónomo orientado para aprendizagem, possa ser desligado, sem contudo perder a informação que adquiriu durante a sua aprendizagem, necessita de um espaço para guardar dados, relativos à aprendizagem, que não se alterem com a ausência de alimentação eléctrica. Um destes tipos de armazenamento de dados é a EEPROM (*Electrically-Erasable Programmable Read-Only Memory*), um tipo de memória não-volátil que permite que os dados armazenados não sejam apagados quando a alimentação eléctrica é interrompida. Houve a necessidade da implementação de uma unidade de EEPROM externa, uma vez que a dimensão deste tipo de memória disponível nos microcontroladores é insuficiente (1024 *bits* para a família da série PIC18 e 4096 *bits* para a família dsPIC), para a quantidade de dados relativos à aprendizagem que são necessários armazenar, mesmo para os métodos de aprendizagem mais simples.





A0	Pinos de configuração do endereço do dispositivo
A1	
A2	Não utilizado. Deve ser ligado ao V_{CC}.
V_{SS}	Massa
V_{CC}	De +2.5 a 5V
WP	Pino de protecção contra escrita
SCL	Linha de *Serial Clock*
SDA	Linha de *Serial Data*

(a) (b)

Figura 3.8: Unidade de EEPROM de interface série 24LC1025 da Microchip. (a) Fotografia do 24LC1025; (b) *Pinout* do 24LC1025.

O dispositivo de armazenamento de memória EEPROM implementado foi o 24LC1025 da Microchip com 1 Mbit de memória, como já foi referido na secção 2.2. O microcontrolador acede a este dispositivo, escrevendo e lendo dados em posições específicas da memória disponível, através da comunicação por I^2C™, que é descrita na secção de "Protocolos de comunicação e comando" (3.3). A memória deste dispositivo está agrupada em páginas de 128 *bytes*, cada *byte* é composto por oito *bits*, ao qual está associado um endereço único que vai de 00000h a 1FFFFh. É possível escrever um *byte* de cada vez ou uma página de *bytes*. Quanto à leitura dos dados, é possível ler um *byte* ou uma leitura sequencial de *bytes*. Este dispositivo tem dois pinos que, ligando cada um ao V_{CC} ou à massa, configuram o endereço I^2C™ do dispositivo, o que permite a utilização de um máximo de quatro destes dispositivos na mesma interface I^2C™. Além disso, tem ainda um pino de protecção de dados que, estando conectado ao V_{CC}, não permite operações de gravação, sendo apenas permitido operações de leitura.

Embora este dispositivo seja de maior capacidade de memória EEPROM apresenta face a outros dispositivos disponíveis no mercado, ele pode ser permutado por outro dispositivo de memória EEPROM, desde que a interface de comunicação seja $I^2C^{™}$. Para o caso de ser necessário mais capacidade de armazenamento de dados, embora na placa de circuito principal do SALbot apenas seja possível fixar uma unidade EEPROM externa, existe uma porta de expansão $I^2C^{™}$, que permite instalar até um máximo de quatro dispositivos similares no mesmo bus $I^2C^{™}$.

3.2.6 XBee

Como já foi referido, a comunicação sem fios, entre o SALbot e o computador, fica a cargo dos módulos XBee. Mais precisamente os módulos XB24-AWI-001, que são módulos com uma antena de fio metálico, que comunicam através do protocolo *wireless standard* IEEE 802.15.4.

Estes módulos possuem uma interface série, a partir da qual, é possível enviar e receber dados relativos à comunicação sem fios. Assim, no SALbot, o microcontrolador está ligado por porta série ao XBee e, na placa de interface de comunicação, o XBee está ligado por porta série a um MAX232 que transforma os sinais para o protocolo RS232. Isto é, tal como já foi referido, quando o robô envia dados para o computador o microcontrolador envia os dados pela porta série para o XBee. Este envia os dados pelo protocolo IEEE 802.15.4 para outro XBee, o qual os envia para o MAX3232 que, por sua vez, transforma os sinais para serem enviados para o computador através do protocolo RS232. Essa ligação por porta série consiste em dois pinos o TX e o RX. O TX é o pino de transmissão de dados e o RX é o pino de recepção de dados.



Figura 3.9: Módulo XBee XB24-AWI-001 da Digi®.

Os módulos XBee utilizados têm um alcance máximo de 30 m em espaços urbanos ou fechados, e de 90 m para espaços abertos com uma taxa de transferência de 250 kbps. A gama de taxa de transferência de dados pela interface série vai dos 1200 bps até aos 250 kbps, sendo aceites taxas de *baud rate* não estandardizadas. A comunicação sem fios é feita através do protocolo IEEE 802.15.4, operando na gama de frequências de 2.4 GHz, num dos 16 canais previstos para este protocolo nesta gama de frequências. A transmissão de dados sem fios é feita com uma potência de 1 mW, ou 0 dBm, e a sensibilidade de recepção de dados é de -92 dBm.[10]

Como os módulos XBee operam a 3.3 V, foi necessário implementar, na placa de circuito auxiliar, dispositivos para converter os sinais lógicos de 3.3 V para 5 V e de 5 V para 3.3 V da porta série. A conversão dos sinais lógicos de 3.3 V para 5 V é garantida por um MOSFET (Metal Oxide Semiconductor Field Effect Transistor) BSS138, enquanto que a conversão dos sinais lógicos de 5 V para 3.3 V é garantida por um divisor resistivo de duas resistências de 10 kΩ.

A configuração dos módulos XBee pode ser feita através de comandos "AT" ou através de um programa disponibilizado pela Digi®, o X-CTU, onde, através de uma interface gráfica, é possível configurar todos os parâmetros de qualquer módulo XBee. Na medida em que, a comunicação sem fios é realizada em modo transparente, apenas é necessário configurar alguns parâmetros dos módulos XBee para que estes possam comunicar entre si. Basicamente, é necessário configurar os parâmetros CH (canais), ID (PAN ID), DH (Endereço de destino mais significativo), DL (Endereço de destino menos significativo), SC (canais procurados) e BD (*Baud Rate*). O parâmetro CH não é mais do que um dos canais, disponíveis pelo protocolo IEEE 802.15.4 para a gama de frequências de 2.4 GHz, usado para a comunicação sem fios entre os módulos XBee. O parâmetro ID consiste na designação da PAN (*Personal Area Network*) ID, isto é, o identificador da rede, que pode ser entre 0 a 0xFFFF. Caso o valor do parâmetro seja 0xFFFF, o envio de dados é feito em *broadcast* para todas as PAN. O parâmetro DH e DL correspondem ao endereço de 64-*bit* do dispositivo de destino, sendo DH os 32 *bits* mais significativos e DL os 32 *bits* menos significativos do endereço de destino. Se o valor de DH for 0 e DL for menor do que 0xFFFF, a transmissão de dados é feita usando o endereço de 16-*bit*. Se DH for 0 e DL 0xFFF, a transmissão de dados é feita em *broadcast* para a PAN. O parâmetro SC não é mais do que uma máscara de *bits* para se configurar quais os canais, disponíveis pelo protocolo IEEE 802.15.4 para a gama de frequências de 2.4 GHz, nos quais, o módulo XBee pode procurar outros dispositivos.

Por fim, o parâmetro BD corresponde à taxa de transferência de dados da porta série, em que os valores de 0 a 7 correspondem às taxas de transferência de dados estandardizadas de 1200 bps a 115.2 kbps e os valores de 0x80 a 0x3D090 são usados para taxas de transferência de dados não estandardizadas até 250 kbps.[10]

Embora os módulos XBee apenas sejam utilizados para a comunicação entre os robôs e o computador, estes podem ser programados para fazerem mais do que isso. Os módulos XBee, além dos pinos de porta série, alimentação e massa, possuem ainda oito entradas e saídas digitais, das quais seis podem ser configuradas como entradas analógicas, um pino exclusivo para entrada digital, outro pino exclusivo para saída digital e duas de PWM. Tudo isto possibilita o controlo e aquisição de dados de dispositivos remotamente sem a necessidade de um microcontrolador ou quaisquer outros dispositivos de controlo.

Os módulos XBee implementados, quer na placa de circuito auxiliar dos SALbot, quer na placa de interface de comunicação, podem ser permutados por outros módulos XBee, inclusive pelos XBee-PRO ou XBee com o protocolo ZigBee® quer da série 1 quer da série 2. No entanto, os XBee da série 2 não podem comunicar com os da série 1, nem mesmo os módulos com o protocolo ZigBee®. Da mesma forma que os módulos com o protocolo ZigBee® da série 1, também os módulos da série 2 necessitam que um dos módulos de uma rede seja configurado como coordenador. Apesar das diferenças dos módulos XBee da série 1 para a série 2, só é necessário alterar o código do microcontrolador no caso de se pretender usar o modo API (*Application Programming Interface*) em vez do modo transparente.

Os módulos XBee-PRO são semelhantes aos outros módulos XBee, no entanto os XBee-PRO possuem um alcance máximo de 90 m em espaços urbanos ou fechados e 1.6 km para espaços abertos. A potencia máxima de transmissão de dados é de 63 mW, ou 18 dBm, e a sensibilidade de recepção de dados é de -100 dBm. Outra diferença é que os XBee-PRO não utilizam os últimos

quatro canais disponíveis pelo protocolo IEEE 802.15.4 para a gama de frequências de 2.4 GHz.[10]

No entanto, uma vez que os SALbot foram projectados para a realização de testes e ensaios de algoritmos de aprendizagem em ambientes laboratoriais, não existe uma necessidade significativa da implementação de módulos XBee-PRO, já que os módulos implementados possuem um alcance de transmissão de dados mais do que suficiente para o efeito.

3.3 Protocolos de comunicação e comando

Nesta secção são descritos os protocolos de comunicação, do SALbot com o computador e do microcontrolador com o dispositivo de EEPROM externo, e o protocolo de comando dos servos.

3.3.1 Sinal PW de comando dos servomotores

Os servos, como já foi referido, são comandados através de um sinal PW enviado pelo microcontrolador. Os servos implementados foram concebidos para se moverem para uma determinada posição angular. Essa posição angular é dada através de um sinal PW. Os sinais PW usados nos servos, normalmente, têm uma frequência de 50 Hz, isto é, um pulso é gerado a cada 20 ms. A largura de cada pulso indica a posição angular para a qual se deseja que o servo se desloque.

No caso do S3003, como é possível ver na figura 3.10, para um pulso com uma largura de 320 μs, o servo posiciona-se a -90°; para um pulso de 2300 μs, o servo posiciona-se a 90°; e para um pulso de 1320 μs, o servo mantêm-se na posição central.[6][1]

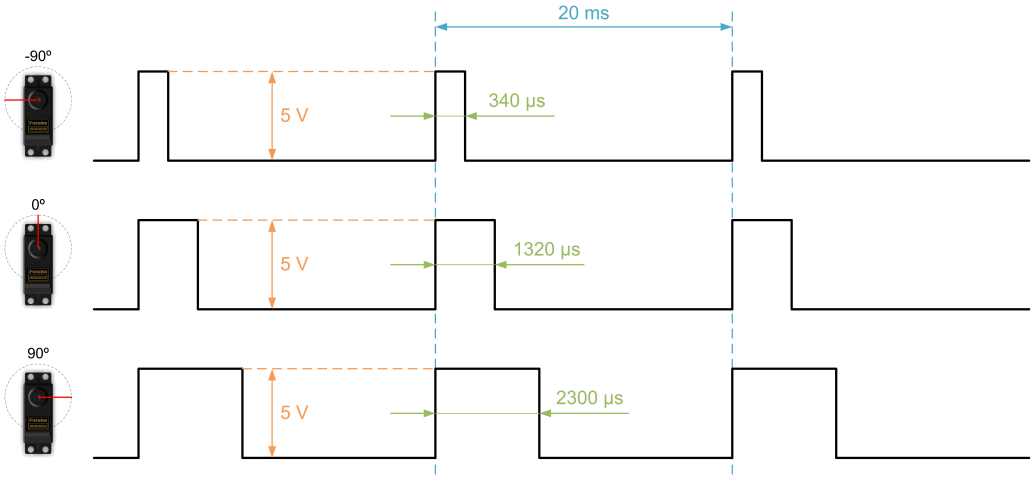


Figura 3.10: Sinal PW do servo S3003 da Futaba®.

No entanto, após a alteração efectuada nos servos para poderem rodar continuamente, o controlador do servo tem a informação de que o servo está sempre posicionado na posição central. Pelo que, ao enviarmos um sinal PW para comandar o servo para a posição de -90°, este vai tentar posicionar-se nessa posição, mas como o controlo interno indica sempre a posição central, o servo continua a rodar. Assim, para rodarmos o servo no sentido horário basta enviar um sinal PW com um pulso entre 1321 μs e 2300 μs, em que a velocidade aumenta quanto maior for a largura do pulso. Para rodarmos o servo no sentido anti-horário deve enviar-se um sinal PW com um pulso entre 1319 μs e 320 μs, em que a velocidade aumenta quanto menor for a largura do pulso.

3.3.2 Comunicação I²C™ com o dispositivo de EEPROM externo

A comunicação entre o microcontrolador e o dispositivo de EEPROM externo é realizada através de uma interface I²C™. A comunicação por I²C™ usa um barramento de duas ligações, o SDA (*Serial Data*) e o SCL (*Serial Clock*). A ligação SDA é bidireccional e é utilizada para o enviar e receber dados e endereços de um dispositivo, em *bytes* de oito *bits*, enquanto que a ligação SCL é utilizada para sincronizar a transferência de dados entre os dispositivos.

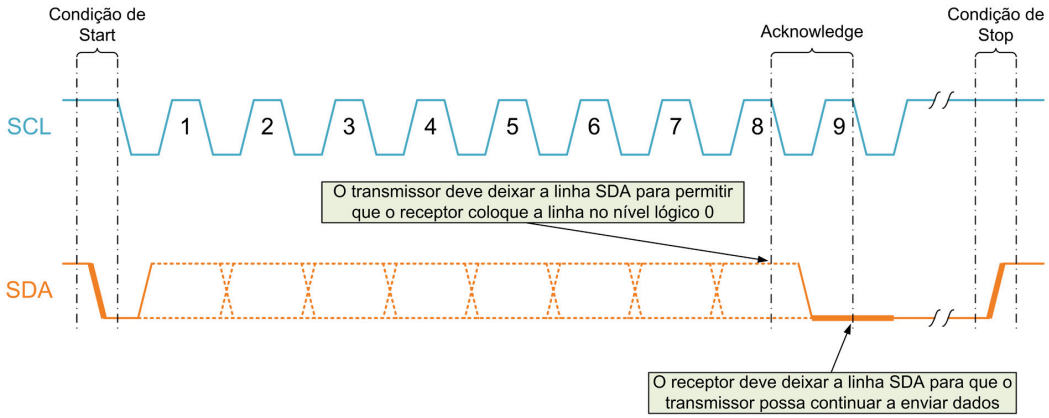


Figura 3.11: Composição da sequência de transferência de dados no I²C™.

A interface de comunicação I²C™ requer que a comunicação seja controlada por um dispositivo *Master*, que, neste caso, é o microcontrolador. É o *Master* que gera os impulsos na linha SCL, controla o acesso ao barramento e gera as condições de *Start* e *Stop*. Os outros dispositivos ligados ao mesmo barramento I²C™ são designados de *Slave*. Qualquer dispositivo, tanto o *Master* como um *Slave*, pode ser transmissor (enviam dados) ou receptor (recebem dados).

No I²C™, a transferência de dados do transmissor para o receptor só deve ser iniciada quando o barramento estiver desocupado, isto é, quando ambas as linhas de SDA e SCL permanecem no nível lógico 1 (à tensão de V_{CC}). Todos os comandos devem iniciar com uma condição de *Start*, que corresponde a uma transição do nível lógico 1 para 0 na linha SDA enquanto a linha SCL permanece no nível lógico 1. E todos os comandos devem terminar com uma condição de *Stop*, que é dada por uma transição do nível lógico 0 para 1 na linha SDA enquanto a linha SCL permanece no nível lógico 1. No final de recepção de cada *byte* o dispositivo receptor deve gerar uma condição de *Acknowledge*, e o *Master* deve gerar um impulso extra na linha SCL para o *bit* de *Acknowledge*. A condição de *Acknowledge* acontece quando o dispositivo receptor coloca a linha SDA no nível lógico 1 e está estabilizado durante o período de nível lógico 1 relativo ao impulso extra gerado na linha SCL.[19]

3.3.2.1 Endereçamento dos dispositivos

Em todos os comandos, o primeiro *byte* a enviar, após a condição de *Start*, é o *byte* de controlo. Como se pode verificar na figura 3.12, o *byte* de controlo é composto por quatro *bits* com o código de controlo, um *bit* relativo à selecção do bloco de memória, dois *bits* de selecção do dispositivo e um *bit* que indica se o tipo de operação é de escrita ou leitura. O código de controlo para os dispositivos 24LC1025 é "1010", em binário.

Uma vez que o 24LC1025 divide a memória em dois blocos de 512 kb, o *bit* de selecção de bloco indica qual o bloco de memória se pretende aceder. Basicamente, o *bit* de selecção do bloco pode ser tratado como o 16º *bit* de endereço de memória. Os *bits* de selecção do dispositivo correspondem ao endereço do dispositivo, com o qual se pretende comunicar. O *bit* que indica o tipo de operação é 0 para operações de escrita e 1 para operações de leitura.

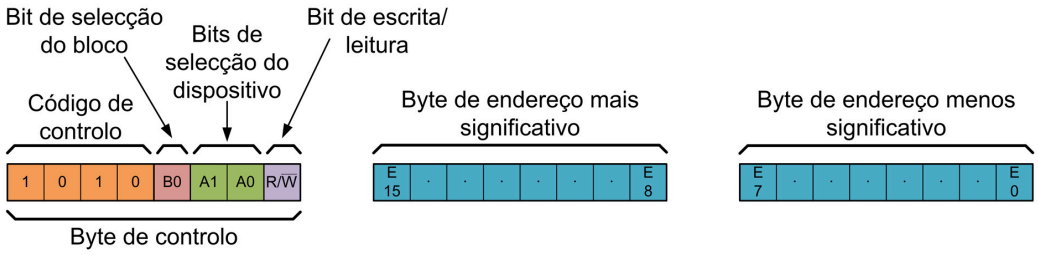


Figura 3.12: Composição da sequência de *bits* do endereçamento de uma mensagem por I²C™.

Depois do *byte* de controlo é enviado o endereço da memória à qual se pretende aceder. O endereço da memória é dividido em dois *bytes*. O *byte* menos significativo contém os oito primeiros *bits* do endereço e o *byte* mais significativo os últimos oito *bits*.

3.3.2.2 Operações de escrita

É possível realizar dois tipos de operações de escrita no 24LC1025: a escrita de dados numa posição da memória e a escrita sequencial de dados em cada posição da memória numa página de 128 *bytes*.

Na operação de escrita de um *byte*, a mensagem é iniciada com a condição de *Start*, seguido do *byte* de comando, no qual o *bit* de tipo de comando deve assumir o valor lógico 0. Seguidamente, após a recepção do *Acknowledge* do receptor, são enviados os *bytes* de endereço, primeiro o mais significativo e depois, após a recepção do *Acknowledge* do receptor, o menos significativo. Por fim, após a recepção de um novo *Acknowledge* do receptor, é enviado o *byte* que será gravado na memória do 24LC1025 e a mensagem é finalizada com uma condição de *Stop*.

A mensagem da operação de escrita de uma página de 128 *bytes* é realizada quase da mesma forma que a operação de escrita de um *byte*. A diferença é que os *bytes* do endereço enviados correspondem ao primeiro endereço, a partir do qual, se pretende escrever e, em vez da condição de stop no final do envio do *byte* de dados, são enviados sequencialmente *bytes* de dados, até um máximo de 127 *bytes* adicionais, e só depois é finalizada a mensagem com a condição de *Stop*. O 24LC1025 auto-incrementa uma posição do *byte* menos significativo do endereço à medida que mais *bytes* de dados são recebidos. Se a quantidade de dados enviados implicar que o *byte* de endereço menos significativo ultrapasse o valor 128, o *byte* do endereço volta ao início da página e escreve os dados restantes por cima dos valores que se encontravam anteriormente nessas posições de memória.

3.3.2.3 Operações de leitura

Existem três tipos de operações de leitura: a leitura do endereço actual, a leitura de um endereço específico e a leitura sequencial.

A operação de leitura do endereço actual é iniciada com o envio de um *byte* de controlo após a condição de *Start*; porém, o *bit* de tipo de comando, ao contrário do que acontece nas operações de escrita, deve assumir o valor lógico 1. Após o envio do *byte* de controlo, o 24LC1025 envia um *Acknowledge* e, em seguida, o *byte* de dados do endereço de memória actual, o *Master* passa o envio de um *Acknowledge*, mas gera uma condição de *Stop* no final da leitura. O dispositivo 24LC1025 possui um contador de endereços que contém o endereço da posição da última memória lida. Assim, o endereço da posição de memória da próxima leitura será o endereço seguinte da última leitura.

A operação de leitura de um endereço específico é iniciada enviando um *byte* de controlo após a condição de *Start*, com o *bit* de tipo de operação a 0. Após a recepção de um *Acknowledge* do receptor, são enviados os dois *bytes* do endereço de memória que se pretende ler, enviando primeiramente o *byte* mais significativo e depois, após a recepção de um *Acknowledge*, o *byte* menos significativo. Seguidamente, é enviado novamente o *byte* de controlo antecedido de uma condição de *Start*, mas, desta vez, o *bit* de tipo de operação deve assumir o valor lógico 1. Depois o 24LC1025 envia um *Acknowledge* e, em seguida, o *byte* de dados, correspondente ao endereço de memória que foi enviado. Por fim, à semelhança da operação de leitura do endereço actual, o *Master* não envia um *Acknowledge* e finaliza a mensagem com uma condição de *Stop*.

A operação de leitura sequencial é idêntica à operação de leitura de um endereço específico. No entanto após 24LC1025 enviar o *byte* de dados relativo ao endereço enviado, é enviado um *Acknowledge* para cada leitura sequencial desejada e que é seguido do envio do *byte* de dados do endereço seguinte pelo 24LC1025. Quando todas as leituras desejadas forem efectuadas o *Master* passa o envio do *Acknowledge* e envia uma condição de *Stop*.

3.3.3 Protocolo de comunicação entre o SALbot e o PC

Uma vez que a comunicação, através dos módulos XBee, entre os SALbot e o computador, é realizada em modo transparente, existe a possibilidade de os SALbot receberem hipotéticas mensagens vindas de outros dispositivos *wireless* os quais poderiam interferir com o funcionamento normal dos robôs. Assim, houve a necessidade de implementar um protocolo que só fosse descodificado pelos SALbot ou pelo programa de interface no computador.

O protocolo foi concebido por forma a que fosse possível o envio de mensagens com comprimento variável, até um máximo de quinze *bytes*, tendo em conta a possibilidade da perda de informações pelo meio da mensagem, já que as comunicações *wireless* são permeáveis a interferências externas. Para permitir o comprimento dinâmico da mensagem foi criado um campo que contém a informação da dimensão da mensagem. E, para evitar a leitura de mensagens com erros, foi criado um campo na mensagem permite validar a mensagem recebida.

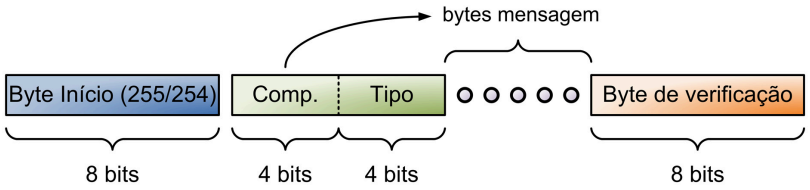


Figura 3.13: Topologia da mensagem de comunicação entre o SALbot e o computador.

