Universidade de Aveiro Departamento de Engenharia Mecânica
2009

Hugo Emanuel
Fonseca Tavares

Concepção e desenvolvimento de pequenos robôs para auto-aprendizagem

DOCUMENTO
PROVISÓRIO

Universidade de Aveiro Departamento de Engenharia Mecânica
2009

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.

DOCUMENTO PROVISÓRIO

Palavras-chave

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

Resumo

O objectivo deste trabalho é projectar e desenvolver pequenos robôs autónomos preparados para aprendizagem de comportamentos de locomoção. Uma vez que existem vários tipos de sensores e outros dispositivos que podem ser integrados num robô orientado para aprendizagem, pretendeu-se projectar um sistema modular que possibilite a adição de dispositivos futuros.

Dos diversos tipos de *machine learning* existentes o *reinforcement learning* é, sem dúvida, um dos mais interessantes. Uma vez que, o *reinforcement learning* é um tipo de aprendizagem não supervisionada, isto implica que seja o robô a julgar o resultado das suas acções, julgamento esse que resulta em "recompensas", positivas ou negativas, consoante o as acções serem, boas ou más respectivamente.

Para que o robô possa fazer esse julgamento deve estar dotado de dispositivos que lhe permitam dar informações sobre o meio que o rodeia. A maioria dos pequenos robôs existentes possui apenas sensores de distância, sensores de movimento, alguns, câmara de video, etc... No entanto são poucos os que possuem redundância sensorial e sensores de contacto. Qualquer pessoa que, resultante de qualquer acção, tenha um contacto físico com o meio, sabe que não deve tomar aquela acção novamente naquelas circunstâncias. Pois, sentiu através da dor, oriunda do contacto físico, que foi uma má acção.

Para que os robôs aprendam eficazmente é necessário que executem más acções, pois, é com as más acções que o robô irá descobrindo o que, realmente, são boas acções. Nesse sentido, procurou-se projectar robôs robustos. Mas, ao mesmo tempo, flexíveis o suficiente para não danificarem o meio que os rodeia.

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

Abstract

The main goal of this project is to design and develop small autonomous robots able to learn movement behaviors. Since there are many types of sensors and other devices that might be integrated into a robot directed to learning, was intended to design a modular system that allows the addition of future devices.

Among the many existentes types od machine learning the reinforcement learning is, without any doubt, one of the most interesting. Since, the reinforcement learning is a unsupervised learning, that means that will be the robot who judge his actions, witch consists in "rewards", positive or negative, as the actions are, good or bad respectively.

For the robot can make that judgement it must be gifted of devices witch allow it to give information about the environment. The most small robots existent only possess only distance and movement sensors, some video cameras, etc... On the other hand, few are who have sensorial redundancy and contact sensors. Any person that, as a result of an action, have a physical contact with the environment knows that he must not take that action again under those circumstances. Because he felt through the pain, from the physical contact, that he made a bad action.

For the robots can learn efficiently it's necessary they take bad actions, because, it's with the bad actions that the robot will discover what, really, good actions are. In that line of thought, we tried to design strong robots. But, as well, flexible enough to don't damage the environment.

Conteúdo

Lista de Figuras

Lista de Tabelas

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, nomeadamente raciocinar, tomar opções e, até quem sabe, ter sentimentos. Existem vários sub-domínios da IA, e é muito comum, em pesquisas na área da robótica, interligar algumas desses sub-domínios. Por exemplo, existem algoritmos de *Machine learning* que recorrem a redes neuronais. No caso deste trabalho, uma vez que se trata de robôs para aprendizagem, naturalmente que serão abordadas as áreas de *Machine learning*. Uma vez que se trata de robôs autónomos, poderia ser abordada, também, a área de planeamento autónomo, no entanto não é esse o objectivo deste trabalho, ainda assim pode ser uma área a ser abordada em trabalhos futuros.

Nas últimas décadas, o *Machine learning* tem sido uma área muito investigada, o facto de a máquina aprender as tarefas que tem de realizar em determinas situações pode 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 requerer 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 uma tarefa árdua. 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 fornecer exemplos à máquina, ou julgar as acções que a máquina tomou, para aprender, é a própria máquina que julga o resultado das acções que realiza.

O facto de ser a máquina a julgar, por ela própria, o resultado das suas acções pode ser vantajoso. Um ser humano, enquanto criança, é ensinado pelos pais e/ou outras pessoas (nada mais nada menos do que 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, isto devido, em parte, por aquilo lhes é ensinado, é certo, mas maioritariamente por aquilo que aprendem por eles próprios. É a capacidade para aprender que torna cada ser humano único. Mas também permite-lhes 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, ou são demasiado dispendiosos, ou não possuem uma plataforma modular que permita a adaptação de mais sensores ou outro tipo de dispositivos que possam ser uma mais valia para a 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 os robôs industriais, estima-se, representam 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, liberta os operários da realização de tarefas monótonas para outro tipo de tarefas. Porem, este tipo de robôs têm a limitação obvia da falta de mobilidade.