A mensagem, como é visível na figura 3.13, é constituída por quatro secções: o *byte* de início, um *byte* com o comprimento e tipo, a mensagem propriamente dita e um *byte* de verificação (ou *checksum*).

O *byte* de início depende do robô que envia ou recebe mensagens. Este *byte*, para o SALbot1, tem o valor 255 decimal e ,para o SALbot2, tem o valor 254 decimal. O segundo *byte* contém o valor do comprimento da mensagem, na parte mais significativa, e o tipo de mensagem, na parte menos significativa. Este comprimento corresponde ao número de *bytes* da mensagem propriamente dita, isto é, o número de *bytes* entre este *byte* e o *byte* de verificação. Depois dos *bytes* da mensagem propriamente dita, existe um *byte* de verificação que, basicamente, permite que a mensagem seja validada por quem a recebeu.

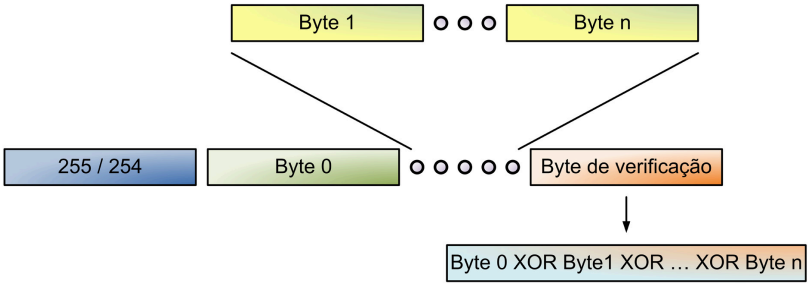


Figura 3.14: Descrição gráfica do *byte* de verificação.

Na figura 3.14 pode visualizar-se como funciona o *byte* de verificação. Independentemente do número de *bytes*, pelo qual é constituída a mensagem, este *byte* corresponde à operação sequenciada de XORs ("ou" exclusivos); ou seja, faz-se a operação XOR do primeiro *byte* com o segundo, depois efectua-se a mesma operação do resultado da anterior com o terceiro *byte* e assim sucessivamente até ao último *byte*.

É através deste *byte* que, quer o programa de interface, quer o programa do SALbot, validam as mensagens recebidas pela porta série.

3.3.3.1 Mensagem de modo de operação

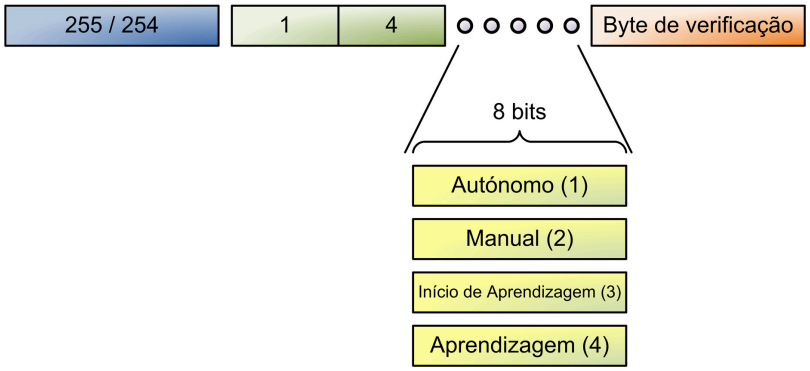


Figura 3.15: Mensagem de alteração do modo de operação do SALbot.

A mensagem de modo de operação (figura 3.15) é enviada pelo programa de interface, para poder comandar o modo de operação do programa do SALbot. A mensagem é constituída apenas por um *byte*, que corresponde ao modo de operação que o programa do SALbot deve tomar: Autónomo, Manual, Inicialização da Aprendizagem ou Aprendizagem.

3.3.3.2 Mensagem de Início/Paragem

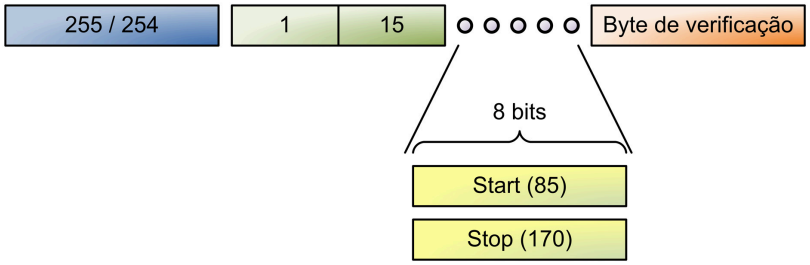


Figura 3.16: Mensagem de Início/Paragem do SALbot.

Quando o SALbot é (re)iniciado, o programa entra no modo Idle (não realiza qualquer operação a não ser a verificação de mensagens recebidas). A mensagem de início/paragem (figura 3.16) é usada para o utilizador iniciar o programa do SALbot a fim deste iniciar o modo de execução (por defeito é o modo Manual). No entanto, sempre que o utilizador desejar, pode enviar uma mensagem para colocar o programa do SALbot no modo *Idle* (enviando o *Stop*).

Esta mensagem é constituída por um *byte* que indica se é para iniciar ou para parar, independentemente do modo de operação do SALbot.

3.3.3.3 Mensagem de pedido de dados

No modo manual, o SALbot quando recebe uma mensagem deste tipo, vai enviar o tipo de dados pedidos. Estes dados podem ser os valores das distâncias dos sensores de distância, o estado dos sensores de contacto, ou os valores da tabela de aprendizagem. Constituída por um *byte*, a mensagem representa o tipo de dados a receber(figura 3.17).

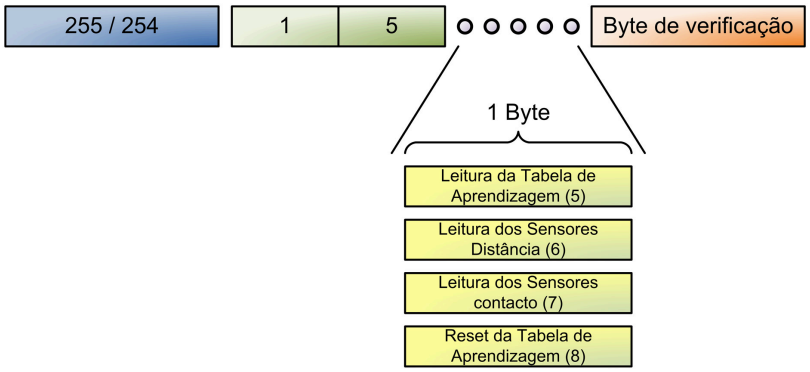


Figura 3.17: Mensagem de pedido de dados.

Embora este tipo de mensagem seja normalmente para pedir dados ao SALbot a partir de uma interface num computador remoto, também é possível instruir o SALbot a apagar a sua tabela de aprendizagem.

3.3.3.4 Mensagem de envio do estado dos sensores de distância

Nesta mensagem, são enviados os dados relativos aos sensores de distância por infra-vermelhos e ultra-sons. Como é possível verificar na figura 3.18, são enviados os valores das distâncias dos sensores de distância em dois *bytes* por cada sensor: o primeiro *byte* com o valor mais significativo e o segundo com o valor menos significativo. Como os valores lidos pelas portas analógicas do microcontrolador são de dez *bits* de resolução, nos *bytes* mais significativos dos valores das distâncias, os últimos seis *bits* são ignorados.

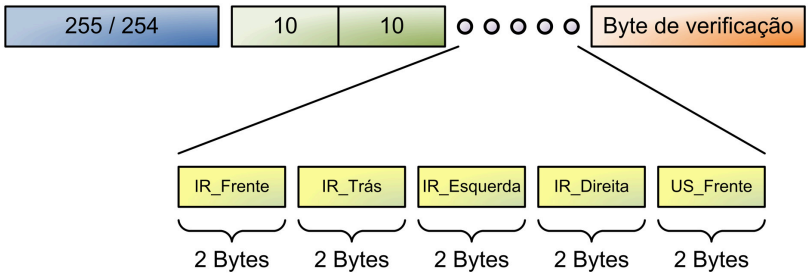


Figura 3.18: Mensagem de envio do estado dos sensores de distância.

3.3.3.5 Mensagem de envio do estado dos sensores de contacto

Uma vez que, os sensores de contacto só têm dois estados (ON e OFF) e estão dispostos por oito zonas de contacto distintas, basta um *byte* para representar o estado de todos os sensores de contacto, em que cada *bit* corresponde ao estado de um sensor. A ordem dos sensores de contacto no *byte* enviado (figura 3.19) corresponde ao *byte* da porta do microcontrolador (PORTD), onde estão implementados os sensores de contacto.

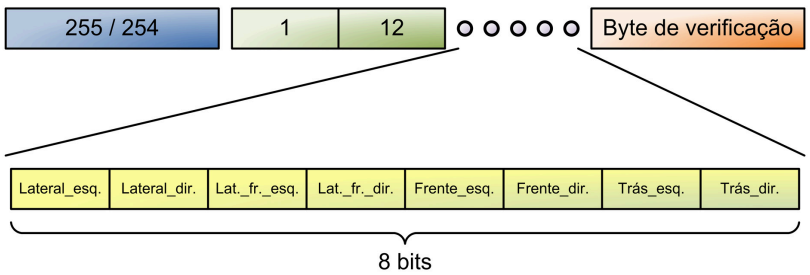


Figura 3.19: Mensagem de envio do estado dos sensores de contacto.

3.3.3.6 Mensagem de envio de um elemento da tabela de aprendizagem

Esta mensagem foi concebida para possibilitar ao SALbot transferir tabelas de aprendizagem para um computador remoto e vice-versa. Assim é possível descarregar tabelas de aprendizagem

do SALbot para um computador remoto, para serem analisadas ou simplesmente serem guardadas e posteriormente poderem ser carregadas para o SALbot.

Em cada mensagem apenas pode enviar-se quinze *bytes* de dados e, mesmo que se pudesse enviar os *bytes* todos da tabela numa só mensagem, tratando-se de uma mensagem muito comprida a enviar por uma comunicação sem fios, a probabilidade da transferência de dados ser interrompida ou corrompida seria grande. Por essas razões, em cada mensagem é enviado apenas um valor de uma posição da tabela de aprendizagem.

Neste tipo de mensagem, como se pode verificar na figura 3.20, são enviados três *bytes*. Os dois primeiros são relativos ao endereço da posição na tabela de aprendizagem, sendo o primeiro *byte* o mais significativo do endereço e o segundo o menos significativo. O terceiro *byte* enviado corresponde ao valor da tabela do respectivo endereço enviado.

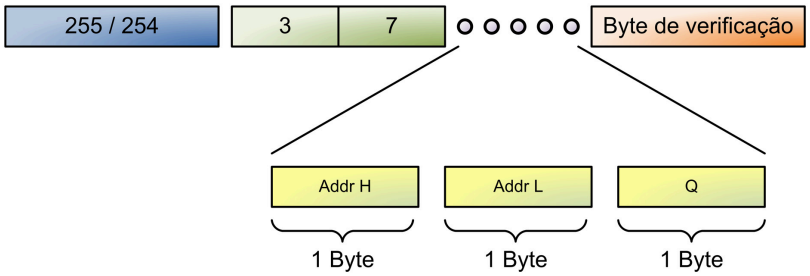


Figura 3.20: Mensagem de envio de um elemento da tabela de aprendizagem.

Para que a linha de comunicações não seja ocupada durante muito tempo com o envio dos valores da tabela de aprendizagem, os valores nulos da tabela de aprendizagem não são enviados.

3.3.3.7 Mensagem de envio do progresso de aprendizagem

Uma das métricas, possíveis para avaliar a aprendizagem em questão, consiste em medir o número de colisões de um robô, em função do número de iterações do algoritmo de aprendizagem. Esta mensagem (figura 3.21), permite ao SALbot enviar dados relativos à progressão da aprendizagem, enviando o número da iteração e o número de colisões, desde o início da aprendizagem.

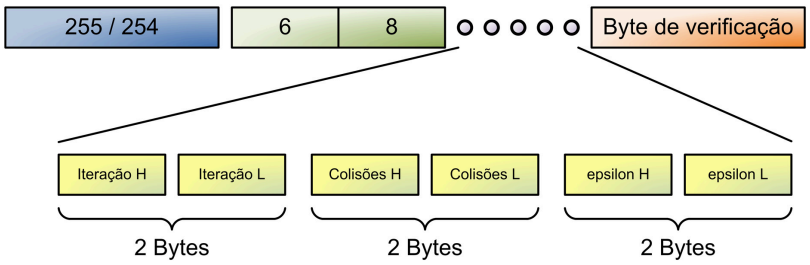


Figura 3.21: Mensagem de envio do progresso de aprendizagem.

Além disso, é também enviado o parâmetro ε do algoritmo de aprendizagem SARSA. Dado que o parâmetro ε representa um número racional entre 0 e 1, este deve ser definido no programa do microcontrolador como um `float`. No entanto, o formato `float` do microcontrolador não é o mesmo de um computador, enquanto o do microcontrolador corresponde a 24 *bits*, no computador

corresponde normalmente a 32 *bits*. Assim, por forma a evitar a conversão de um formato para o outro, o valor enviado na mensagem corresponde a cem vezes o valor de ε, sob a forma de um `unsigned int` (16 *bits*).

3.3.3.8 Mensagem de envio da acção executada

Esta mensagem foi concebida para que seja possível identificar na interface do computador remoto a acção executada por cada SALbot. Nesta mensagem (figura 3.22) é enviado apenas o número correspondente da acção executada pelo SALbot.

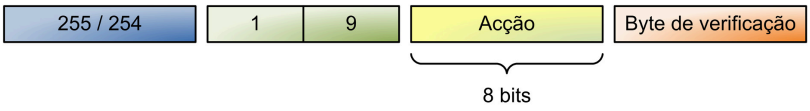


Figura 3.22: Mensagem de envio da acção executada pelo SALbot.

3.4 Software

3.4.1 Algoritmo de aprendizagem

Os dois os SALbot foram programados com o mesmo software, que foi desenvolvido para testar e avaliar o desempenho dos robôs com um algoritmo básico de aprendizagem. O algoritmo de aprendizagem implementado foi o SARSA (*State-Action-Reward-State-Action*), que pode ser visto mais pormenorizadamente no anexo A.3.6. Muito resumidamente, o SARSA (equação 3.2) é um algoritmo de aprendizagem *model-free*, muito idêntico ao *Q-learning* (equação 3.1), sendo que, a maior diferença entre eles, reside no facto de o SARSA ser um método de aprendizagem *on-policy*, enquanto o *Q-learning* é um método de aprendizagem *off-policy*. Ou seja, enquanto no *Q-learning* é estimado o valor máximo de Q do próximo par de estado-acção para o cálculo do valor de Q actual, no SARSA, em vez disso, para o cálculo do valor de Q é usado o valor de Q do próximo par estado-acção após a escolha da próxima acção, usando a mesma política que determinou a acção actual, no estado actual. Nas fórmulas 3.1 e 3.2: a corresponde a uma acção; s corresponde a um estado; r é a recompensa; α é uma constante entre $[0, 1]$ designada por coeficiente de aprendizagem ou *learning rate*; e γ é o factor de desconto, também um valor entre $[0, 1]$.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big] \tag{3.1}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big] \tag{3.2}$$

Para o tipo de aprendizagem pretendido, para testar o SALbot, foi necessário adequar o algoritmo ao robô. Ou seja, uma vez que o objectivo é fazer o robô circular por um ambiente sem colidir, o robô não tem um estado terminal e cada passo corresponde a um episódio. Utilizaram-se os parâmetros de aprendizagem mais comuns para este tipo de algoritmo: $\varepsilon = 0.5; \alpha = 0.5; \gamma = 0.3$. Em que ε (factor de exploração) corresponde à probabilidade de escolher uma acção aleatória.

3.4.2 Acções e estados

Foi necessário, ainda, discretizar as acções e estados possíveis para o SALbot. Tendo em consideração o objectivo da aprendizagem (percorrer um ambiente sem colidir), foram consideradas seis acções distintas: andar em frente (acção 0); curvar para a direita (acção 1); curvar para a esquerda (acção 2); rodar 90° para a direita (acção 3); rodar 90° para a esquerda (acção 4) e andar para trás (acção 5). As acções foram configuradas em passos de duração, que variam consoante a acção. Esses passos de duração correspondem ao envio do sinal PW n vezes.

Quanto aos estados, cada estado é definido consoante as medições dos sensores de distância e contacto. Foram definidos quatro níveis de distância para os sensores de distância por infra-vermelhos laterais e frontal, e três níveis de distância para os sensores de distância por infra-vermelhos traseiro e ultra-sons frontal. Nos sensores de contacto, utilizaram-se quatro *bits* em que cada *bit* corresponde a uma zonas distinta: frente, lateral esquerda, lateral direita e traseira; a que correspondem 16 estados diferentes. Assim obtemos $4 \times 4 \times 4 \times 3 \times 3 \times 16 = 9216$ estados possíveis. Embora a maioria dos estados previstos seja muito improvável de acontecer, como por exemplo o SALbot colidir nos quatro lados e estar longe de um objecto de todos os lados, os estados têm de ser contemplados, caso contrário estaríamos, de certa forma, a indicar ao robô que nunca iria aprender aqueles estados. Para que a combinação dos sensores corresponda a um estado, concebeu-se uma função (equação 3.3) que combina os estados de cada sensor num estado único.

$$Estado = a + Ab + ABc + AB\Gamma d + AB\Gamma\Delta e + AB\Gamma\Delta Ef \tag{3.3}$$

Sensor	Nível	Número de combinações
pára-choques	a	A
sensor de ultra-sons frontal	b	B
sensor de infra-vermelhos frontal	c	Γ
sensor de infra-vermelhos traseiro	d	Δ
sensor de infra-vermelhos lateral esquerdo	e	E
sensor de infra-vermelhos lateral direito	f	Z

Para o SALbot receber uma recompensa pela acção executada existe uma função que calcula a recompensa. A recompensa depende se o SALbot colidiu ou não, caso ocorra uma colisão a recompensa é de -10 por cada sensor de contacto activado. Se não existir nenhuma colisão, a recompensa é a soma das recompensas para cada sensor, ou seja, cada sensor possui uma recompensa para cada nível:

Nível	sensor de ultra-sons frontal	sensor de infra-vermelhos frontal	sensor de infra-vermelhos traseiro	sensor de infra-vermelhos lateral esquerdo	sensor de infra-vermelhos lateral direito
0	$+2$	$+4$	$+3$	0	0
1	$+1$	$+3$	$+2$	$+2$	$+2$
2	0	$+1$	-3	-1	-1
3	$-$	-3	$-$	-3	-3

A recompensa total é a soma da recompensa do nível de cada sensor.

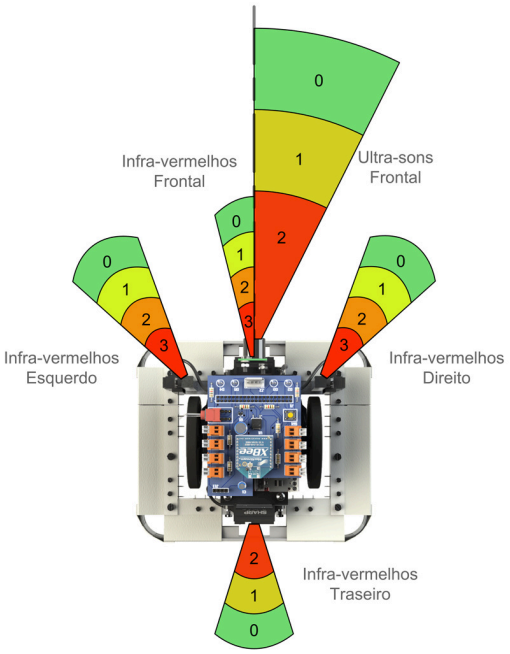


Figura 3.23: Representação dos níveis dos sensores de distância.

Os níveis de cada sensor de distância (representados na figura 3.23) foram definidos tendo em conta as acções previstas (figura 3.24) e as limitações dos sensores. Os níveis máximos correspondem a objectos mais próximos e os níveis mínimos objectos mais distantes. Foram retirados os valores lidos dos sensores de distância por infra-vermelhos para a situação em que o robô se encontra com obstáculos encostados ao pára-choques para definir o nível máximo. Isto foi feito porque os sensores se encontram a menos de 30 mm da face dos pára-choques e eles não podem detectar obstáculos a distâncias menores de 30 mm.

Os níveis médios dos sensores de distância por infra-vermelhos laterais e frontal, bem como o nível mínimo do sensor traseiro, foram definidos pelos valores retirados dos sensores para a detecção de obstáculos tangentes ao círculo, que corresponde à rotação do SALbot, e paralelos aos lados do robô. Para a definição dos níveis mínimos dos sensores de distância por infra-vermelhos laterais e frontal, mediram-se os valores retirados destes para a detecção de obstáculos a uma distância de: 190 mm para a frente e 160 mm para os lados.

No caso do sensor de distância por ultra-sons, o nível máximo foi definido para o valor que representa a distância de 150 mm, uma vez que o sensor envia sempre este valor para objectos abaixo desta distância. O nível mínimo foi definido para uma distância aproximada a 300 mm.

Assim, os níveis dos sensores foram definidos para a seguinte gama de valores:

Nível	sensor de ultra-sons frontal	sensor de infra-vermelhos frontal	sensor de infra-vermelhos traseiro	sensor de infra-vermelhos lateral esquerdo	sensor de infra-vermelhos lateral direito
0	≥ 24	< 240	< 390	< 200	< 200
1	$[12, 24[$	$[240, 390[$	$[390, 400[$	$[200, 350[$	$[200, 355[$
2	< 12	$[390, 400[$	≥ 400	$[350, 450[$	$[355, 460[$
3	–	≥ 400	–	≥ 450	≥ 460

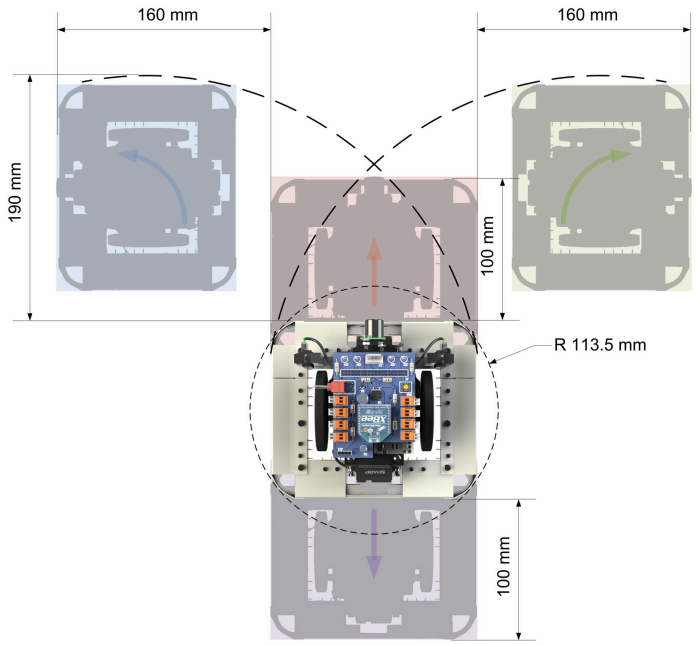


Figura 3.24: Representação das várias acções do SALbot.

O SALbot guarda o valor de Q de cada par de estado-acção numa tabela designada de tabela de aprendizagem (tabela 3.1). Nessa tabela, as acções estão dispostas ao longo das colunas e os estados ao longo das linhas. Embora se fale de uma tabela de aprendizagem, o SALbot guarda os valores em endereços da EEPROM externa. Surgiu assim a necessidade de transformar as duas dimensões da tabela numa dimensão a que corresponde um endereço. Semelhantemente ao que é feito para a definição de estado, multiplica-se o valor do estado pelo número de acções e somando depois o valor da acção, obtendo-se assim um endereço único a que corresponde um valor de Q para um determinado par de estado-acção.

Tabela 3.1: Tabela de aprendizagem do SALbot.

Estado \ Acção	a_0	a_1	$\cdots$	a_i	$\cdots$	a_5
s_0	$Q(s_0, a_0)$	$Q(s_0, a_1)$	$\cdots$	$Q(s_0, a_i)$	$\cdots$	$Q(s_0, a_5)$
s_1	$Q(s_1, a_0)$	$Q(s_1, a_1)$	$\cdots$	$Q(s_1, a_i)$	$\cdots$	$Q(s_1, a_5)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\vdots$
s_j	$\cdots$	$\cdots$	$\cdots$	$Q(s_j, a_i)$		$\vdots$
$\vdots$	$\vdots$	$\vdots$			$\ddots$	$\vdots$
s_{9215}	$Q(s_{9215}, a_0)$	$Q(s_{9215}, a_1)$	$\cdots$	$\cdots$	$\cdots$	$Q(s_{9215}, a_5)$

3.4.3 Gerador de números aleatórios

Para o método de aprendizagem implementado, é necessário gerar números aleatórios: para escolher acções aleatórias; e para escolher entre procurar na tabela de aprendizagem a acção, a que corresponde o maior valor de Q para o estado actual, ou gerar uma acção aleatória.

Uma vez que o microcontrolador não possui um contador de tempo real e para evitar usar uma função pseudo-aleatória (que devolve valores aleatórios com uma sequência predefinida), usou-se uma média dos valores lidos dos sensores de distância, assim que o SALbot é iniciado, para usar como "semente" da função aleatória. Assim, com essa "semente" a função pseudo-aleatória gera valores aleatórios com uma sequência que depende do valor da "semente". Isto é, para "sementes" diferentes a sequência de números aleatórios é completamente diferente.

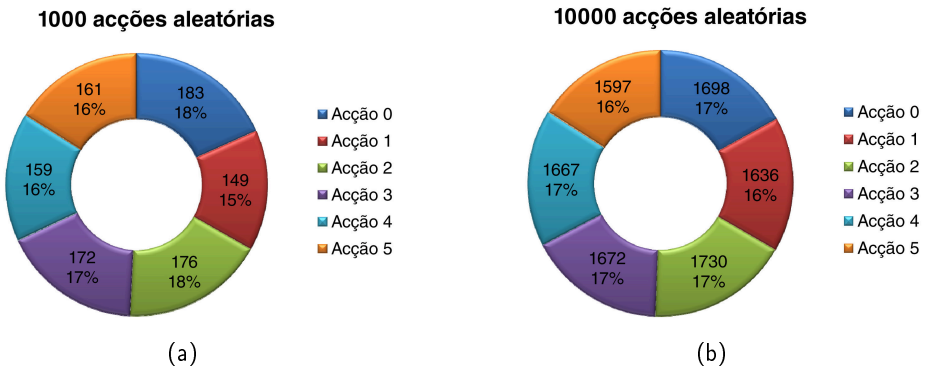


Figura 3.25: Geração de: (a) 1000 acções aleatórias; (b) 10000 acções aleatórias.

Analisando a figura 3.25, é possível verificar que as percentagens das acções escolhidas aleatoriamente são muito semelhantes entre elas. Isto é, a probabilidade de escolher uma determinada acção é igual para todas.

3.4.4 Modos de funcionamento

A aprendizagem começa pelo estado de Inicialização da Aprendizagem (figura 3.27b). Nesse estado, primeiro são inicializados os parâmetros e valores de aprendizagem, depois o robô verifica o estado em que se encontra, e escolhe aleatoriamente uma acção; no final, o modo de operação do SALbot é automaticamente alterado para Aprendizagem. Na inicialização dos parâmetros e valores de aprendizagem, primeiro são repostos os parâmetros ε, α e γ, a contagem do número de iterações e de colisões é colocada a zero e a tabela de aprendizagem é apagada e colocada a zeros. Não é possível colocar o SALbot no modo de Aprendizagem sem que este tenha passado anteriormente pelo estado de Inicialização da Aprendizagem.

Já no modo de Aprendizagem (figura 3.28), é iniciado um ciclo que só termina quando o SALbot mudar de modo de operação. Nesse ciclo o SALbot começa por executar a acção previamente escolhida, verifica o estado após a execução da acção, calcula o valor da recompensa pela acção executada, escolhe uma nova acção (aleatória ou a acção com maior valor de Q para o estado em que se encontra), calcula o valor de Q para o par de estado-acção anterior e volta ao início do ciclo.

Foram definidos cinco estados de operação diferentes para o SALbot: o *Idle*, Autónomo, Manual, Inicialização da Aprendizagem e Aprendizagem. Estes estados de operação correspondem a diferentes modos de execução do SALbot. O estado *Idle* faz com que o robô fique em repouso e

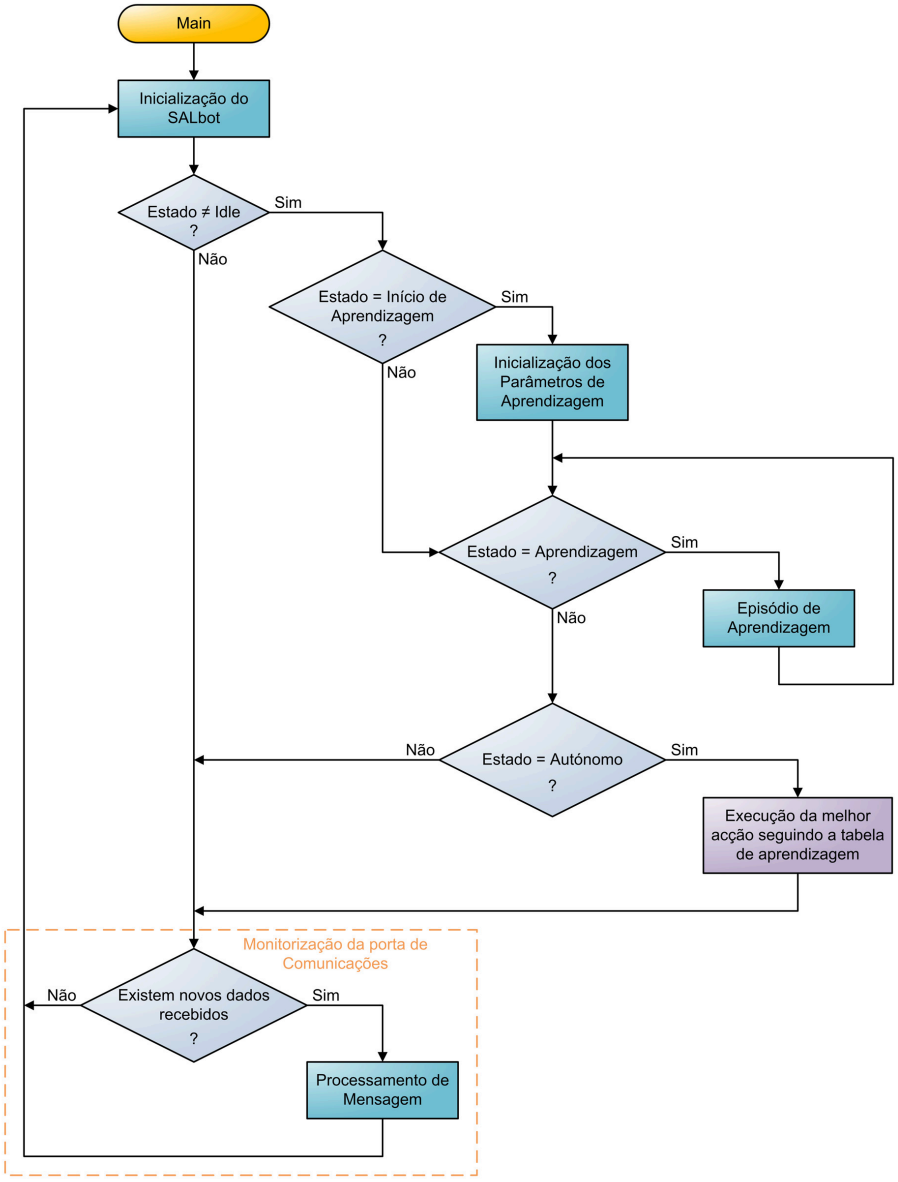
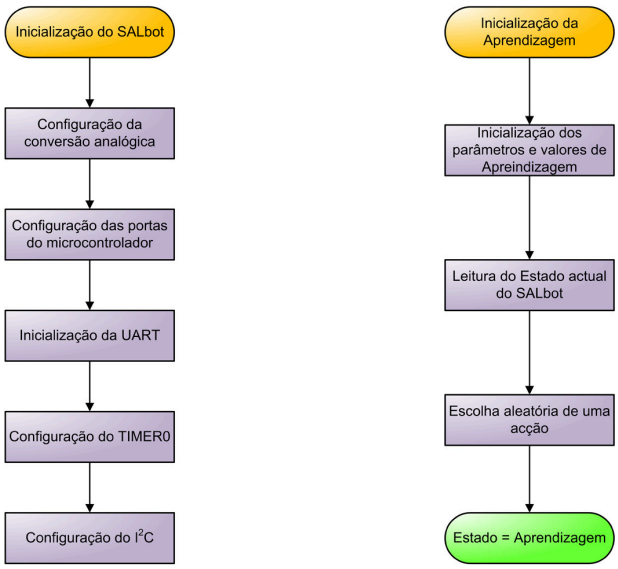


Figura 3.26: Fluxograma da rotina *Main*.

apenas verifique se recebeu algum comando. No estado Autónomo, o robô executa acções perante os estados em que se encontra, procurando na tabela de aprendizagem a acção que maior valor tem para o estado em que o robô se encontra. O modo de Inicialização da Aprendizagem apenas tem a finalidade de inicializar os parâmetros de aprendizagem para iniciar uma nova aprendizagem. Após a execução das tarefas associadas ao modo de Inicialização da Aprendizagem, o estado do SALbot é automaticamente mudado para Aprendizagem. Enquanto o robô se mantiver no estado de Aprendizagem são gerados episódios de aprendizagem.

A figura 3.26 mostra o fluxograma da rotina *Main* do SALbot, onde é possível verificar as funções de cada estado.



(a) (b)

Figura 3.27: Fluxograma das rotinas: (a) Inicialização do SALbot; (b) Inicialização da Aprendizagem.

A inicialização do SALbot e da aprendizagem (figura 3.27) são rotinas que configuram e parametrizam funções e variáveis.

Na inicialização do SALbot, são configuradas as portas, digitais e analógicas, do microcontrolador, a porta série, o temporizador, utilizado para controlar os sinais PW que comandam os servos, e os parâmetros da comunicação I^2C™.

A rotina de inicialização da aprendizagem serve para anular a tabela de aprendizagem, repor os parâmetros de aprendizagem nos valores originais e iniciar uma aprendizagem, observando o estado actual do robô e escolhendo uma acção aleatória.

No episódio de aprendizagem (figura 3.28) é executado uma iteração do algoritmo SARSA. Para que seja possível ao SALbot sair do modo de aprendizagem, ou executar outras tarefas, após a execução da acção, o SALbot vai verificar se existem dados recebidos pela porta série. Após o processamento das mensagens, ou caso não existam mensagens, é continuada a iteração do algoritmo de aprendizagem.

A rotina de processamento de mensagens (figura 3.29), verifica se os dados recebidos pela porta série correspondem a uma mensagem válida, segundo o protocolo implementado para a comunicação entre os robôs e a interface no computador. Depois, caso seja recebido um comando, este é executado.

s

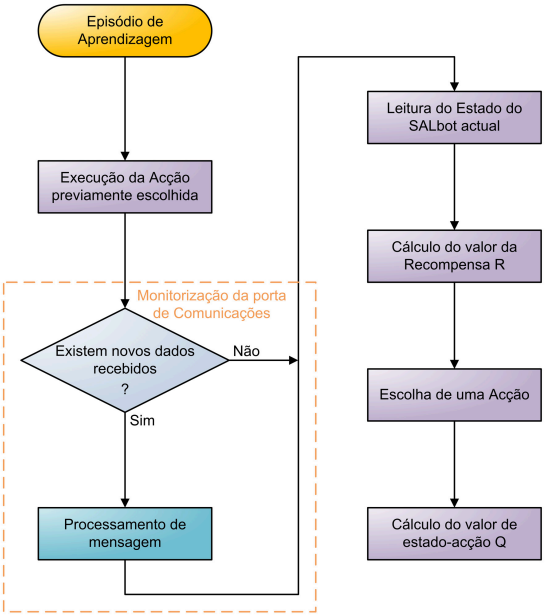


Figura 3.28: Fluxograma da rotina de Aprendizagem.

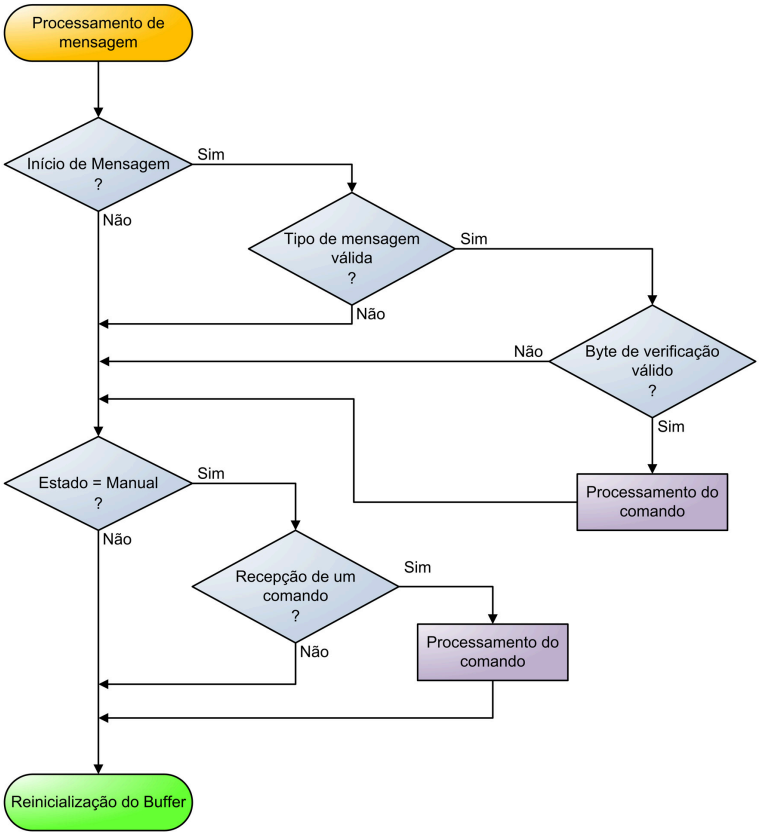


Figura 3.29: Fluxograma da rotina de Processamento de mensagens.

3.5 Interface no PC

A interface gráfica (figuras 3.30 e 3.31) no computador remoto foi concebida com o fim de se poder comandar e monitorar os dois SALbot. Através da interface, é possível escolher o modo de execução de cada um dos SALbot. No modo Manual, é possível pedir dados relativos aos sensores de distância e de contacto ao SALbot, bem como carregar e descarregar tabelas de aprendizagem do robô. Em qualquer modo, é possível abrir e guardar ficheiros relativos à progressão de aprendizagem (gráfico) e tabela de aprendizagem. Estes ficheiros são do tipo csv (*comma separated values*) para que seja possível visioná-los e interpretá-los num programa de folhas de cálculo (como, por exemplo, o Excel), já que a interface não mostra a tabela de aprendizagem.

Na interface existem dois ambientes: o do SALbot1 e SALbot2; dentro de cada um desse ambientes existem mais dois: Sensores e Aprendizagem. No ambiente Sensores de cada um dos SALbot é possível monitorar as distâncias, valores e nível dos sensores de distância, o estado do pára-choques o tipo de movimento do robô e ainda mudar o modo de operação do robô para Manual, Autónomo ou *Idle*. Os modos de Inicialização de Aprendizagem e Aprendizagem, apenas podem ser escolhidos no ambiente Aprendizagem. Nesse ambiente, também é possível monitorar o progresso de aprendizagem, bem como as operações relativas aos dados da aprendizagem do robô já descritas, como carregar e descarregar a tabela de aprendizagem do SALbot.

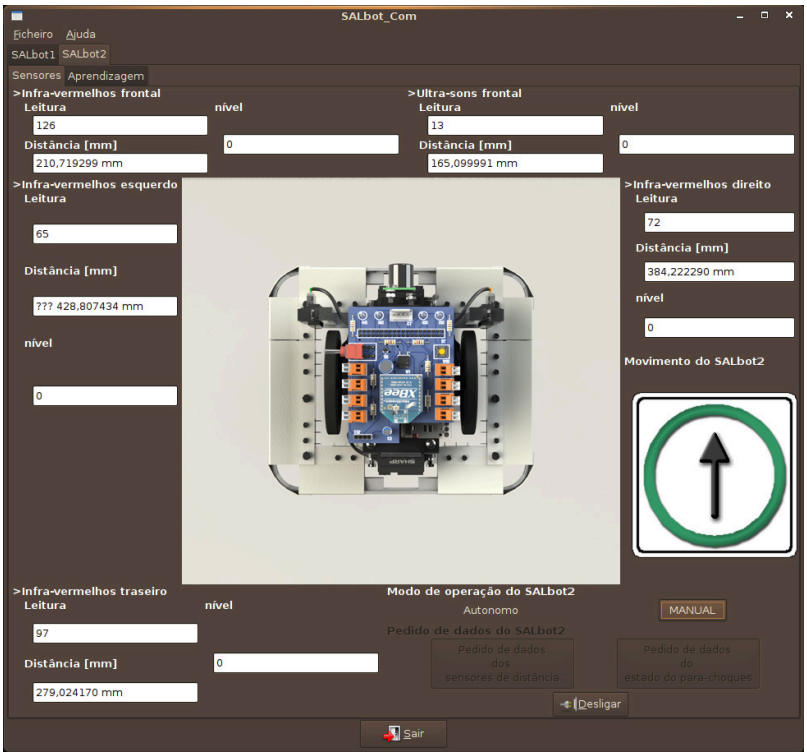


Figura 3.30: Secção, da interface gráfica, de monitorização dos sensores e de comando do modo de operação do SALbot2.

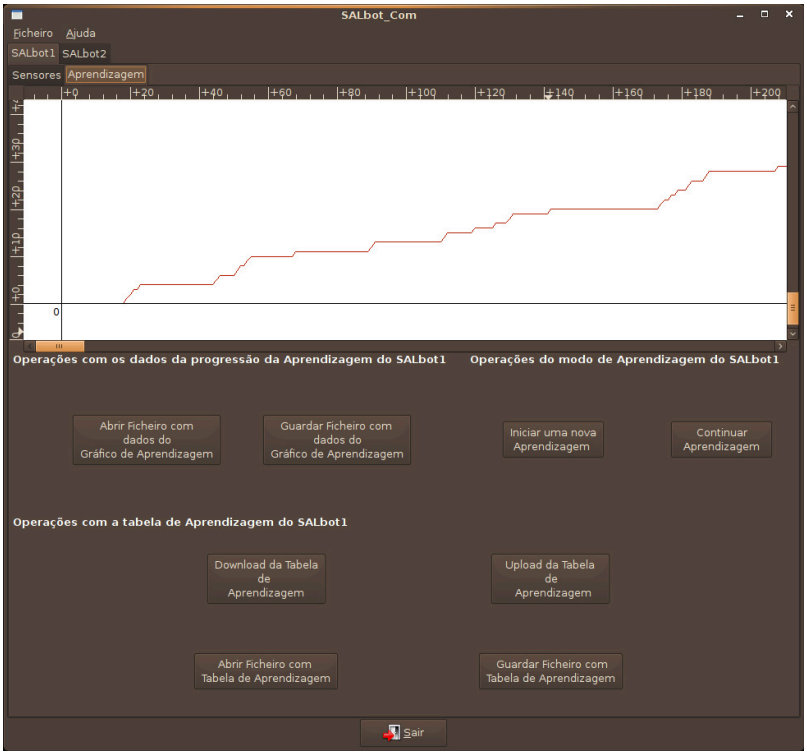


Figura 3.31: Secção, da interface gráfica, de monitorização e comando da aprendizagem do SALbot1.

Capítulo 4

Testes e Resultados

"É durante as tempestades que o verdadeiro marinheiro aprende a velejar."

Ricardo Jordão Magalhães

Com a finalidade de validar o SALbot como um robô habilitado para aprendizagem, foram feitos testes com um algoritmo de aprendizagem básico. Estes testes foram realizados num ambiente restrito, onde o robô aprende a escolher o melhor movimento para os vários estados que vai visitando.

Neste capítulo é feita uma abordagem ao ambiente de testes usado e são expostos e analisados os resultados dos testes provenientes da aprendizagem dos SALbot.

4.1 Ambiente de testes

Para testar a aprendizagem dos SALbot, foi construído um ambiente de testes (figura 4.1) onde decorreram os ensaios, dos quais, se extraíram os dados relativos à aprendizagem, examinados na secção de resultados (4.2). Uma vez que o objectivo principal da aprendizagem consiste no SALbot percorrer um ambiente sem colidir, optou-se por ambiente com a forma de um corredor: com 2 m de comprimento por 0.65 m de largura e com uma altura de 150 mm.

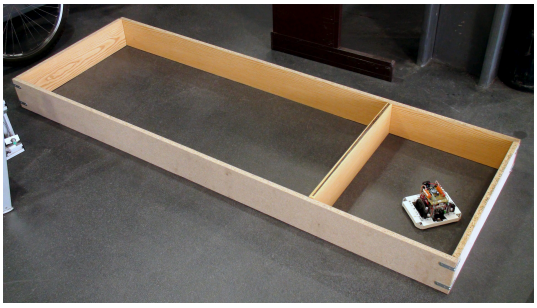


Figura 4.1: Ambiente de testes dos SALbot.

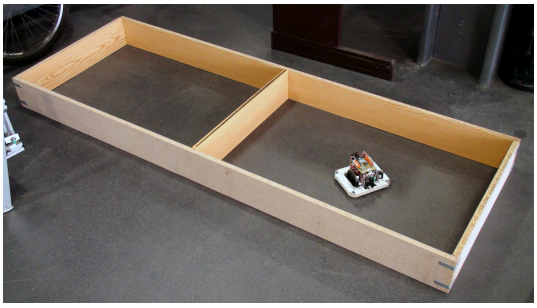
É possível variar a dimensão do ambiente de testes, colocando uma tábua, transversalmente, ao longo do corredor. Ao variarmos a dimensão do ambiente, permitimos ao SALbot aprender novos estados, pois se o ambiente for sempre muito grande o robô nunca irá aprender a movimentar-se em espaços pequenos e vice-versa. Além de que, se o ambiente for muito grande, o SALbot necessita de mais iterações para aprender todos os estados possíveis, comparativamente a ambientes mais pequenos.

4.2 Resultados

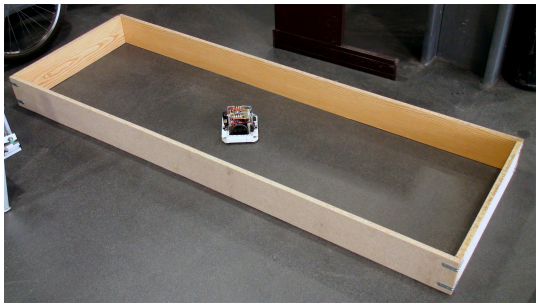
Uma vez que, o algoritmo de aprendizagem do SALbot tem um parâmetro $\varepsilon = 0.5$ (ver secção 3.4.1), mesmo que já numa fase avançada de aprendizagem, a probabilidade de o SALbot escolher uma acção aleatória, para um estado que já aprendeu, é de 50%. Se, por um lado, isso garante que continua a experimentar outras acções para estados já conhecidos, por outro, dificulta a percepção da evolução da aprendizagem do robô.



(a)



(b)



(c)

Figura 4.2: (a) 1/4 do ambiente de testes; (b) 1/2 do ambiente de testes; (c) Ambiente de testes completo.

Assim, optou-se por avaliar a aprendizagem dos SALbot de formas diferentes:

i) Após 500, 1000, 2000 e 5000 iterações (cada iteração corresponde ao SALbot procurar na tabela qual a melhor acção a executar) foram descarregadas as tabelas de aprendizagem para comparação entre os diversos testes realizados;

ii) Escolheram-se alguns estados das diferentes tabelas de aprendizagem de um teste onde se compara o comportamento do SALbot com o que seria de esperar da sua aprendizagem para esses estados;

iii) Ao longo da aprendizagem, é actualizado um gráfico na interface gráfica no computador remoto, que mostra o número de colisões ao longo das iterações em que o SALbot foi procurar à tabela de aprendizagem a melhor acção aprendida para o estado em que se encontra.

Os sensores de distância por infra-vermelhos laterais foram orientados a $\approx 40^{\circ}$ do plano lateral do robô. Uma vez que a maioria das acções são orientadas para a frente, esta configuração permite a detecção de objectos laterais e frontais.

Assim, realizaram-se três testes de aprendizagem distintos: para um ambiente pequeno, em $1/4$ do ambiente total (figura 4.2a); para um ambiente médio, em $1/2$ do ambiente total (figura 4.2b); e para o ambiente completo (figura 4.2c).

4.2.1 Teste de aprendizagem para o ambiente pequeno

Este teste de aprendizagem foi realizado em $1/4$ do ambiente de testes, dos quais foram retiradas as tabelas de aprendizagem após 500, 1000, 2000 e 5000 iterações.

Ao fim de 5000 iterações o SALbot visitou 345 estados diferentes. Entre as 500 e 1000 iterações, foram mantidos os valores de Q para 46 estados e foram aprendidos 52 novos estados. Entre as 1000 e 2000 iterações, 65 estados mantiveram-se iguais e foram visitados 53 novos estados. Por fim, entre as 2000 e 5000 iterações, a tabela manteve-se igual para 67 estados e 100 novos estados foram visitados.

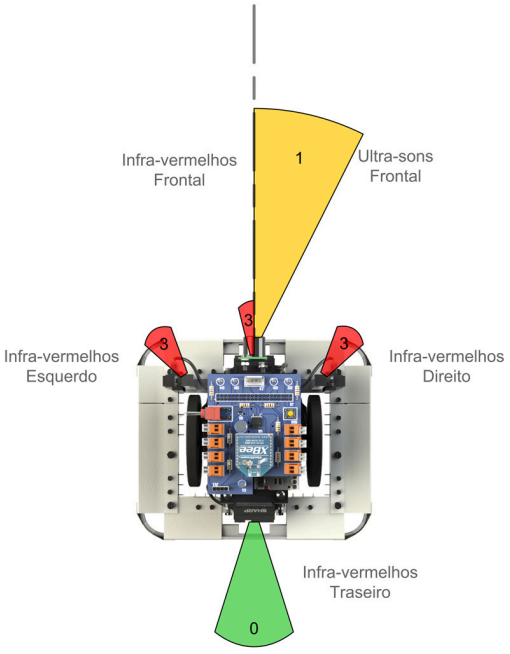


Figura 4.3: Representação do estado 8800 do SALbot.

O estado 8800, como se pode verificar na figura 4.3, corresponde a: nível 3 nos sensores de distância por infra-vermelhos frontal e laterais; nível 0 no sensor de distância traseiro; nível 1 no sensor de distância por ultra-sons frontal; e sem nenhum sensor de contacto do pára-choques activo. Este estado não seria de esperar que fosse visitado, uma vez que existe uma disparidade entre os

sensores de distância por infra-vermelhos e ultra-sons frontais. No entanto, isto pode dever-se ao facto de haver reflexão múltipla (*multipath*) do eco, do feixe de ultra-sons.

Tabela 4.1: Evolução da tabela de aprendizagem para o estado 8800 do teste para 1/4 do ambiente.

Estado 8800						
Acção / Iteração	0	1	2	3	4	5
500	0	0	0	0	−6	0
1000	0	0	0	0	−6	9
2000	0	0	0	−5	−2	14
5000	−10	0	0	−5	−3	15

Para este estado (8800) é de esperar que a acção de andar para trás (acção 5) seja a que maior recompensa dê ao SALbot, tendo em conta a função de recompensas. Analisando a evolução da tabela de aprendizagem para este estado (tabela 4.1), facilmente se verifica que o SALbot aprendeu que a melhor acção que tem de executar, quando se encontra nesse estado, é de facto andar para trás (acção 5). No entanto, é também perceptível que, antes das 500 iterações, o SALbot ainda não tinha visitado este estado um número de vezes suficientes para que testasse todas as acções, e nem mesmo após as 5000 iterações foi possível ao SALbot testar todas as acções. Ainda assim, o SALbot, depois da aprendizagem, tem informação suficiente para evitar a colisão quando se encontra neste estado, informação que antes da aprendizagem não possuia.

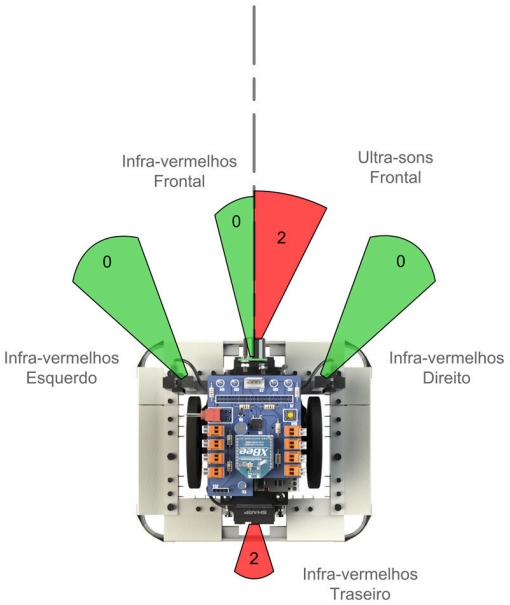


Figura 4.4: Representação do estado 416 do SALbot.

Outro estado interessante é o 416, que como se pode ver na figura 4.4 corresponde a: nível 0 para os sensores de distância por infra-vermelhos frontal e laterais; nível 2 para o sensor de distância

por infra-vermelhos traseiro; nível 2 para o sensor de distância por ultra-sons frontal; e sem nenhum sensor de contacto do pára-choques activo. Neste estado espera-se que o robô escolha a acção de andar para a frente (acção 0).

Tabela 4.2: Evolução da tabela de aprendizagem para o estado 416 do teste para 1/4 do ambiente.

Estado 416						
Acção / Iteração	0	1	2	3	4	5
500	7	8	0	0	0	−6
1000	7	10	0	0	0	−6
2000	13	7	0	5	0	−6
5000	13	10	8	5	−3	−2

Fazendo uma análise à tabela 4.2, onde estão expostos os valores da tabela de aprendizagem para este estado, após as várias iterações, verifica-se que ao final das 5000 iterações o SALbot aprendeu que a melhor acção que deve escolher é andar para a frente (acção 0) e tinha experimentado todas as acções possíveis. No entanto, esta acção já tinha sido experimentada desde o final das 500 iterações, não sendo, no entanto, a que melhor recompensa oferecia até ao final das 2000 iterações. É também perceptível uma instabilidade da evolução da tabela para a acção de curvar para a direita (acção 1), que aumentou das 500 para as 1000 iterações, diminuindo das 1000 para as 2000 e voltando a aumentar das 2000 para as 5000. Isto pode dever-se ao SALbot, por vezes, colidir nas paredes e outras vezes não, isto é, o estado representativo do SALbot em relação ao ambiente é o mesmo para ambas as situações, mas o estado real não. Outro facto interessante é o aumento do valor de Q para a acção de andar para trás (acção 5) das 2000 para as 5000 iterações, embora permaneça negativo. Neste caso, uma das causas pode ser a diminuição da carga das baterias dos servos, o que implica que estes rodem mais devagar, o que faz com que o SALbot não percorra o espaço todo, que normalmente percorreria, evitando assim a colisão.

O outro estado escolhido deste teste para análise é o 6402. Como se pode verificar na figura 4.5 este estado corresponde a: nível 1 nos sensores de distância por infra-vermelhos e ultra-sons frontais; nível 2 no sensor de distância por infra-vermelhos lateral direito; nível 3 no sensor de distância por infra-vermelhos lateral esquerdo; nível 0 no sensor de distância por infra-vermelhos traseiro; e um ou mais sensores de contacto da parte lateral esquerda do pára-choques activo. Para este estado esperas-se que a acção com maiores recompensas seja andar para trás (acção 5).

Tabela 4.3: Evolução da tabela de aprendizagem para o estado 6402 do teste para 1/4 do ambiente.

Estado 6402						
Acção / Iteração	0	1	2	3	4	5
500	0	−11	−6	−11	0	0
1000	0	−11	−6	−11	0	5
2000	0	−4	−6	1	0	11
5000	−9	−7	−8	−5	−11	11

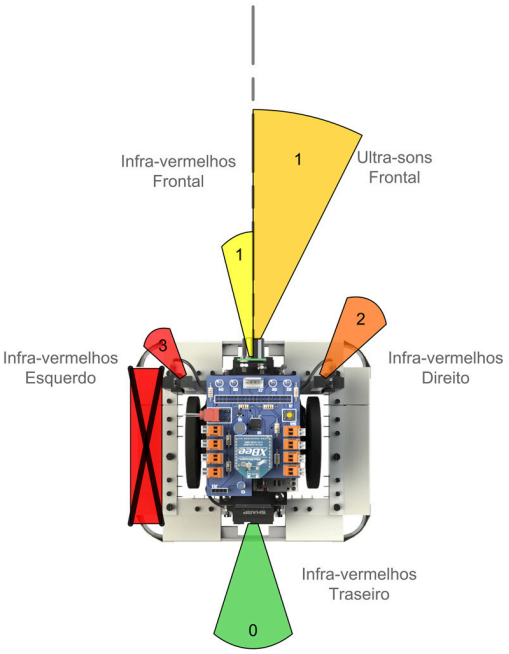


Figura 4.5: Representação do estado 6402 do SALbot.

Analisando a tabela 4.3, verifica-se que todas as acções foram experimentadas pelo SALbot após 5000 iterações e que a acção que dá mais recompensas é, como seria de esperar, a acção de andar para trás (acção 5). De facto, esta acção é a única com o valor de Q positivo após as 5000 iterações. Como acontece no caso do estado 416, também neste estado existe uma inconsistência no valor de Q para a acção de curvar para a direita (acção 1), mas o facto mais interessante é a variação do valor da tabela de aprendizagem para a acção de rodar para a direita (acção 3) entre as 1000 e 5000 iterações. Para esta acção, o valor de Q aumentou das 1000 iterações para as 2000, passando de um valor grande negativo para um pequeno positivo, diminuindo após as 5000 iterações, assumindo, de novo, um valor negativo. Isto pode ser explicado da mesma forma que o caso da acção de curvar para a direita (acção 1) do estado 416, sendo que entre as 1000 e 2000 iterações o SALbot executou vária vezes a acção de rodar para direita que resultaram em boas recompensas.

4.2.2 Teste de aprendizagem para o ambiente médio

Este teste foi realizado em $1/2$ do ambiente de testes, do qual se retiraram as tabelas de aprendizagem após 500, 1000, 2000 e 5000 iterações, tal como no teste anterior.

Neste teste o SALbot após 5000 iterações tinha visitado 297 estados diferentes. Das 500 para as 1000 iterações foram visitados 53 estados novos e os valores da tabela de aprendizagem de 34 estados mantiveram-se iguais, das 1000 para as 2000 o SALbot visitou 66 novos estados e manteve igual os valores da tabela de aprendizagem em 55 estados e das 2000 para as 5000 iterações o SALbot aprendeu em 66 novos estados e os valores de Q da tabela de aprendizagem manteve-se inalterado para 51 estados.

Das tabelas de aprendizagem obtidas deste teste escolheram-se três estados para serem analisados: o 592; o 1809; e o 7556;

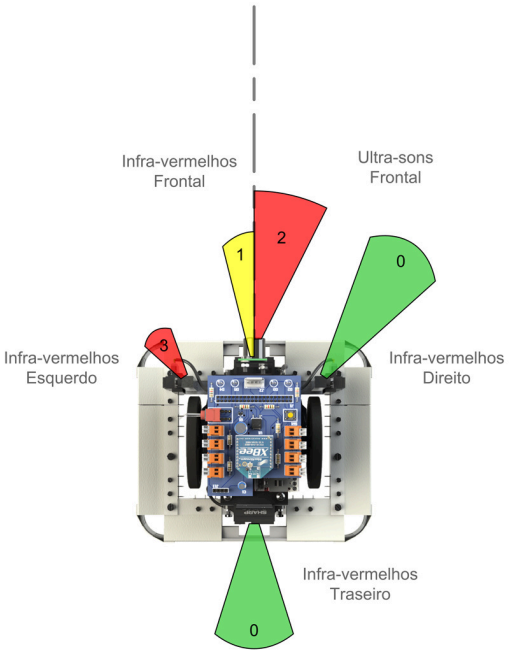


Figura 4.6: Representação do estado 1809 do SALbot.

O estado 1809, representado na figura 4.6, corresponde a: nível 1 no sensor de distância por infra-vermelhos frontal; nível 2 no sensor de distância por ultra-sons frontal; nível 3 no sensor de distância por infra-vermelhos lateral esquerdo; nível 0 nos sensores de distância por infra-vermelhos lateral direito e traseiro; e nenhum sensor de contacto do sistema de pára-choques activo. Tendo em conta os valores dos sensores a que corresponde o estado, é de esperar que o SALbot aprenda que a acção que garante maiores recompensas é a acção de andar para trás (acção 5).

Tabela 4.4: Evolução da tabela de aprendizagem para o estado 1809 do teste para $1/2$ do ambiente.

Estado 1809						
Acção / Iteração	0	1	2	3	4	5
500	0	0	0	0	0	0
1000	0	0	0	0	0	0
2000	0	0	−11	0	0	−6
5000	−10	−5	−11	−10	−4	−9

Analisando a evolução da tabela de aprendizagem para este estado (tabela 4.4), verifica-se que não só a melhor acção não é a que seria esperada, como também nenhuma acção é boa. Isto é, após 5000 iterações o SALbot experimentou todas as acções possíveis e nenhuma delas garante boas recompensas. Desta análise também se verifica que este estado foi visitado pela primeira vez entre as 1000 e 2000 iterações. Isto pode dever-se ao facto de o ambiente de testes ser maior do que no teste anterior, o que implica um maior número de iterações para visitar mais vezes um estado.

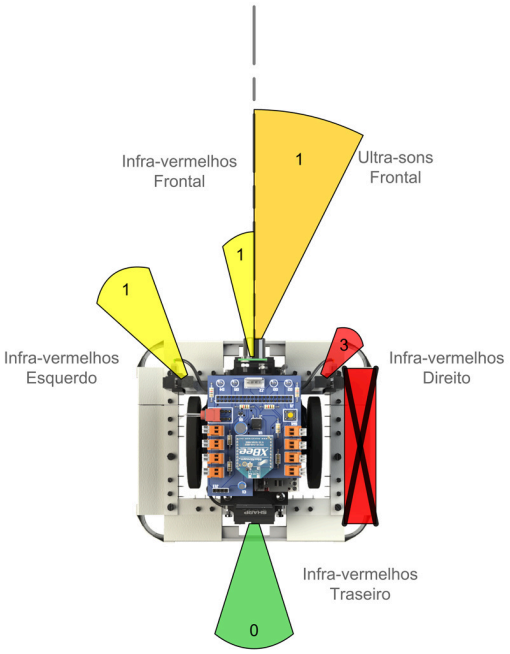


Figura 4.7: Representação do estado 7556 do SALbot.

Para o estado 7556 é de esperar que o SALbot aprenda que a acção que garante maiores recompensas seja a acção de andar para trás (acção 5). Este estado, representado na figura 4.7, corresponde a: nível 1 nos sensores de distância por infra-vermelhos frontal e lateral esquerdo e por ultra-sons frontal; nível 3 no sensor de distância por infra-vermelhos lateral direito; nível 0 no sensor de distância por infra-vermelhos traseiro; e pelo menos um dos sensores de contacto do lado direito do sistema de pára-choques activo.

Tabela 4.5: Evolução da tabela de aprendizagem para o estado 7556 do teste para $1/2$ do ambiente.

Estado 7556						
Iteração \ Acção	0	1	2	3	4	5
500	0	0	0	0	−6	4
1000	0	0	1	−6	−1	10
2000	−11	−5	1	−5	−5	11
5000	−10	−2	1	−3	−3	13

Da análise à evolução da tabela de aprendizagem entre as 500 e 5000 iterações (tabela 4.5), verifica-se que ao final das 5000 iterações o SALbot aprendeu que a acção que garante maiores recompensas para este estado é andar para trás (acção 5), como era de esperar, em que o valor de Q para essa acção foi sendo reforçado ao longo das iterações. Além disso, todas as acções foram experimentadas ao final das 2000 iterações. No entanto, existe um resultado curioso, para a acção de rodar para a direita (acção 3), que é o lado da colisão, a partir das 1000 iterações até às 5000 o valor de Q vai aumentando, embora permaneça negativo. Isto significa que existem casos em

que o SALbot, estando nesse estado e executando essa acção, adquire boas recompensas, o que não seria de esperar. Porém, isto pode-se dever ao facto de, por exemplo, o SALbot se encontrar perto de um dos cantos do ambiente de testes colidindo na parte traseira da lateral esquerda e de, devido ao estado da bateria dos servos, não rodar os 90° completos para a esquerda.

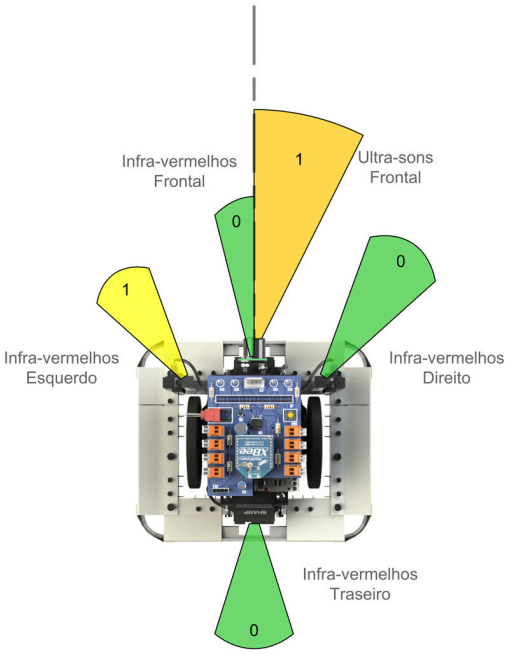


Figura 4.8: Representação do estado 592 do SALbot.

O outro estado escolhido deste teste para ser analisado foi o 592, representado na figura 4.8. Este estado corresponde a: nível 1 nos sensores de distância por infra-vermelhos lateral esquerdo e ultra-sons frontal; nível 0 nos sensores de distância por infra-vermelhos frontal, traseiro e lateral direito; e nenhum sensor de contacto do sistema de pára-choques activo. Para este estado espera-se que o SALbot aprenda que a melhor acção a executar seja andar para a frente (acção 0).

Tabela 4.6: Evolução da tabela de aprendizagem para o estado 592 do teste para 1/2 do ambiente.

Estado 592						
Iteração \ Acção	0	1	2	3	4	5
500	14	9	−11	11	12	13
1000	14	9	−2	12	12	12
2000	13	2	−4	9	12	12
5000	13	3	−2	9	14	12

Fazendo uma análise à progressão dos valores da tabela de aprendizagem (tabela 4.6), de facto, a acção de andar para a frente (acção 0) garante boas recompensas, no entanto, entre as 2000 e as 5000 iterações, esta acção deixou de ser a que maiores recompensas garante ao SALbot, em que a acção dominante passou a ser rodar para a esquerda (acção 4). Outro facto retirado da análise, é

que já no final das 500 iterações o SALbot tinha experimentado todas as acções disponíveis neste estado, pelo que se pode concluir que este estado foi vezes visitado muitas vezes neste teste.

4.2.3 Teste de aprendizagem para o ambiente completo

Neste teste, a aprendizagem ocorreu na totalidade do espaço do ambiente de testes.

No final do teste, isto é, após as 5000 iterações o SALbot visitou, pelo menos uma vez, 255 estados diferentes, dos quais 30 novos estados foram visitados entre as 500 e 1000 iterações, 52 entre as 1000 e 2000 iterações e 77 entre as 2000 e 5000 iterações. Os valores da tabela de aprendizagem mantiveram-se iguais para: 30 estados entre as 500 e 1000 iterações; 36 estados entre as 1000 e 2000 iterações; e 40 estados entre as 2000 e 5000 iterações.

Tal como nos testes anteriores, foram escolhidos três estados indicativos para analisar a evolução da tabela de aprendizagem: o 64; o 2368; e o 5921.

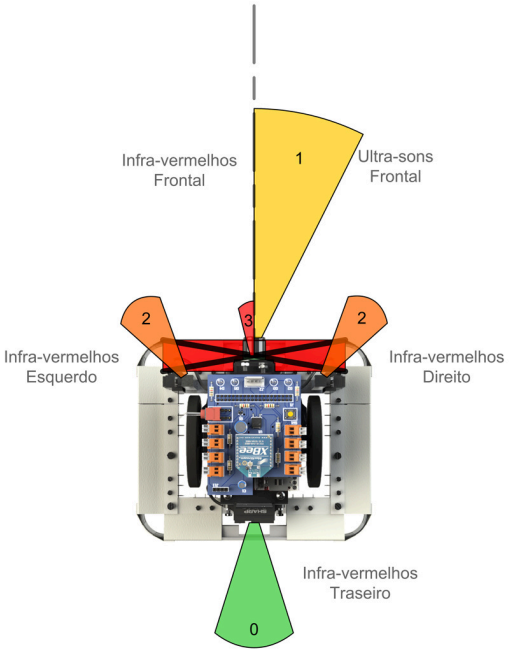


Figura 4.9: Representação do estado 5921 do SALbot.

O estado 5921, representado na figura 4.9, corresponde a: nível 1 no sensor de distância por ultra-sons frontal; nível 3 no sensor de distância por infra-vermelho frontal; nível 2 para os sensores de distância por infra-vermelhos laterais direito e esquerdo; nível 0 para o sensor de distância por infra-vermelhos traseiro; e com um dos sensores de contacto da parte frontal do sistema de pára-choques activo. Tal como no estado 8800, também este não seria de esperar que fosse visitado, devido à grande diferença entre níveis dos sensores de distância frontais por ultra-sons e infra-vermelhos. Ainda assim, e uma vez que também existe uma colisão na parte frontal do SALbot, é de esperar que a melhor acção que o robô deve executar perante este estado é andar para trás (acção 5).

Analisando a tabela 4.7, na qual se verifica a evolução da tabela de aprendizagem para o estado em questão, é de fácil percepção que a acção que garante maiores recompensas ao SALbot

Tabela 4.7: Evolução da tabela de aprendizagem para o estado 5921 do teste para a totalidade do ambiente.

Estado 5921						
Acção / Iteração	0	1	2	3	4	5
500	−11	−4	−3	−9	−5	14
1000	−1	−2	−5	2	−5	15
2000	−3	−4	−7	3	−3	11
5000	−5	−10	−2	−2	−4	16

é a acção de andar para trás (acção 5), como era de esperar. Mas, para além da existência de alguma instabilidade nos valores de Q para algumas acções, na acção de andar para a frente (acção 0) existe um aumento significativo do valor de Q entre as 500 e 1000 iterações, que depois vai diminuindo até às 5000. Este resultado é muito estranho visto que, se já existe uma colisão na parte frontal do robô, se este andasse para a frente (acção 0) deveria continuar a colidir, o que implicaria uma recompensa negativa e consequentemente um valor de Q menor. Isto pode dever-se ao facto de a colisão ser numa das extremidades da parte frontal do sistema de pára-choques, o que é possível tendo em conta a diferença de níveis entre os sensores de distância frontais, e de o robô andando para a frente, uma vez que os servos não iniciam a rotação exactamente ao mesmo tempo e existir um servo que roda a uma velocidade maior que o outro, saia do estado de colisão. Outra conclusão, retirada da análise feita à evolução da tabela de aprendizagem para este estado, é que todas as acções foram experimentadas pelo menos uma vez ao final de 500 iteração

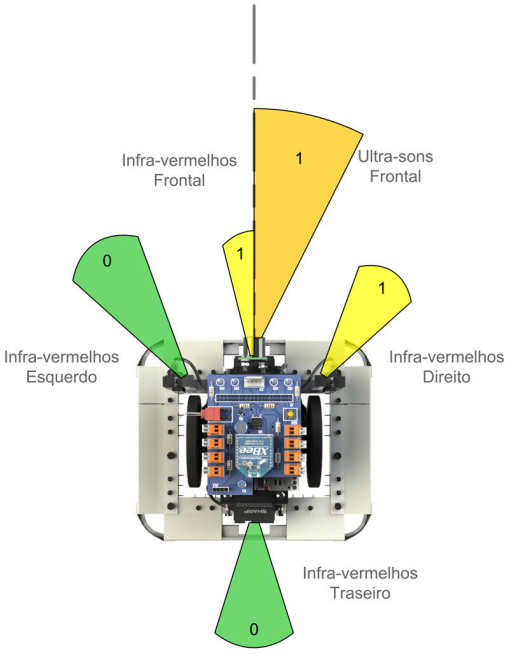


Figura 4.10: Representação do estado 2368 do SALbot.

O estado 2368, como se pode verificar na figura 4.10, corresponde a: nível 1 nos sensores de distância por ultra-sons frontal e infra-vermelhos frontal e lateral direito; nível 0 nos sensores de distância por infra-vermelhos lateral esquerdo e traseiro; e nenhum sensor de contacto do sistema de pára-choques activo. Para este estado, é de esperar que o SALbot aprenda que curvar para a esquerda (acção 2) é a acção que maiores recompensas garante.

Tabela 4.8: Evolução da tabela de aprendizagem para o estado 2368 do teste para a totalidade do ambiente.

Estado 2368						
Iteração \ Acção	0	1	2	3	4	5
500	10	0	−5	8	11	13
1000	6	−9	11	14	12	12
2000	12	−4	−4	13	13	13
5000	12	−3	1	14	7	13

Examinando a evolução da tabela de aprendizagem neste estado (tabela 4.8), verifica-se que a acção que se esperava ser a que maiores recompensas garantisse ao SALbot é a que possui valores mais instáveis e nunca chega a ser a acção com o maior valor de Q para este estado. Para o mesmo estado, após as 5000 iterações, o SALbot aprendeu que a acção que lhe garante uma maior recompensa é a acção de rodar para a direita (acção 3).

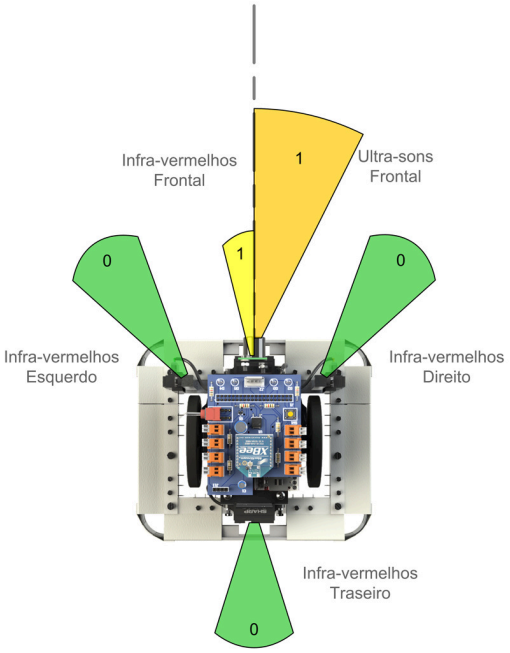


Figura 4.11: Representação do estado 64 do SALbot.

Por último, temos os estado 64, que corresponde a: nível 1 nos sensores de distância por infra-vermelho e ultra-sons frontais; nível 0 nos sensores de distância por infra-vermelhos laterais e traseiro; e nenhum sensor de contacto do sistema de pára-choques activo. Para este estado, não existe uma acção que seja esperada como a que maiores recompensas garanta. Na verdade qualquer acção é passível de ser a melhor. Quanto muito poder-se-á dizer que andar para a frente (acção 0) não deverá ser a que melhores recompensas garanta ao SALbot.

Tabela 4.9: Evolução da tabela de aprendizagem para o estado 64 do teste para a totalidade do ambiente.

Estado 64						
Acção / Iteração	0	1	2	3	4	5
500	0	−5	−6	0	0	12
1000	6	−5	9	0	5	12
2000	11	−3	1	10	9	12
5000	15	−3	2	12	10	12

Fazendo uma análise à evolução dos valores da tabela de aprendizagem para este estado (tabela 4.9), a acção de andar para trás (acção 5) é a que apresenta o valor de Q mais estável ao longo do teste, tendo sido a melhor acção para ser executada neste estado, durante quase todo o teste. No entanto, entre as 2000 e as 5000 iterações o robô aprendeu que a acção que lhe garante uma maior recompensa é a de andar para a frente (acção 0). Mas o facto mais interessante é a grande instabilidade nos valores de Q para a acção de curvar para a esquerda (acção 2), onde, entre as 500 e as 1000 iterações, passou de um valor negativo para um positivo e descendo entre as 1000 e as 2000 iterações para um valor positivo baixo. Em que, mais uma vez, se pode dever ao facto de o estado poder representar diferentes situações da posição do robô, em relação ao ambiente que o rodeia.

4.2.4 Comparativo entre os diferentes testes de aprendizagem

Comparando a evolução da aprendizagem dos vários testes realizados (tabela 4.10), nota-se que para o teste realizado num ambiente pequeno, foram aprendidos mais estados do que nos restantes testes, bem como se preservaram, em mais estados, os valores da tabela de aprendizagem. Na verdade, é de fácil percepção que, à medida que a área do ambiente de testes foi aumentando, o SALbot foi visitando menos estados diferentes. Daí que a instabilidade do valores da tabela de aprendizagem aumente com a área do ambiente de testes. Até porque, visitando menos estados diferentes, o SALbot visita mais vezes os mesmos estados. E, devido ao facto de a percepção do robô, por vezes, corresponder a várias situações distintas, é verificada essa instabilidade nos valores de Q da tabela de aprendizagem.

Tabela 4.10: Comparação das tabelas de aprendizagem entre os diferentes testes de aprendizagem.

	Estados visitados durante a etapa			Novos estados desde a etapa anterior			Estados inalterados desde a etapa anterior		
Teste / Iteração (Etapa)	1/4	1/2	1	1/4	1/2	1	1/4	1/2	1
500	140	112	96	–	–	–	–	–	–
1000	192	165	126	52	53	30	46	34	30
2000	245	231	178	53	66	52	65	55	36
5000	345	297	255	100	66	77	67	51	40

Outra análise realizada, para efeitos de comparação entre os diferentes testes realizados, diz respeito aos gráficos que relacionam o número de colisões acumuladas ao longo das iterações (em que o SALbot vai à tabela de aprendizagem procurar a acção que possui o maior valor de Q para o estado em que se encontra). Na figura 4.12 estão representados os gráficos que mostram a evolução do número de colisões acumuladas ao longo das iterações dos testes efectuados. É de referir que, no teste realizado num ambiente pequeno, os dados entre as 1000 e 2000 iterações relativos ao número de colisões acumuladas perderam-se devido a um problema, alheio à comunicação e ao programa de interface gráfica, originado pelo sistema operativo, impedindo, por isso, de guardar esses dados. Ainda assim, é possível ver a tendência do gráfico.

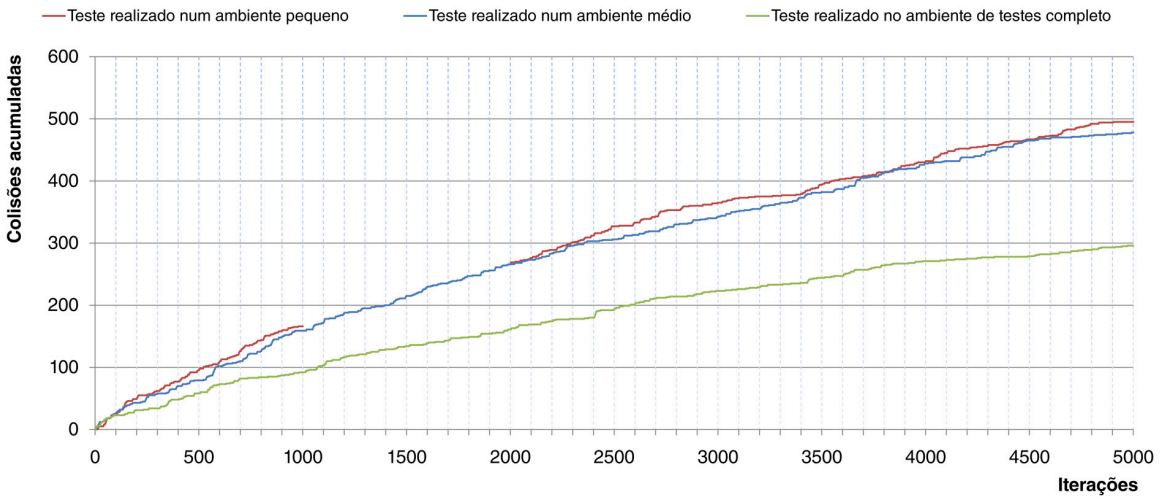


Figura 4.12: Comparação entre os testes dos gráficos do número de colisões ao longo das iterações.

Seria de esperar que, à medida que a área do ambiente aumentasse o número de colisões acumuladas diminuísse, no entanto, isso só se verifica quando se aumenta de metade da área do ambiente de testes para a sua totalidade. Isto é, entre o teste realizado num ambiente pequeno e o teste realizado num ambiente médio, não existem diferenças significativas na evolução do número de colisões. Isso pode dever-se ao facto de, em ambos os testes, se visitarem sensivelmente os mesmos estados ambíguos, isto é, os estados em que a percepção do robô pode corresponder a diferentes

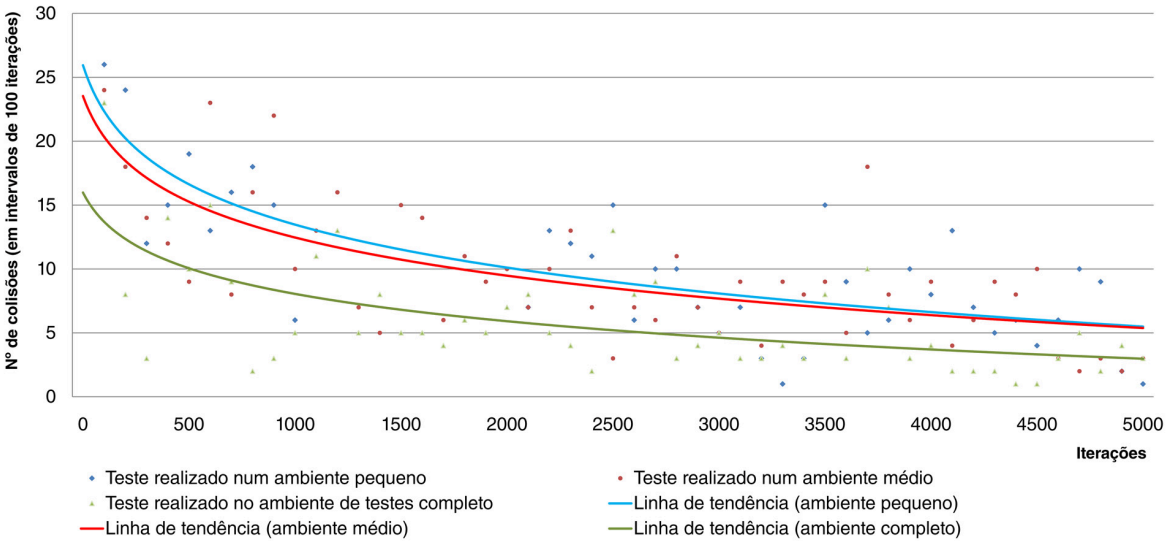


Figura 4.13: Número de colisões em intervalos de 100 iterações para os testes de aprendizagem.

situações, nos quais o SALbot, executando uma acção, pode receber diferentes recompensas. É de salientar que, quando o SALbot se encontrava nos cantos do ambiente de testes, o número de colisões acumuladas aumentava consideravelmente, como, por exemplo, se pode verificar no gráfico do teste realizado na área total do ambiente de testes, perto das 2400 iterações (mais sensivelmente entre as 2403 e 2420 iterações, entre as quais se registaram 11 colisões).

Apesar dos testes terem sido realizados para poucas iterações (embora cada um destes tenha demorado aproximadamente 5 horas seguidas), é possível verificar uma certa tendência dos gráficos em convergir para um valor máximo, especialmente para o gráfico do último teste. Contudo, essa tendência pode não se verificar uma vez que, devido aos tais estados ambíguos, o SALbot, mesmo após n iterações, ainda continuar a aumentar o número de colisões acumuladas. Isto é, apesar de não ser garantido que o SALbot deixe definitivamente de colidir, o número de colisões vai diminuindo à medida que vai aprendendo.

Na figura 4.13, é possível ter uma melhor percepção da evolução da aprendizagem nos diversos testes efectuados. Neste gráfico são representadas as colisões em intervalos de 100 iterações (é de relembrar que, no teste efectuado num ambiente pequenos, não existem dados entre as 1000 e 2000 iterações). Para melhor interpretação do gráfico, foram representadas linhas de tendência para cada teste. Assim, analisando as linhas de tendência, verifica-se que o número de colisões tende para um valor próximo de zero à medida que o número de iterações vai aumentando.

4.2.5 Resultados em operação no modo autónomo

Após a realização dos vários testes de aprendizagem, realizaram-se testes de execução do SALbot no modo autónomo para cada ambiente (pequeno, médio e completo). Mais detalhadamente, para cada ambiente, colocou-se o SALbot a executar 1000 acções no modo autónomo, para duas tabelas de aprendizagem distintas: com aprendizagem de 500 iterações; e com aprendizagem de 5000 iterações. Durante estes testes, foram retirados o número de colisões acumuladas ao longo do número de acções.

Através da visualização dos gráficos, obtidos com os dados retirados destes testes,(figura 4.14, 4.15 e 4.16) é clara a evolução da aprendizagem do SALbot. Em todos os ambientes, existe uma redução do número de colisões da aprendizagem com 500 iterações para a aprendizagem com 5000 iterações.

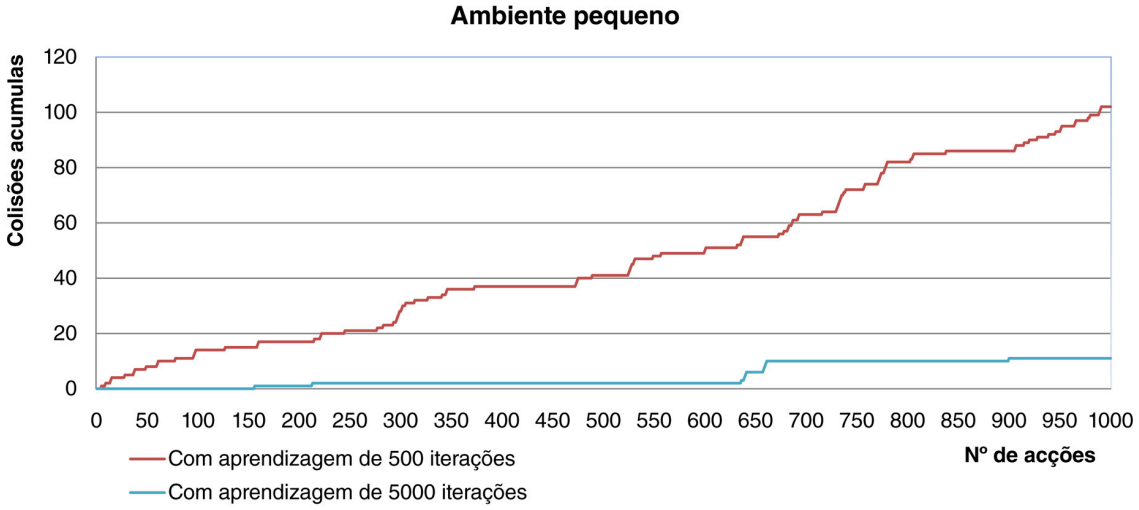


Figura 4.14: Comparação entre etapas da aprendizagem do número de colisões no modo autónomo no ambiente pequeno.

Comparando os gráficos já referidos, o SALbot, com a tabela de aprendizagem de 500 iterações, colidiu mais vezes no ambiente pequeno do que nos outros ambientes, o que faz sentido, visto que o espaço do ambiente é menor, isto é, existe menos espaço livre para o robô se movimentar. No entanto, o número de colisões, com a tabela de aprendizagem de 5000 iterações, é menor no ambiente pequeno do que nos restantes. Isto deve-se ao facto de, sendo menor a dimensão do ambiente, o SALbot visitar mais e mais vezes os estados possíveis do ambiente.

Analisando os gráficos das figuras: 4.17, 4.18 e 4.19, torna-se mais evidente a evolução da aprendizagem de 500 iterações para 5000 iterações, em qualquer dos ambientes. As linhas de tendência para as aprendizagens de 5000 iterações, além de representarem um número de colisões menor, são mais estáveis (menor declive) do que para as aprendizagens de 500 iterações. Isto é, comparando o número de colisões entre as aprendizagens de 500 e 5000 iterações, o número de colisões diminui e a variação do número de colisões também diminui.

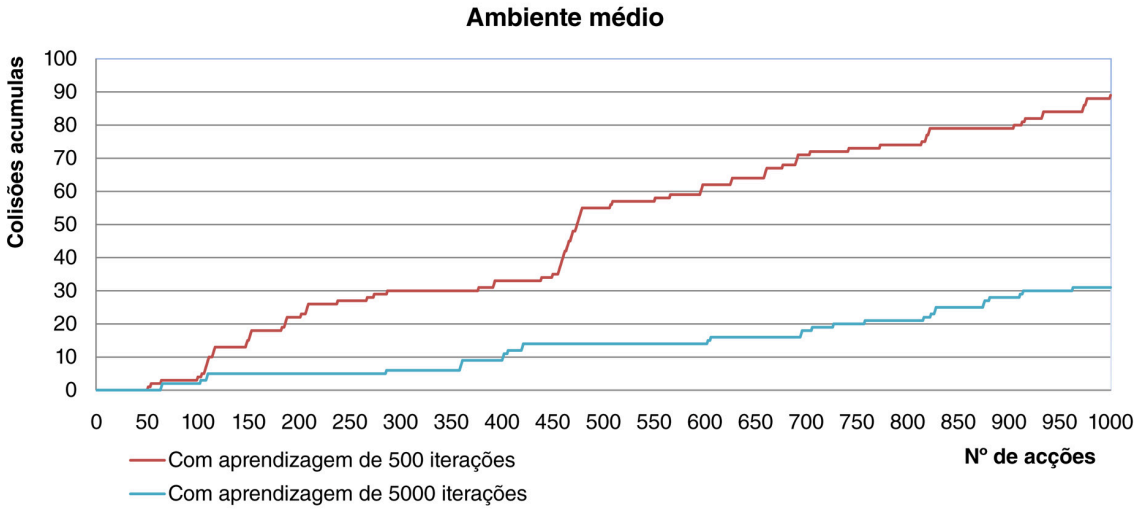


Figura 4.15: Comparação entre etapas da aprendizagem do número de colisões no modo autónomo no ambiente médio.

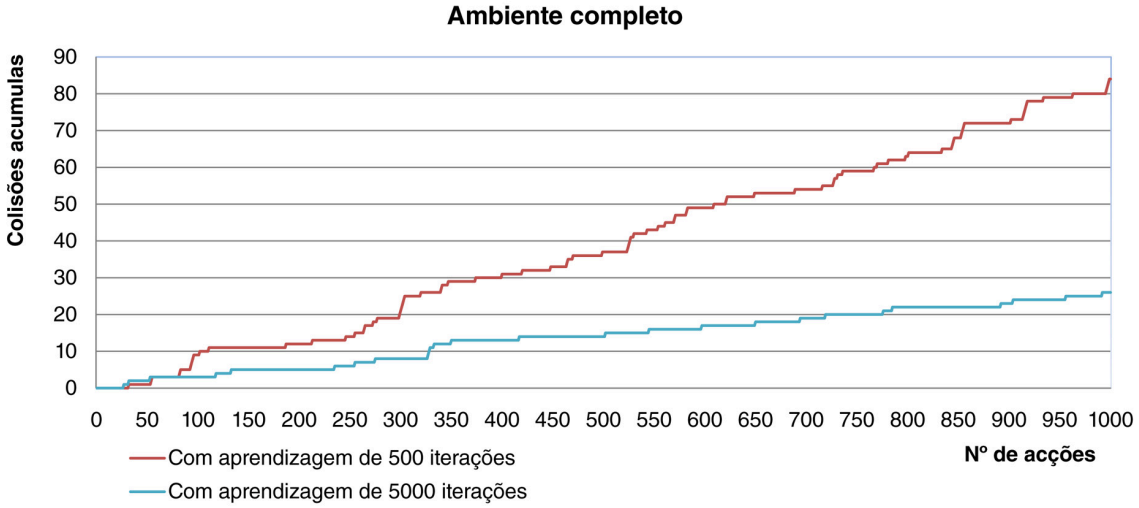


Figura 4.16: Comparação entre etapas da aprendizagem do número de colisões no modo autónomo num ambiente completo.

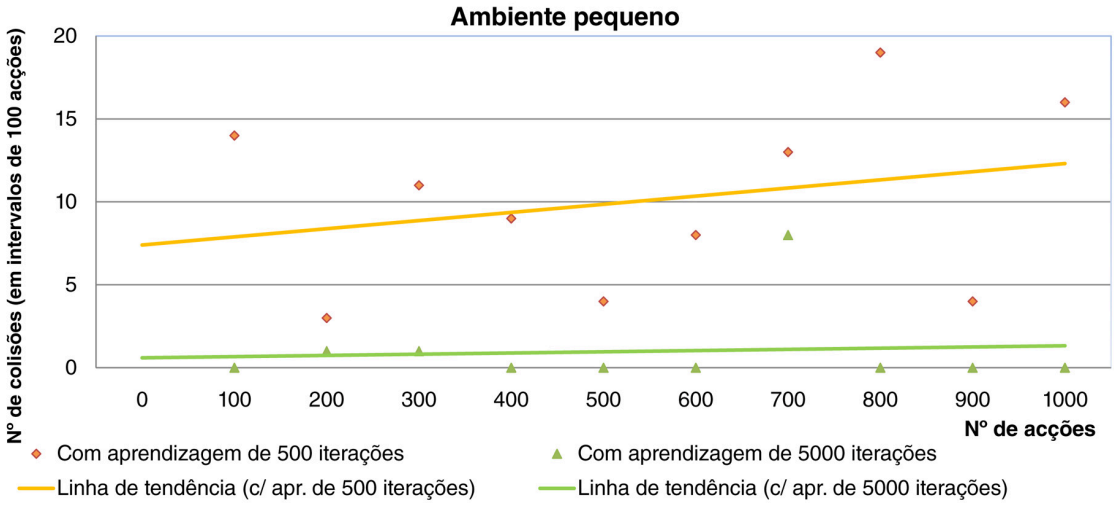


Figura 4.17: Número de colisões em intervalos de 100 acções para o modo autónomo no ambiente pequeno.

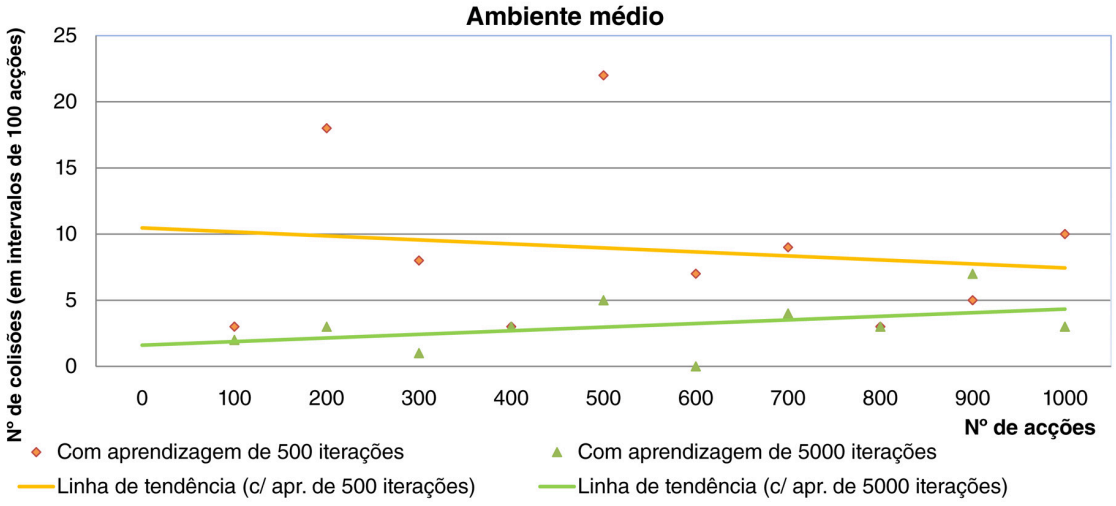


Figura 4.18: Número de colisões em intervalos de 100 acções para o modo autónomo no ambiente médio.

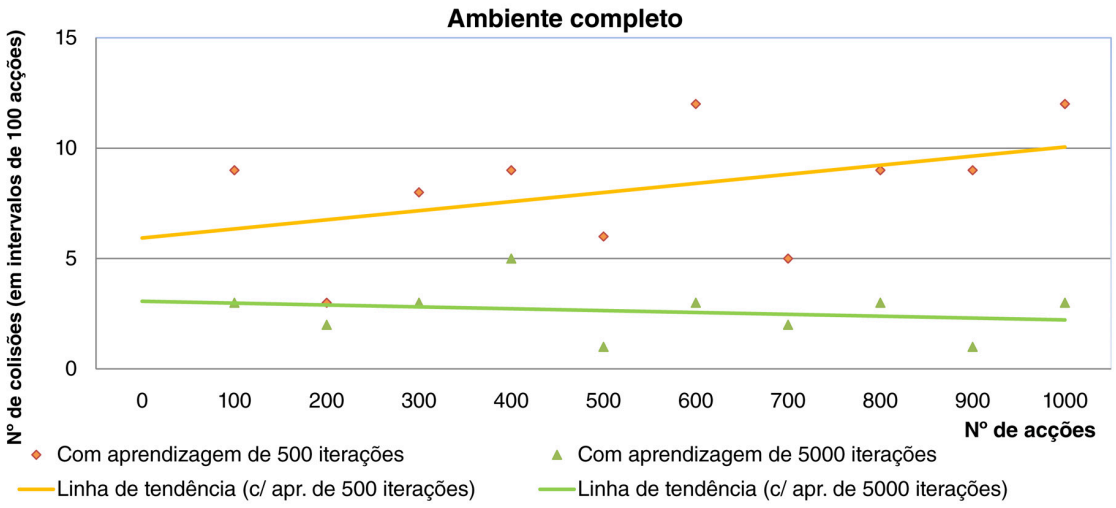


Figura 4.19: Número de colisões em intervalos de 100 acções para o modo autónomo no ambiente completo.

Capítulo 5

Conclusões e trabalhos futuros

"A experiência nunca falha, apenas as nossas opiniões falham, ao esperar da experiência aquilo que ela não é capaz de oferecer."

Leonardo da Vinci

5.1 Conclusões

Dos diferentes tipos de aprendizagem de máquina existentes, escolheu-se o *reinforcement learning* como base para a concepção e desenvolvimento dos SALbot. Ainda assim, é possível configurar o SALbot, para se utilizar qualquer tipo de aprendizagem de máquina. Foram instalados sensores de distância por infra-vermelhos e ultra-sons e sensores de contacto para aquisição de informação do ambiente, um sistema de locomoção, um controlador, um sistema de comunicação sem fios e unidades de bateria, que, em conjunto, garantem a autonomia dos robôs. Além disso, os SALbot foram concebidos numa plataforma modular, permitindo assim a implementação de novos dispositivos no futuro, quer para o melhoramento na área de aprendizagem, quer para possibilitar a sua utilização para outras áreas da robótica.

O chassi e o sistema de pára-choques dos SALbot foram concebidos para serem robustos e fiáveis, relativamente à funcionalidade dos robôs. Além disso, o sistema de pára-choques foi concebido para detectar colisões em oito zonas distintas, e, assim, permitir ao SALbot distinguir com maior precisão a zona de contacto. Após a realização dos testes, o chassi e o sistema de pára-choques provaram ser fiáveis e robustos para a função dos robôs. O chassi suportou eficazmente todos os componentes do SALbot durante, e após, os testes efectuados e o sistema de pára-choques cumpriu o objectivo desejado.

O chassi e o sistema de pára-choques foram construídos, em grande parte, recorrendo a processos artesanais. Apesar de grande parte da construção depender da precisão do trabalho manual, a robustez e fiabilidade dos componentes do robô nunca ficaram comprometidas.

Foram também desenvolvidas placas de circuito impresso, uma principal e uma auxiliar para cada SALbot, e uma para as comunicações por *wireless* entre um computador remoto e os SALbot. Na placa de circuito impresso principal estão integrados e são ligados os componentes necessários para a autonomia do SALbot, destacando-se: um microcontrolador; os servos; os sensores de distância; e as baterias de alimentação da placa e dos servos. Além disso, a placa de circuito

impresso principal possui ainda uma porta de expansão, que garante a referida modularidade do sistema, permitindo a integração da placa auxiliar, que integra o sistema de comunicações por *wireless* e onde são ligados os sensores de contacto do sistema de pára-choques.

Em geral, os componentes electrónicos cumpriram a sua função sem grandes problemas. No entanto, os servos ficaram um pouco aquém do que seria desejável, esperava-se que fossem mais precisos e que tivessem todos o mesmo comportamento para um comando. O facto de a actuação dos servos variar de uns para os outros faz com que, por exemplo, seja impossível o robô realizar uma trajectória perfeitamente rectilínea. Outro dos problemas verificados no *hardware* electrónico, tem a ver com os módulos XBee, onde se verificou que, por vezes, falhava a comunicação entre os robôs e o computador remoto a uma distância de aproximadamente dois metros. Isso pode-se ter devido a alguma interferência com as redes Wi-Fi (protocolo IEEE 802.11 a,b,g,n) existentes no laboratório de robótica. Este problema pode ser resolvido através de *software*, implementado no protocolo de comunicação, entre o robô e um computador remoto, uma mensagem de aviso de recepção, isto é, comunicação com confirmação de recepção. Ou então, também pode ser resolvido implementando módulos XBee-PRO, que possuem uma potência de transmissão e sensibilidade de recepção maiores.

A memória não volátil, existente nos SALbot para armazenamento de dados da aprendizagem, foi suficiente para o tipo aprendizagem utilizada nos testes efectuados. No entanto, para aprendizagens mais complexas (com mais estados e acções) é possível que, mesmo com a implementação de mais três unidades de EEPROM externa (através do barramento $I^2C^{™}$), não seja suficiente para armazenar dados da aprendizagem. Uma solução possível é implementar uma unidade para armazenamento e leitura de dados para um cartão SD (*Secure Digital*), onde é possível armazenar até um máximo de 4 GB de dados. Para isso, apenas seria necessário implementar um microcontrolador que tratasse da interface com o cartão SD e, através do barramento $I^2C^{™}$, comunicasse, com o mesmo protocolo das unidades EEPROM, com o microcontrolador do SALbot.

Outra limitação constatada, tem a ver com a capacidade de carga das baterias de alimentação das placas, que é apenas de 250 *mAh*. Verificou-se que uma bateria com carga máxima permite alimentar o SALbot, sensivelmente, durante apenas uma hora e quinze minutos, pelo que, na realização dos testes de aprendizagem, foi necessário mudar três a quatro vezes as baterias. A implementação de baterias com maior capacidade de carga resolverá este problema, recorrendo, por exemplo, a baterias de iões de lítio.

O protocolo de comunicações, implementado para comunicação entre os SALbot e um computador remoto, mostrou-se bastante eficaz não admitindo mensagens corrompidas ou outro tipo de mensagens inválidas. Faltando apenas, como já foi referido, implementar uma mensagem para aviso de recepção de uma mensagem, reduzindo assim a perda de dados.

Da análise realizada às tabelas de aprendizagem, obtidas a partir dos testes de aprendizagem efectuados, são de destacar algumas considerações. A primeira é o facto de, um mesmo estado de percepção do SALbot poder corresponder a várias situações do ambiente que o rodeia, o que pode retardar ou inviabilizar a aprendizagem de acções nesses estados ambíguos. Outra consideração, provém do facto de o SALbot ter visitado estados improváveis de serem visitados, como o estado 8800 (4.3) em que existe uma grande diferença entre as leituras dos sensores de distância frontais. A resolução e/ou minimização destes problemas passa por integrar mais sensores de distância possibilitando ao SALbot a obtenção de mais informação do ambiente e pela criação de mais estados (aumentando o número de níveis dos sensores de distância e as zonas de contacto).

Apesar das questões atrás referidas, os testes de aprendizagem realizados permitiram provar um dos principais objectivos deste trabalho: a capacidade de aprendizagem dos SALbot. Embora o resultado da aprendizagem possa não ter sido o esperado para todos os casos, da análise realizada

às tabelas de aprendizagem concluiu-se que existe uma evolução dos valores de Q para os pares de estado-acção experimentados pelo robô. O que traduz efectivamente a aprendizagem do SALbot em navegar num ambiente sem colidir. Além disso, os resultados dos testes realizados para a execução do SALbot, no modo autónomo, indicam claramente a evolução da aprendizagem do SALbot. Nos quais se verifica que o SALbot diminui o número de colisões com o aumento do número de iterações de aprendizagem.

5.2 Trabalhos futuros

A realização deste trabalho abriu um amplo leque de trabalhos futuros na área de aprendizagem robótica, dos quais se destacam: comparar diversas configurações do método de aprendizagem implementado neste trabalho; testar e comparar novos métodos de aprendizagem; fazer a comparação entre um algoritmo de navegação autónoma com um algoritmo de aprendizagem; Integrar novos dispositivos (sensores, microcontroladores, unidades de memória, etc...) no SALbot para testar aprendizagens mais complexas.

Apesar de terem sido concebidos para a área de aprendizagem robótica, os SALbot podem integrar projectos de outras áreas da robótica envolvendo robôs móveis autónomos.

Independentemente da área da robótica que possa ser abordada em projectos futuros, seria uma mais valia fazer algumas alterações no *hardware* dos SALbot. Como as alterações já referidas anteriormente: substituição dos servos e baterias, da implementação de um maior número de sensores de distância e uma unidade de armazenamento de dados superior. Mas também seria uma mais valia integrar no sistema mais um microcontrolador, permitindo assim que um deles ficasse responsável pela actuação dos servos e aquisição dos dados dos sensores, e o outro ficasse responsável pelo processamento dos dados e algoritmo de aprendizagem (entre outros). A implementação de um dispositivo de relógio de tempo real seria também uma melhoria significativa, visto que permitiria ao microcontrolador ter o controlo exacto do tempo que é importante para gerar números aleatórios. Para isso, seria também interessante a concepção de uma nova placa de circuito impresso, que integrasse não só esse novo microcontrolador, como duas portas de expansão (para permitir ser integrada na placa de circuito principal e a integração da placa de circuito auxiliar do SALbot) e outros dispositivos necessários, como o relógio de tempo real.

Espera-se, com a realização deste projecto, permitir futuras investigações na área da aprendizagem robótica, testando e validando algoritmos de aprendizagem que possam ser implementados noutros sistemas robóticos, com a finalidade de melhorar o desempenho funcional desses sistemas.

Apêndice A

Algoritmos de Reinforcement Learning

A.1 *Markov Decision Processes*

No *Reinforcement Learning*, um agente toma as suas decisões em função dos sinais obtidos do ambiente, e que constituem o "estado". Um estado que contenha todas as informações do ambiente, é designado *Markoviano*, ou como tendo a propriedade *Markoviana*.[30] Assumindo que, o número de estados são finitos, passamos a trabalhar em termos de probabilidades e somas, em vez de integrais e probabilidades densas. Se considerarmos como um ambiente geral irá responder num tempo $t+1$, a uma acção realizada em t, a resposta pode depender de tudo o que se passou anteriormente. Neste caso, a dinâmica pode ser definida da seguinte forma:

$$P_r\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \dots, r_1, s_0, a_0\},\ \forall\ s', r, s\ e\ a \tag{A.1}$$

em que r é a recompensa, s o estado, s' o próximo estado e a a acção. Se o estado for *Markoviano*, então a resposta do ambiente no tempo $t+1$ passa a depender apenas do estado e da acção em t. Neste caso, a dinâmica passa a ser definida por:

$$P_r\{s_{t+1} = s', r_{t+1} = r \mid s_t, at\},\ \forall\ s', r, s_t\ e\ a_t \tag{A.2}$$

Ou seja, um estado inicial é um estado *Markoviano*, apenas se a equação A.1 for igual a A.2 para todo o s', r e contiver um historial dos valores, $s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \dots, r_1, s_0, a_0$. Neste caso, o ambiente e as tarefas também se designam como tendo a propriedade *Markoviana*. No *reinforcement learning*, a propriedade *Markoviana* é importante, porque as decisões e valores são assumidas apenas em função do estado actual.

Um processo *Markoviano* é um processo estocástico, que satisfaz a propriedade *Markoviana*. E uma tarefa de *Reinforcement Learning* que contenha processos *Markovianos*, em tempos discretos, é designada por MDP (*Markov Decision Process*). No entanto, mesmo que a informação do estado seja não-*Markoviana*, continua a ser válido tratar o estado, no *reinforcement learning*, como sendo uma aproximação ao estado *Markoviano*. No caso de os estados e as acções serem finitas, então estamos na presença de um MDP finito. Para qualquer estado s e acção a a probabilidade de o próximo estado ser s', designado por probabilidade de transição é dada por:

$$\mathcal{P}^a_{ss'} = P_r\{s_{t+1} = s' \mid s_t = s, a_t = a\} \tag{A.3}$$

De igual modo, para um dado estado s, acção a e próximo estado s', o valor esperado para a recompensa é dado por:

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \tag{A.4}$$

Tanto a probabilidade de transição $\mathcal{P}^a_{ss'}$ como a recompensa esperada $\mathcal{R}^a_{ss'}$ especificam os aspectos mais importantes da dinâmica de um MDP finito.[30]

A.2 Funções de Valor

Quase todos os algoritmos de *reinforcement learning* são baseados na estimativa de funções de valor. As funções de valor são funções de estados, ou pares de estado-acção, que estimam o valor, que pode ser esperado de futuras recompensas, por um agente estar num dado estado, ou por um agente realizar uma acção num dado estado. As funções de valor são definidas adequadamente com as políticas em causa.[30]

Uma política π, de um agente, é um mapeamento de cada estado $s \in \mathcal{S}$ e acção $a \in \mathcal{A}(s)$ para a probabilidade $\pi(s, a)$ de um agente escolher uma acção a, possível, estando num estado s. Para MDPs, o valor de um estado s, regido por uma dada política π, designado por $V^\pi(s)$ ou função de valor de estado para a política π, é dado por:

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\} \tag{A.5}$$

em que, $E_\pi\{\}$ representa a esperança de um agente ter seguido a política π num tempo t e γ ($0 < \gamma < 1$, utilizando-se, normalmente, valores próximos de 1) é designado por factor de desconto.

Da mesma maneira, o valor de executar uma acção a num estado s, através de uma política π, designado por $Q^\pi(s, a)$ ou função de valor de estado-acção para a política π, é dado por:

$$Q^\pi(s, a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\} \tag{A.6}$$

No *reinforcement learning*, as funções de valor satisfazem relações recursivas particulares. Para qualquer política π e estado s, a seguinte condição mantém a coerência entre o valor do estado s e o valor dos estados seguinte:

$$\begin{aligned}
V^\pi(s) &= E_\pi\{R_t \mid s_t = s\} \\
&= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s\right\} \\
&= E_\pi\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s\right\} \\
&= \sum_a \pi(s, a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_{t+1} = s'\right\}\right] \\
&= \sum_a \pi(s, a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right], \;\forall\, a \in \mathcal{A}(s), s \in \mathcal{S}
\end{aligned} \tag{A.7}$$

Esta equação (eq. A.7) representa a relação entre o valor de um estado e o valor do estado seguinte, designada por equação de *Bellman*, cuja única solução é V^{π}.[30]

Encontrar uma política, cujas acções devolvam as melhores recompensas, designada por $V^*(s)$, é o objectivo de uma tarefa de *reinforcement learning*. Uma política π é melhor que uma política π' se a sua recompensa esperada for maior, ou igual, que a recompensa esperada para π'. A política que devolva a maior recompensa é designada por π^*, ou política óptima. Para MDPs finitos, a função de valor óptima é definida por:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \ \forall \ s \in \mathcal{S} \tag{A.8}$$

Existe pelo menos uma política óptima π^*, no entanto, podem existir mais do que uma. Essas políticas, para além de partilharem as mesmas funções de estado óptimas V^*, partilham, também, as funções de estado-acção óptimas. A função de estado óptima, ou Q^*, é definida por:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \ \forall \ s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{A.9}$$

Uma vez que, para cada par estado-acção, esta função devolve a recompensa esperada de executar uma acção a num estado s, seguindo uma política óptima π^*, então é possível definir Q^* em função de V^*:

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \tag{A.10}$$

Sendo V^* a função de valor para uma política, esta deve satisfazer a condição de coerência, dada pela equação de *Bellman* (eq. A.7). Esta condição de coerência, designada por equação de *Bellman* para V^* ou equação óptima de *Bellman*, pode ser escrita, sem referenciar qualquer tipo específico de política, da seguinte forma:

$$\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s,a) \\
&= \max_{a} E_{\pi^*}\{R_t \mid s_t = s, a_t = a\} \\
&= \max_{a} E_{\pi^*}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a \right\} \\
&= \max_{a} E_{\pi^*}\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s, a_t = a \right\} \\
&= \max_{a} E\{r_{t+1} + \gamma V^*(s_t + 1) \mid s_t = s, a_t = a\} \qquad \text{(A.11)} \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s') \right] \qquad \text{(A.12)}
\end{aligned}$$

As duas últimas equações (A.11 e A.12) são duas formas da equação óptima de *Bellman*. Basicamente, a equação óptima de *Bellman* traduz o facto de que um valor de um estado, numa política óptima, deve igualar a recompensa esperada para a melhor das acções para aquele estado.[30] De igual forma, a equação óptima de *Bellman* para Q^* é dada por:

$$\begin{aligned}
Q^*(s,a) &= E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \;\middle|\; s_t = s, a_t = a \right\} \\
&= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right] \qquad \text{(A.13)}
\end{aligned}$$

A equação óptima de *Bellman* tem uma única solução, para MDPs finitas, independente da política. Na verdade, a equação óptima de *Bellman* é um sistema de equações, um para cada estado. Assim, para n estados, existem n equações com n incógnitas. É possível resolver o sistema de equações para V^*, e consequentemente para Q^*, se a dinâmica do meio ($\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$) for conhecida.

Uma vez tendo V^*, é fácil determinar a política óptima. Basicamente, a política óptima é aquela que é *greedy* (ávida) em relação à função de valor óptima V^*. Tendo V^* em conta as recompensas para todos os comportamentos futuros, se V^* for usada para avaliar as consequências das acções a curto prazo, uma política *greedy* (ávida) é óptima a longo prazo. Por outras palavras, a recompensa a longo prazo é transformada num valor que está disponível imediatamente para cada estado e, portanto, uma procura no passo seguinte contém as acções óptimas a longo prazo.

Escolher as acções óptimas tendo Q^* é ainda mais simples, pois, não é necessário realizar uma procura no passo seguinte para cada estado, basta simplesmente procurar a acção que maximiza $Q^*(s, a)$. A função de valor de estado-acção dá-nos a recompensa esperada a longo prazo na forma de um valor, imediatamente disponível para cada par estado-acção. A função de valor, por representar pares de estado-acção, em vez de apenas estados, permite escolher as acções óptimas sem nada conhecer sobre a dinâmica do ambiente, isto é, sem conhecer nada sobre os possíveis estados futuros e os seus valores.[30]

A.3 Métodos *Model-free*

Como já foi referido anteriormente, existem dois tipos básicos de *reinforcement learning*: o *model-based* e o *model-free*. Serão descritos alguns métodos de cada um destes de tipos, sendo dada mais relevância aos métodos *model-free*, uma vez que este tipo é mais adequado para pequenos robôs, pelo facto de não ser necessário um modelo do ambiente. Nesta secção, são descritos alguns métodos *model-free* de *reinforcement learning*, nomeadamente o *Monte Carlo Control*, e três tipos de métodos de diferença temporal (TD - *Temporal-Difference*): o *Q-learning*, SARSA (*State-Action-Reward-State-Action*) e o TD(0).

A.3.1 *Monte Carlo Control*

Os métodos *Monte Carlo* são modos de resolver problemas de *reinforcement learning*, baseando-se na média de recompensas obtidas, nos quais, apenas são necessárias sequências de estados, acções e recompensas, através da interacção de um agente com o ambiente para aprender.

No método *Monte Carlo Control*, a função de valor é ciclicamente alterada, por forma a aproximar a função de valor à política utilizada e a política é ciclicamente melhorada em função da função valor actual.[30] Similarmente à iteração de política, realizam-se passos de avaliação de política e passos de aperfeiçoamento de política alternadamente, iniciando-se com uma política aleatória π_0 e terminando com uma política óptima π^* e uma função de valor de estado-acção óptima Q^*, da seguinte forma:

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} Q^*, \tag{A.14}$$

Onde $\xrightarrow{E}$ representa uma avaliação de política completa e $\xrightarrow{I}$ representa um aperfeiçoamento de política completo. A avaliação de política resume-se ao cálculo da função de valor de estado V^π para uma política π, recorrendo à equação A.8, e é definida por:

$$\begin{aligned} V^\pi(s) &= E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \mid s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\},\ \forall\ s \in \mathcal{S} \end{aligned} \tag{A.15}$$

em que $\pi(s, a)$ é a probabilidade de executar uma acção a num estado s seguindo a política π. O aperfeiçoamento de política consiste no cálculo do valor de estado-acção de escolher uma acção a num estado s seguindo uma política π:

$$Q^\pi(s, a) = E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a\} \tag{A.16}$$

e comparar esse valor com o valor de estado, para verificar se é ou não melhor mudar para uma nova política nesse estado s. Se, para um par de políticas π e π', se verificar que:

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s),\ \forall\ s \in \mathcal{S} \tag{A.17}$$

então a política π' é tão boa ou melhor que a política π. Por outras palavras, a recompensa esperada por executar $a \neq \pi$ num estado s, é maior ou igual do que seguir a política π. Assim, para qualquer função de valor de estado-acção Q, a política *greedy* correspondente é dada por:

$$\pi(s) = \arg\max_a Q(s, a),\ \forall\ s \in \mathcal{S} \tag{A.18}$$

O algoritmo mais simples do método *Monte Carlo* é o *Monte Carlo ES* (*Monte Carlo* com exploração inicial), em que, após cada episódio, é obtida uma recompensa, que é usada para a avaliação de política e depois é realizada o aperfeiçoamento de política para todos os estados visitados no episódio. Este algoritmo é descrito na tabela A.1.

Para que o método de *Monte Carlo* tenha, facilmente, a garantia de convergência, são assumidas duas condições: em cada episódio começa-se por explorar e que a avaliação de política pode ser realizada num número infinito de episódios. Porém, para se poder obter um algoritmo prático, é necessário descartar estas duas condições não realísticas. Uma das formas de evitar a realização de avaliação de política, num número infinito de episódios, é interromper a convergência da avaliação de política, antes de voltar ao aperfeiçoamento de política. A outra é aproximar a função de valor de estado-acção Q^π, em vez de calculá-la com precisão.

A única maneira geral de garantir que todas as acções são escolhidas infinitamente várias vezes, evitando o uso de exploração inicial, é um agente continuar a escolhe-las. Existem duas abordagens que garantem essa condição, designadamente os métodos *on-policy* e *off-policy*. Os métodos *on-policy* avaliam e aperfeiçoam as políticas que são realizadas para tomar decisões, enquanto que, nos métodos *off-policy*, essas duas funções são separadas para duas políticas distintas. A política usada para tomar decisões, designada por política de comportamento, pode nem sequer estar relacionada com a política que é avaliada e aperfeiçoada, designada por política de estimativa.[30]

Tabela A.1: Pseudo-código do *Monte Carlo ES*, um algoritmo do método *Monte Carlo Control* que usa a exploração inicial.[30]

Inicializar, $\forall\ s \in \mathcal{S}, a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrário
$\pi(s) \leftarrow$ arbitrário
$R(s,a) \leftarrow \emptyset$

Repetir infinitamente:
Gerar um episódio usando exploração inicial e π
Para cada (s,a) visitado no episódio:
$r \leftarrow$ recompensa por seguir π
adicionar r a $R(s,a)$
$Q(s,a) \leftarrow$ média de $R(s,a)$
Para cada s do episódio:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

A.3.2 *on-policy Monte Carlo Control*

No método *on-policy Monte Carlo Control*, para evitar o uso de exploração inicial, são usadas, geralmente, políticas *soft*, isto é, $\pi(s,a) \geq 0,\ \forall\ s \in \mathcal{S}, a \in \mathcal{A}(s)$. Existem várias políticas *soft*, entre as quais, se destacam a ε-*soft*, a ε-*greedy* e a *softmax* que utiliza a distribuição de *Boltzmann*, designada por estratégia de exploração de *Boltzmann*. Estas políticas estão definidas na tabela A.2, onde, na política *softmax*, é definida a probabilidade de se escolher uma acção a num estado s e em que $\tau \geq 0$ é um parâmetro, designado por "temperatura". No limite, quando o número de iterações t tende para infinito, $\tau \to 0$ e é definido por:

$$\tau = T\upsilon^t$$

em que υ é um factor de "fortificação", uma constante próxima de 1, e T é um factor de escala.[29]

A ideia geral do *on-policy Monte Carlo Control* é que, em relação a Q^π, qualquer política ε-*greedy* é um aperfeiçoamento de uma política ε-*soft* de π, que é garantido pelo aperfeiçoamento de política. Para tal, se considerarmos π' como sendo uma política ε-*greedy*, temos que:

$$\begin{aligned} Q^\pi(s,\pi'(s)) &= \sum_a \pi'(s,a) Q^\pi(s,a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a Q^\pi(s,a) + (1-\varepsilon) \max_a Q^\pi(s,a) \\ &\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a Q^\pi(s,a) (1-\varepsilon) \sum_a \frac{\pi(s,a) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} Q^\pi(s,a) \end{aligned} \tag{A.22}$$

Tabela A.2: Tipos de estratégia de exploração *soft*

$$\varepsilon\text{-soft} \quad : \quad \pi(s,a) \geq \frac{\varepsilon}{|\mathcal{A}(s)|},\ \forall\ s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{A.19}$$

$$\varepsilon\text{-greedy} \quad : \pi(s,a) \begin{cases} \dfrac{\varepsilon}{|\mathcal{A}(s)|} & \rightarrow \text{não maximizante} \\ 1-\varepsilon+\dfrac{\varepsilon}{|\mathcal{A}(s)|} & \rightarrow greedy\ \text{(ávida)} \end{cases} \tag{A.20}$$

$$softmax \quad : \quad \pi(s,a) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{b=1}^{|\mathcal{A}(s)|} e^{\frac{Q(s,b)}{\tau}}} \tag{A.21}$$

Uma vez que, a soma é uma média de valores não negativos que tendem para 1, e como tal deve ser menor ou igual que o maior valor médio, então temos que:

$$\begin{aligned} Q^{\pi}(s,\pi'(s)) &= \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_{a} Q^{\pi}(s,a) - \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_{a} Q^{\pi}(s,a) + \sum_{a}\pi(s,a)Q^{\pi}(s,a) \\ &= V^{\pi}(s),\ \forall\ s \in \mathcal{S} \end{aligned}$$

$$\pi' \geq \pi \Leftrightarrow V^{\pi'}(s) \geq V^{\pi}(s),\ \forall\ s \in \mathcal{S}$$

É necessário provar, igualmente, para quando π' e π são melhores ou iguais que todas as outras políticas ε-*soft*. Se considerarmos um novo ambiente, igual ao original, à excepção das políticas serem ε-*soft*. O comportamento no novo ambiente é o mesmo do que no ambiente original com políticas ε-*soft*. Sendo $\tilde{V}^*$ e $\tilde{Q}^*$, a função de valor de estado óptimo e a função de valor de estado-acção óptimo, respectivamente. Então, uma política π é óptima, entre políticas ε-*soft*, se e só se $V^{\pi} = \tilde{V}^*$, condição essa provada por Sutton & Barto em *Reinforcement Learning: an Introduction*.[30]

Assim, alcançamos a melhor política, apenas, a partir de políticas ε-*soft*, mas, por outro lado, eliminamos a condição de exploração inicial. O algoritmo que traduz o método *on-policy Monte Carlo Control* está descrito na tabela A.3.

Tabela A.3: Pseudo-código do algoritmo do método *on-policy Monte Carlo Control*.[30]

Inicializar, $\forall\ s \in \mathcal{S}, a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrário
$R(s,a) \leftarrow \emptyset$
$\pi(s) \leftarrow$ política ε-*soft* arbitrária

Repetir infinitamente:
Gerar um episódio usando π
Para cada (s,a) visitado no episódio:
$r \leftarrow$ recompensa por seguir π
adicionar r a $R(s,a)$
$Q(s,a) \leftarrow$ média de $R(s,a)$
Para cada s do episódio:
$a^* \leftarrow \arg\max_a Q(s,a)$
$\forall\ a \in \mathcal{A}(s)$:

$$\pi(s,a) \leftarrow \begin{cases} 1-\varepsilon+\frac{\varepsilon}{|\mathcal{A}(s)|} & , \text{se } a = a^* \\ \frac{\varepsilon}{|\mathcal{A}(s)|} & , \text{se } a \neq a^* \end{cases}$$

A.3.3 *off-policy Monte Carlo Control*

No método *off-policy Monte Carlo Control*, como já foi referido atrás, temos duas políticas distintas, a política de comportamento, que escolhe as acções, e a política de estimativa, que é avaliada e aperfeiçoada e a partir da qual se aprende. A vantagem de ter duas políticas para funções diferentes é que, a política de estimativa pode ser determinística (por exemplo ε-*greedy*), enquanto a política de comportamento pode seguir uma política que garante experimentar todas as acções possíveis.

Mas, para que seja possível termos essas duas políticas distintas, é necessário utilizar o método de avaliar uma política seguindo outra. Para avaliar uma política π seguindo uma política π' é necessário que, pelo menos ocasionalmente, as acções realizadas por π sejam também realizadas por π', isto é, é necessário que $\pi(s,a) \geq 0$ implique que $\pi'(s,a) \geq 0$. Sendo $p_i(s)$, $p'_i(s)$, num episódio i, as probabilidades de executar uma sequência de estados e acções partindo de um estado s, dadas as políticas π e π' e $R_i(s)$ a recompensa observada, então a função de valor de estado é dada por:

$$V(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i}{p'_i} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i}{p'_i}} \tag{A.23}$$

Embora o método *Monte Carlo* seja um método *model-free*, pelo que não é possível ter conhecimento de p_i e p'_i, a proporção $\frac{p_i}{p'_i}$ pode ser determinada sem qualquer conhecimento da dinâmica do ambiente, dependendo, apenas, das duas políticas utilizadas da seguinte forma:

$$p_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k)\mathcal{P}^{a_k}_{s_k s_{k+1}}$$

$$\frac{p_i(s_t)}{p'_i(s_t)} = \frac{\prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k)\mathcal{P}^{a_k}_{s_k s_{k+1}}}{\prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k)\mathcal{P}^{a_k}_{s_k s_{k+1}}} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}$$

Conhecendo isto, é possível conceber o algoritmo do método *off-policy Monte Carlo Control* descrito na tabela A.4, em que $N(s, a)$ e $D(s, a)$ correspondem, respectivamente, ao numerador e dominador da função de valor de estado-acção $Q(s, a)$.

Tabela A.4: Pseudo-código do algoritmo ε-*soft* do método *off-policy Monte Carlo Control*.[30]

Inicializar, $\forall\ s \in \mathcal{S}, a \in \mathcal{A}(s)$:
- $Q(s, a) \leftarrow$ arbitrário
- $N(s, a) \leftarrow 0$
- $D(s, a) \leftarrow 0$
- $\pi(s) \leftarrow$ política determinística ε-*soft* arbitrária

Repetir infinitamente:
- Seleccionar π' usa-la para gerar um episódio:
  - $s_0, a_0, r_1, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$
- $\tau \leftarrow$ última vez em que $a_\tau \neq \pi(s_\tau)$
- Para cada (s, a) visitado no episódio, depois de τ:
  - $t \leftarrow$ o tempo da primeira ocorrência, depois de τ de (s, a)
  - $\omega \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k, a_k)}$
  - $N(s, a) \leftarrow N(s, a) + \omega R_t$
  - $D(s, a) \leftarrow D(s, a) + \omega$
  - $Q(s, a) \leftarrow \frac{N(s,a)}{D(s,a)}$
- Para cada $s \in \mathcal{S}$ do episódio:
  - $\pi(s) \leftarrow \arg\max_a Q(s, a)$

A.3.4 TD(0)

O método da aprendizagem por diferença temporal, ou aprendizagem TD (*Temporal Diference learning*), é considerado um método central e original do *reinforcement learning*. Este, combina o método de *Monte Carlo* com a programação dinâmica, ou DP (*Dynamic Programming*). Tal como nos métodos *Monte Carlo*, também os de diferença temporal são *model-free*, podendo-se aprender através da experiência directa com o ambiente, sem a necessidade de conhecer a sua dinâmica. E os métodos de diferença temporal também estimam e actualizam, baseando-se, em parte, noutras estimativas aprendidas anteriormente, sem esperar pelo final dos episódios, à semelhança da programação dinâmica.

Enquanto que, nos métodos de *Monte Carlo*, é necessário esperar pelo final de cada episódio para receber a recompensa num passo t, determinando o incremento da função de valor de estado $V(s_t)$, nos métodos de diferença temporal, apenas é necessário esperar pelo próximo passo. Isto é, no passo $t+1$ é observada a recompensa $r+1$, é estimado $V(s_{t+1})$ e é actualizado $V(s_t)$. O método de diferença temporal mais simples é conhecido como TD(0) e a função de valor de estado é dada por:

$$V(s_t) \leftarrow V(s_t) + \alpha\Big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big] \tag{A.24}$$

onde α é uma constante, designada por parâmetro de coeficiente de aprendizagem. O algoritmo descrito na tabela A.5 especifica o método TD(0).

Tabela A.5: Pseudo-código do algoritmo TD(0).[30]

Inicializar:
$V(s) \leftarrow$ arbitrário
$\pi(s) \leftarrow$ política a ser avaliada

Repetir para cada episódio:
Inicializar s
Repetir para cada passo do episódio até s terminar:
$a \leftarrow$ acção escolhida por π para o estado s
Executar a acção a
Observar a recompensa r e o próximo estado s'
$V(s) \leftarrow V(s) + \alpha\big[r + \gamma V(s') - V(s)\big]$
$s \leftarrow s'$

A.3.5 *Q-learning*

O *Q-learning*[33], um método *off-policy*, é o método mais conhecido, e um dos mais importantes, do *reinforcement learning*. Se as recompensas e acções forem estocásticas, para poder-se modelar a incerteza num sistema (que resulta em parte da dinâmica do ambiente não controlável por um agente), são consideradas a probabilidade de obter uma recompensa $P(r_{t+1} \mid s_t, a_t)$ e a probabilidade de visitarmos o próximo estado $P(s_{t+1} \mid s_t, a_t)$.[4] Se considerarmos, por exemplo,

um agente como sendo um robô que, devido a variações de temperatura, por vezes se desvia da trajectória que escolheu seguir, ou avança menos do que esperava, então temos:

$$Q(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{A.25}$$

Sendo assim, não é possível fazer uma atribuição directa do valor da recompensa esperada, visto que, para o mesmo par de estado-acção, é possível que o robô obtenha recompensas distintas ou próximos estados diferentes.[4] O que no Q-learning se faz é manter uma média dos valores de Q:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \Big[r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t) \Big] \tag{A.26}$$

Neste caso, a função de valor de estado-acção Q aprendida, aproxima directamente a função de valor de estado-acção óptima Q^*, independentemente da política seguida. O que simplifica significativamente o algoritmo e permite uma convergência rápida. A política continua a determinar qual o próximo par de estado-acção a ser actualizado, no entanto, a actualização contínua dos pares de estado-acção é tudo o que é necessário para corrigir a convergência para Q^*. O procedimento do algoritmo de *Q-learning* é descrito na tabela A.6.

Tabela A.6: Pseudo-código do algoritmo *Q-learning*.[30]

Inicializar:
$Q(s, a) \leftarrow$ arbitrário

Repetir para cada episódio:
Inicializar s
Repetir para cada passo do episódio até s terminar:
$a' \leftarrow$ tendo s', baseado em Q, usando uma política (por ex.: ε-*greedy*)
Executar a acção a
Observar o próximo estado s' e a recompensa r
$Q(s, a) \leftarrow Q(s, a) + \alpha \Big[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \Big]$
$s \leftarrow s'$

O *Q-learning* tornou-se num dos métodos de *reinforcement learning* mais populares, devido ao facto de ser um algoritmo "insensível à exploração", ou seja, apesar de existir a problemática *exploration-exploitation*, a convergência de aprendizagem não depende da política de exploração.[16]

A.3.6 SARSA - *State-Action-Reward-State-Action*

O método SARSA, que significa a sequência estado-acção-recompensa-estado-acção, é um algoritmo *on-policy* para a aprendizagem TD. Muito semelhante ao método de aprendizagem *Q-learning*, a grande diferença entre eles é o facto de, no método SARSA, a recompensa máxima para o próximo passo não ser, necessariamente, usada para actualizar o valor de Q. Em vez disso, uma nova acção a' é escolhida, após a observação da recompensa r no novo estado s', usando a mesma política que, no estado original s, determinou a acção original a. A função de valor de estado-acção, para o método SARSA, é dada por:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big] \tag{A.27}$$

Esta actualização ocorre após a transição de um estado s_t não terminal para um estado s_{t+1}, e se, este último, for terminal, então $Q(s_{t+1}, a_{t+1}) = 0$. O algoritmo do método SARSA, descrito na tabela A.7 é de fácil concepção. Tal como em qualquer método *on-policy*, a função de valor de estado-acção Q^π é continuamente estimada para a política de comportamento π e, ao mesmo tempo, altera π avidamente, isto é, por forma a escolher a melhor acção, respectivamente a Q^π.

Tabela A.7: Pseudo-código do algoritmo SARSA.[30]

Inicializar:
$Q(s, a) \leftarrow$ arbitrário

Repetir para cada episódio:
Inicializar s
$a' \leftarrow$ tendo s', baseado em Q, usando uma política (por ex.: ε-*greedy*)
Repetir para cada passo do episódio até s terminar:
Executar a acção a
Observar o próximo estado s' e a recompensa r
$a' \leftarrow$ tendo s', baseado em Q, usando uma política (por ex.: ε-*greedy*)
$Q(s, a) \leftarrow Q(s, a) + \alpha\Big[r + \gamma Q(s', a') - Q(s, a)\Big]$
$s \leftarrow s'$
$a \leftarrow a'$

A convergência do algoritmo do método SARSA depende da natureza da dependência da política π em relação a Q, como por exemplo ε-*greedy* ou ε-*soft*. O método SARSA converge para uma política óptima π^* e uma função de valor de estado-acção Q^*, enquanto os pares de estado-acção forem visitados infinitamente e a política convergir para uma política ávida. Esta última condição pode ser alcançada com políticas ε-*greedy* usando $\varepsilon = \frac{1}{t}$.[30]

A.4 Métodos *Model-based*

Nos métodos *model-based*, a dinâmica do ambiente é conhecida, isto é, os parâmetros do modelo do ambiente $P(r_{t+1} \mid s_t, a_t)$ e $P(s_{t+1} \mid s_t, a_t)$. Assim sendo, não existe a necessidade de explorar os estados e acções e é possível obter a função de valor de estado óptima e a política óptima directamente usando programação dinâmica, ou DP (*Dynamic Programming*). A função de valor de estado óptima V^* é única e é a solução para o sistema de equações A.11 e A.13.[4]

Uma vez tendo a função de valor de estado óptima, a política óptima é escolher a acção que maximiza o valor no próximo passo:

$$\pi^*(s_t) = \arg\max_{a_t} \left(E\Big[r_{t+1} \mid s_t, a_t\Big] + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1} \mid s_t, a_t) V^*(s_t) \right) \tag{A.28}$$

Nesta secção descrevem-se dois tipos de métodos *model-based* de programação dinâmica: *Policy Iteration* e *Value Iteration*.

A.4.1 *Policy Iteration*

O método *model-based* de programação dinâmica *Policy Iteration* é, como o próprio nome sugere, um algoritmo iterativo onde uma política π é repetidamente aperfeiçoada até se encontrar a política óptima π^*.[4] Tendo uma política π, usando a função de valor V^π, sido aperfeiçoada para uma política π', é possível então calcular $V^{\pi'}$ e aperfeiçoar, mais uma vez, uma política π' para uma política π''. Seguindo este pensamento obtemos uma sequência de aperfeiçoamento de política e funções de valor:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^*, \tag{A.29}$$

tal como no método *Monte Carlo Control*, também no *Policy Iteration* começa-se com uma política π_0, a partir da qual são realizados passos de avaliação política $\xrightarrow{E}$ e passos de aperfeiçoamento de política $\xrightarrow{E}$ alternadamente, até ser encontrada uma política óptima π^* e uma função de valor óptima V^*.

No método *Policy Iteration*, cujo algoritmo pode ser visto na tabela A.8, a avaliação de política, ela própria uma computação iterativa, inicia a partir de uma função de valor para a política anterior, o que faz com que aumente significativamente a rapidez da sua convergência.[30] Tendo em conta a função de valor, que é calculada resolvendo as equações lineares, é verificado se é possível aperfeiçoar a política. Quando já não for possível aperfeiçoar a política, então essa é a política óptima.[4]

Tabela A.8: Pseudo-código do algoritmo *Policy Iteration*.[30]

1. Inicializar $\forall\ s \in \mathcal{S}$:
   $V(s) \in \Re \leftarrow$ arbitrário
   $\pi(s) \in \mathcal{A}(s) \leftarrow$ arbitrário

2. Avaliação de política:
   Repetir, enquanto $\Delta < \theta$ (um número pequeno positivo):
   $\Delta \leftarrow 0$
   Para cada $s \in \mathcal{S}$:
   $\nu \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')\right]$
   $\Delta \leftarrow \max\left(\Delta, |\nu - V(s)|\right)$

3. Aperfeiçoamento de política:
   política-estável $\leftarrow$ verdadeiro
   Para cada $s \in \mathcal{S}$:
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s')\right]$
   Se $b \neq \pi(s)$, então:
   política-estável $\leftarrow$ falso
   Se política-estável, então:
   Pára
   Caso contrário, voltar para 2.

A.4.2 *Value Iteration*

O *Value Iteration*, à semelhança do *Policy Iteration*, é um algoritmo iterativo *model-based* de programação dinâmica. No entanto, enquanto que no *Policy Iteration* o aperfeiçoamento de política é iterativo, o que faz com que apenas convirja para V^* no limite, no *Value Iteration* os passos de avaliação de política podem ser interrompidos sem comprometer a garantia de convergência do *Policy Iteration*. E assim, é possível obter uma política óptima π^*, que, uma vez que apenas são tidas em conta as acções que maximizam V^π, pode convergir mesmo antes de se obter V^*.[4] Combinando aperfeiçoamento de política com passos incompletos de avaliação de política, o *Value Iteration* pode ser escrito da seguinte forma:

$$\begin{aligned} V_{k+1}(s) &= \max_a E\left\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, a_t = a\right\} \\ &= \max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V_k(s')\right],\ \forall\ s \in \mathcal{S} \end{aligned} \tag{A.30}$$

onde k é o número da iteração. Tal como na avaliação de política, no método *Value Iteration*, idealisticamente é necessário um número infinito de iterações para convergir exactamente para V^*. Na prática, o valor óptimo V^* é encontrado quando a diferença máxima entre duas funções de valor consecutivas é menor que um valor pequeno θ, isto é: $\max_{s \in \mathcal{S}} \left|V_{k+1}(s) - V_k(s)\right| \leq \theta$.[4]

Tabela A.9: Pseudo-código do algoritmo *Value Iteration*.[30]

Inicializar $\forall\ s \in \mathcal{S}$:

$V(s) \leftarrow$ arbitrário (por ex.: $V(s) = 0$)

Repetir, até $\Delta < \theta$:

Para cada $s \in \mathcal{S}$:

$\Delta \leftarrow 0$

$\nu \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

$\Delta \leftarrow \max\left(\Delta, |\nu - V(s)|\right)$

Resulta uma política determinística π, tal que:

$$\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$$

O algoritmo do método *Value Iteration*, descrito na tabela A.9, combina eficazmente um ciclo de avaliação de política com um ciclo de aperfeiçoamento de política, em cada iteração. Uma rápida convergência pode ser alcançada, entrepondo vários ciclos de avaliação de política entre cada aperfeiçoamento de política.[30]

Apêndice B

Desenhos técnicos

Apêndice C

Placas de circuito impresso

C.1 Traçado do circuito impresso das Placas

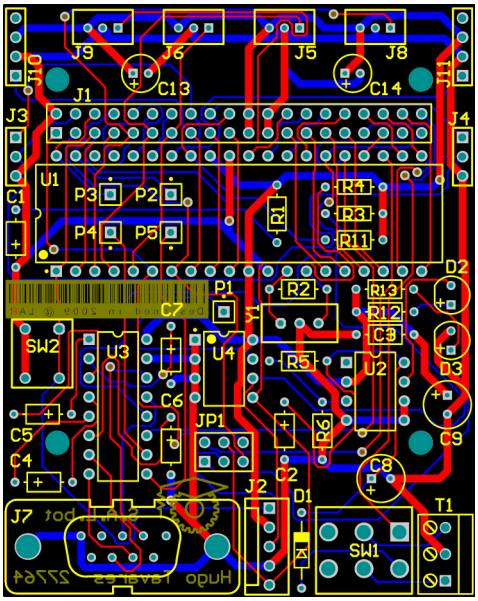


Figura C.1: Traçado do circuito impresso da placa principal do SALbot.

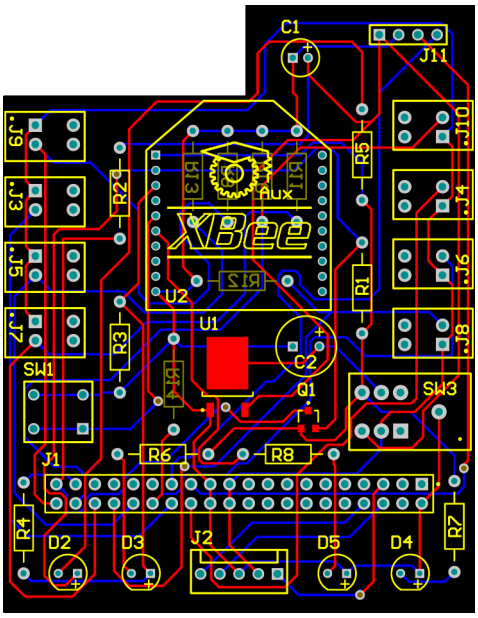


Figura C.2: Traçado do circuito impresso da placa auxiliar do SALbot.

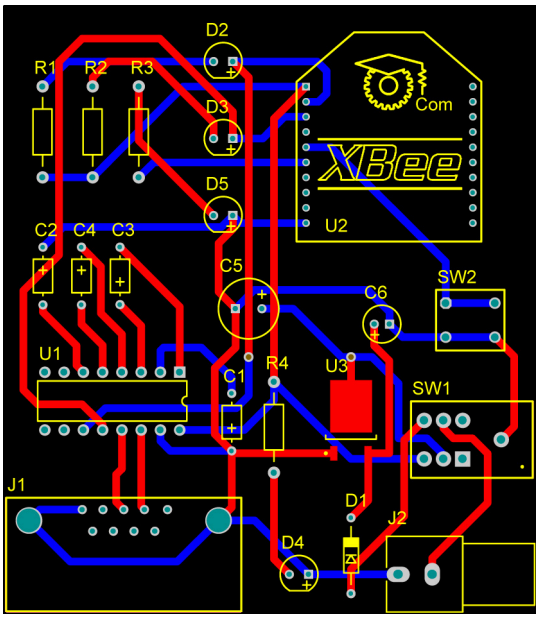


Figura C.3: Traçado do circuito impresso da placa de interface de comunicação.

C.2 Esquemas eléctricos

C.3 Imagens das placas

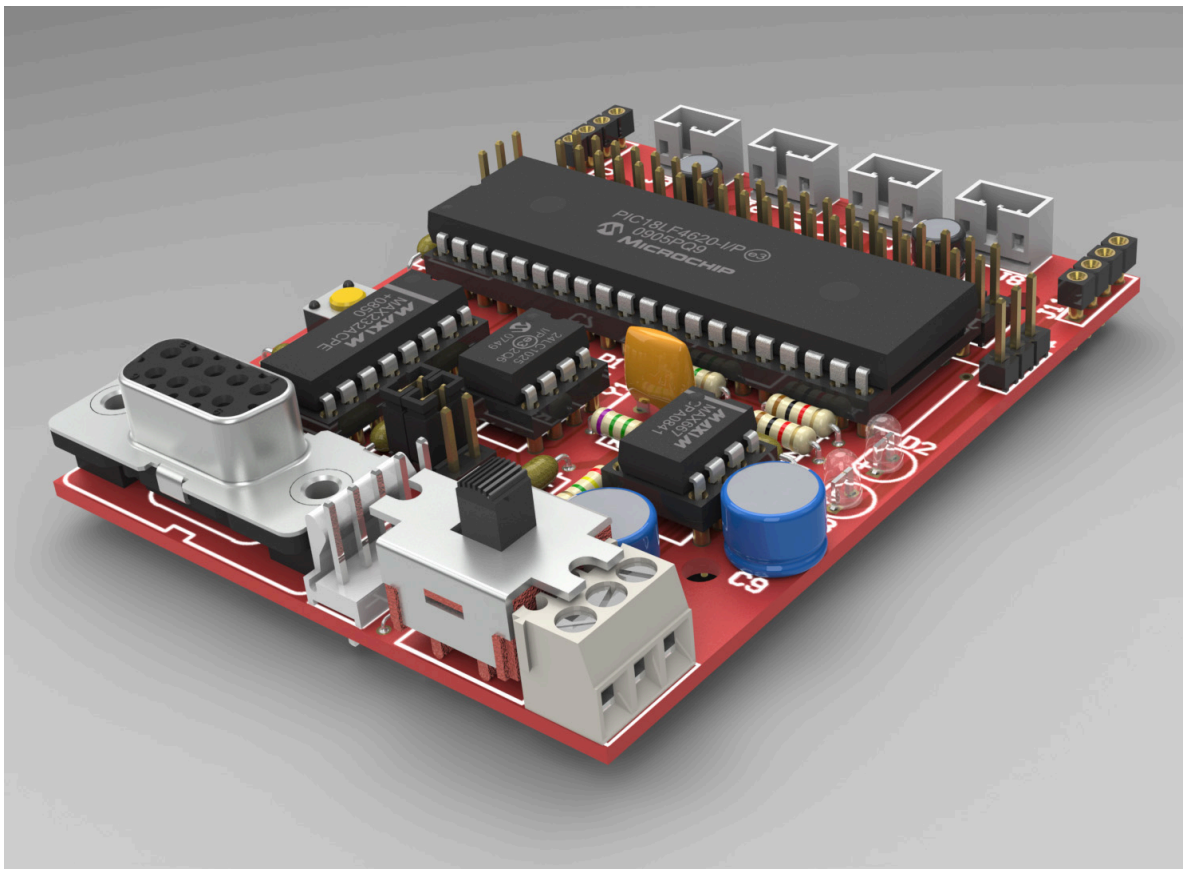


Figura C.4: Imagem foto-realista da placa principal do SALbot.

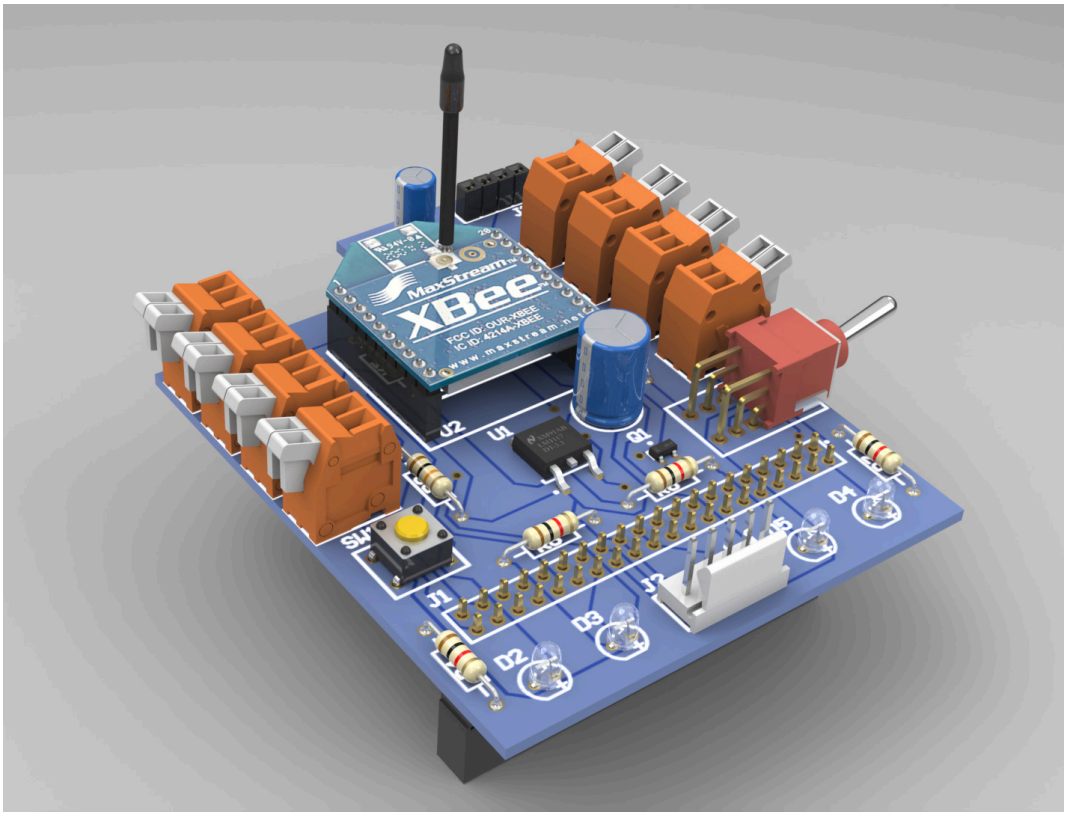


Figura C.5: Imagem foto-realista da placa auxiliar do SALbot.

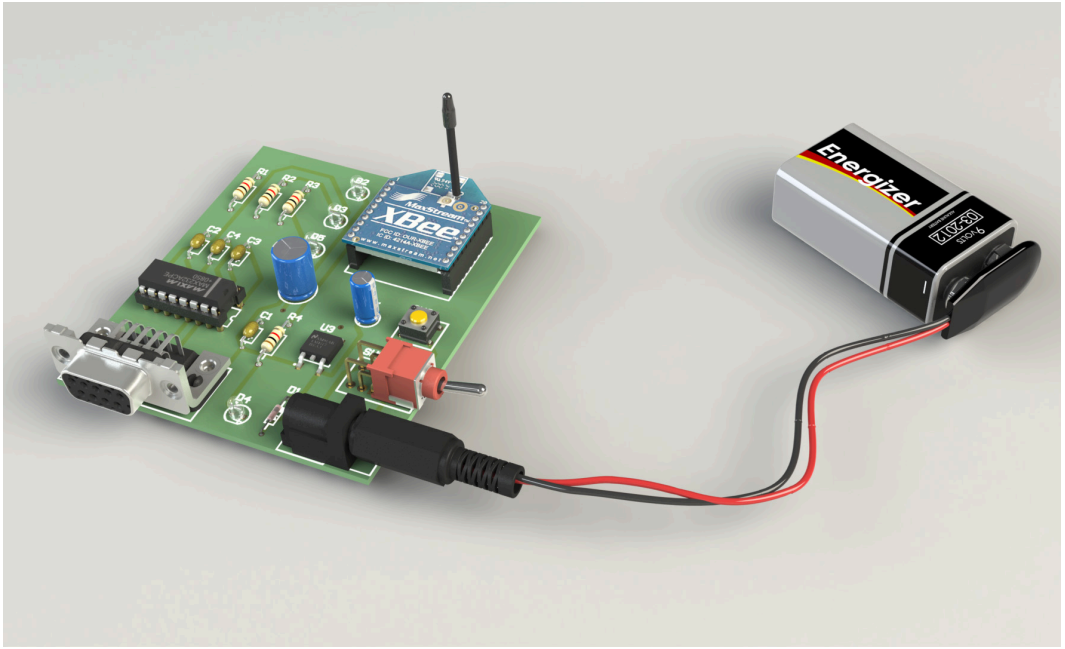


Figura C.6: Imagem foto-realista da placa de interface de comunicação.

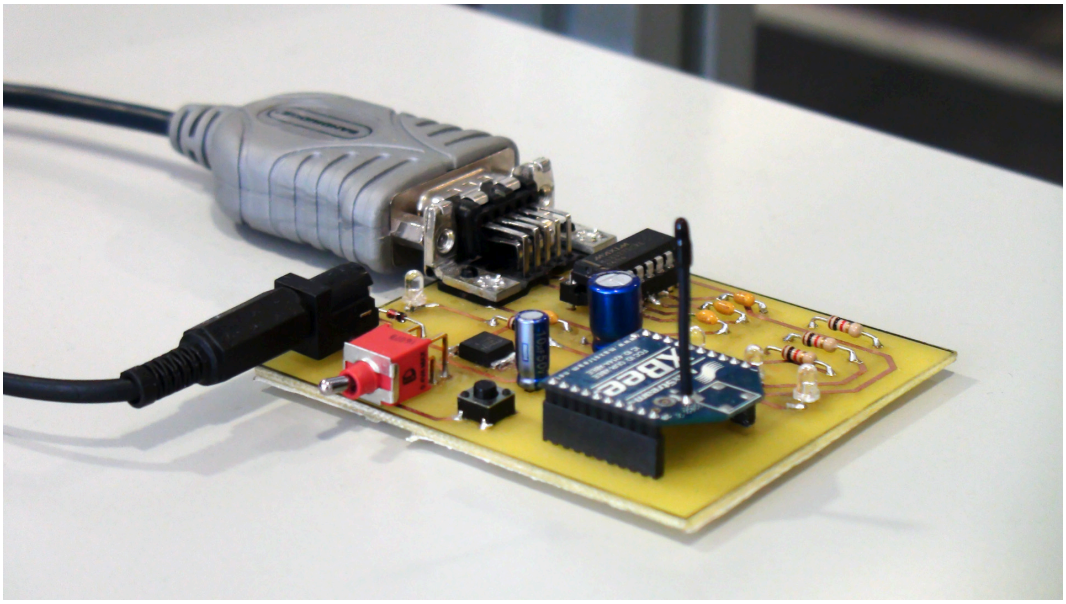


Figura C.7: Imagem da placa de interface de comunicação.

Apêndice D

Imagens dos robôs

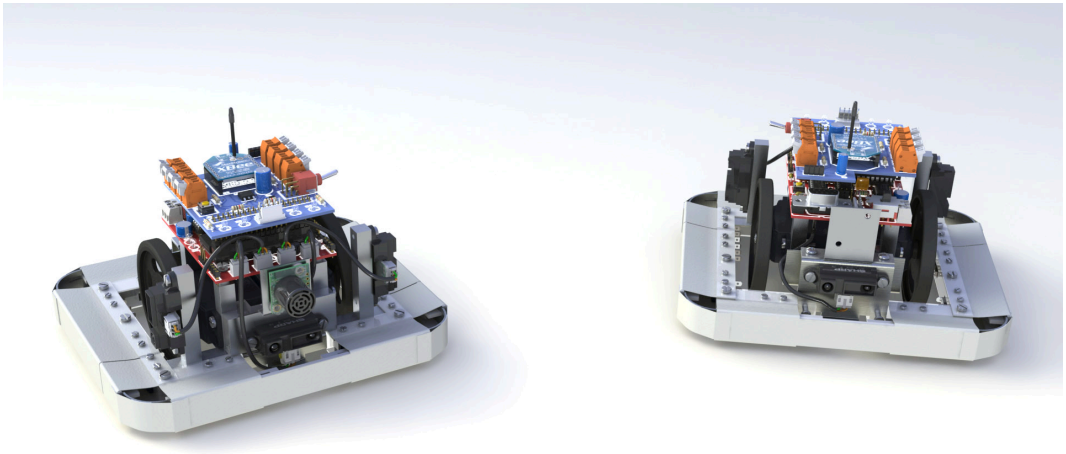


Figura D.1: Imagem foto-realista dos SALbot.

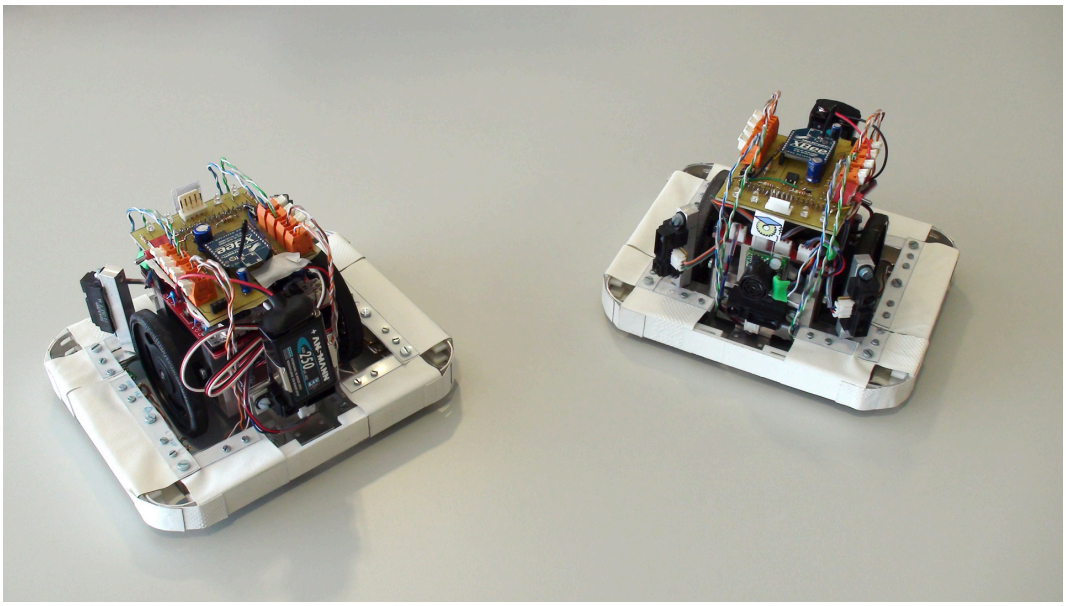


Figura D.2: Imagem dos SALbot.

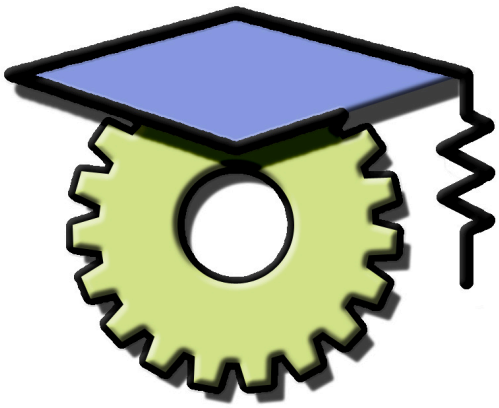


Figura D.3: Logótipo do SALbot.

Referências

[1] Futaba standard servos. `http://www.futaba-rc.com/servos/servos.html`, online on April 2010.

[2] Mark III complete kit. `http://www.junun.org/MarkIII/Info.jsp?item=1`, online on April 2010.

[3] Mobilerobots, research & university robots, software & accessories. `http://www.activrobots.com/ROBOTS/`, online on April 2010.

[4] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[5] Lipson H Bongard J., Zykov V. Automated synthesis of body schema using multiple sensor modalities. In *the 10th Int. Conference on Artificial Life (ALIFE X)*, pages 220–226, 2006.

[6] Thomas Bräunl. *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[7] Mark Elshaw Cornelius Weber and Norbert Michael Mayer, editors. *Reinforcement Learning*. I-Tech Education and Publishing, 2008.

[8] Peter Dayan and Yael Niv. Reinforcement learning: the good, the bad and the ugly. *Current opinion in neurobiology*, 18(2):185–196, April 2008.

[9] A. T. De Almeida, M. Thoma, and O. Khatib, editors. *Autonomous Robotic Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[10] Digi International, Digi International Inc. 11001 Bren Road East Minnetonka, MN 55343. *XBee® / XBee-PRO® RF Modules - 802.15.4*, v1.xEx edition, September 2009.

[11] A. Eriksson, G. Capi, and K. Doya. Evolution of meta-parameters in reinforcement learning algorithm. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[12] Eiji Uchibe Genci Capi and Kenji Doya. Selection of neural architecture and the environment complexity. In *the 3rd International Symposium on Human and Artificial Intelligent Systems*, 2002.

[13] Stan Gibilisco. *Concise Encyclopedia of Robotics*. McGraw-Hill, 2002.

[14] Joseph L. Jones and Anita M. Flynn. *Mobile robots: inspiration to implementation*. A. K. Peters, Ltd., Natick, MA, USA, 1993.

[15] Hod Lipson Josh Bongard, Victor Zykov. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118 – 1121, 2006.

[16] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH*, 4:237–285, 1996.

[17] Maxbotics® Inc. *LV-MaxSonar® - EZ0™ Data Sheet*, v3.0c edition, January 2007.

[18] Merlin Systems Corp. Ltd. *Miabot PRO BT v2 User Manual*, rev. 1.3 edition, Mars 2005.

[19] Microchip Technology Inc. *24AA1025/24LC1025/24FC1025 1024K I^2C CMOS Serial EEPROM Data Sheet*, 2005.

[20] Takashi Minato, Yuichiro Yoshikawa, Tomoyuki Noda, Shuhei Ikemoto, Hiroshi Ishiguro, and Minoru Asada. Cb^2: A child robot with biomimetic body for cognitive developmental robotics. 2007.

[21] F. Mondada, E. Franzi, and A. Guignard. The Development of Khepera. In *Experiments with the Mini-Robot Khepera*, HNI-Verlagsschriftenreihe, Heinz Nixdorf Institut, pages 7–14, 1999.

[22] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a Robot Designed for Education in Engineering. In Paulo J.S. Gonçalves, Paulo J.D. Torres, and Carlos M.O. Alves, editors, *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59–65, Portugal, 2009. IPCB: Instituto Politécnico de Castelo Branco.

[23] Nicola Nosengo. The bot that plays ball. *Nature*, Vol. 460(No. 7259):pp. 1076–1078, August 2009.

[24] Christopher JCH Watkins Peter Dayan. *Reinforcement Learning*. Encyclopedia of Cognitive Science, England: MacMillan Press, 2001.

[25] Gabriela Serban. A reinforcement learning intelligent agent, 2001.

[26] SHARP. *GP2D120 Datasheet*, May 2006.

[27] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008.

[28] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Books. The MIT Press, 2004.

[29] Frederic Murphy Steven O. Kimbrough, Ming Lu. *Formal Modelling in Electronic Commerce*, chapter Learning and Tacit Collusion by Artificial Agents in Cournot Duopoly Games, pages 477–492. International Handbooks on Information Systems. Springer Berlin Heidelberg, 2005.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[31] Fumihide Tanaka, Aaron Cicourel, and Javier R. Movellan. Socialization between toddlers and robots at an early childhood education center. *Proceedings of the National Academy of Sciences*, Vol. 104(No. 46):pp. 17954–17958, November 2007.

[32] N.G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, J. Santos-Victor, M.C. Carrazzo, F. Becchi, and D.G. Caldwell. icub: the design and realization of an open humanoid platform for cognitive and neuroscience research. *International Journal of Advanced Robotics*, Vol. 21(No. 10):pp. 1151–75, 2007.

[33] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge,England, 1989.