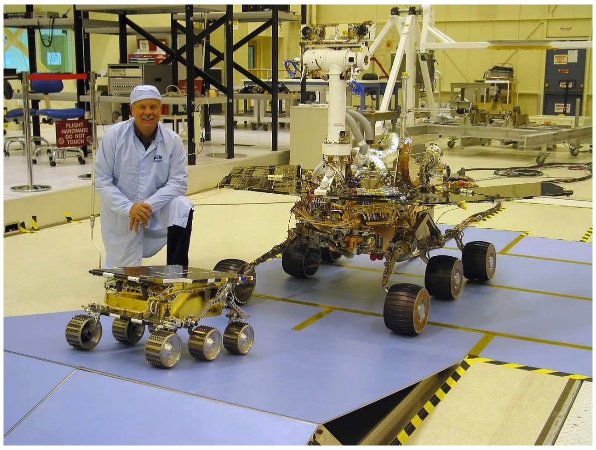


Figura 1.1: Robôs exploradores de Marte: Spirit e Sojourner da NASA

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 dos robôs móveis das décadas seguintes. A década de 1980 prosperou com projectos de robôs móveis, e tão cedo se sentiu a necessidade de lidar com a realidade do mundo físico, que apareceram problemas que estimularam novas direcções de pesquisa. Actualmente distanciando-se do conceito original, no qual o robô era apenas uma aplicação de técnicas de IA[26].[1]

Um robô móvel implica que este 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 se poderem movimentar pelo meio em que se encontra, podendo executar as suas tarefas em vários locais.

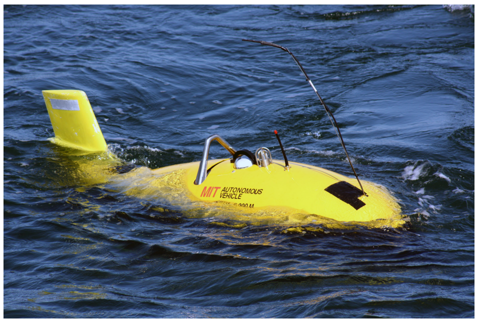
A necessidade da utilização de dispositivos robóticos em ambientes hostis, como por exemplo em Marte, impulsiona a implementação de sistemas de locomoção cada vez mais incomuns (ver figura 1.1). Os robôs "teleoperados"[2] têm ganho popularidade em terrenos perigosos e/ou inabitáveis (ver figuras 1.3b, 1.2b, 1.2d, 1.3a e 1.2c). 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

[1] Este parágrafo foi transcrito!
[2] não sei se esta palavra existe

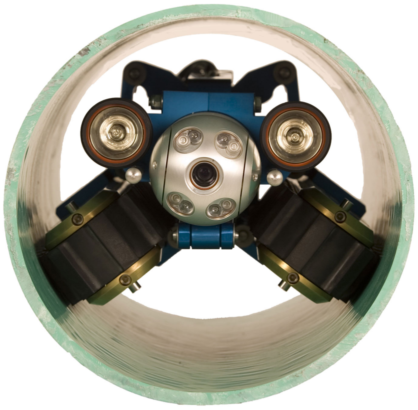
do robô da Plustech (ver figura 1.2a), o operador apenas tem de indicar a direcção para a qual ele quer que o robô se desloque, enquanto que o robô possui um controlo automático da coordenação das pernas. Na figura 1.2c pode-se ver um robô aquático que, apesar de possuir motores que lhe permitem uma 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)



(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.

Mas não existem apenas robôs para serem operados em locais onde o Homem não pode ir, existem, também, robôs que partilham o mesmo espaço habitado pelos humanos (figuras 1.4a, 1.5a, 1.4c, 1.5b e 1.5c). Estes robôs não são úteis por causa da sua mobilidade, mas sim por causa da sua autonomia. De facto, 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 complexo forem as tarefas que um robô tem de executar, maior pode ser a sua autonomia e, normalmente, recorrem ao uso de inteligência artificial para a execução dessas tarefas.

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 continua 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 assim que normalmente funcionam os robôs insecto.[13]

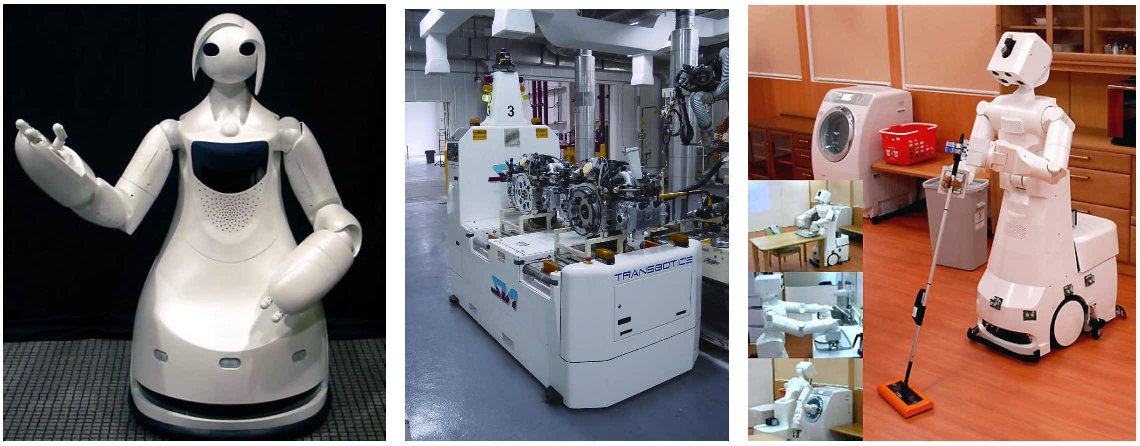
(a) (b)

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.

Robôs AGV (*Autonomous Ground Vehicle*), distribuem peças por vários pontos de uma linha de montagem autonomamente. Este tipo de robôs têm 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). Estes robôs recorrem à emissão de feixes laser para navegarem autonomamente, para isso, é apenas necessário colocar reflectores nas paredes, colunas ou máquinas da fábrica, para que reflictam os fixes laser de volta ao robô, dando-lhe, assim, a posição dos obstáculos.

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 por vácuo, aspira 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.

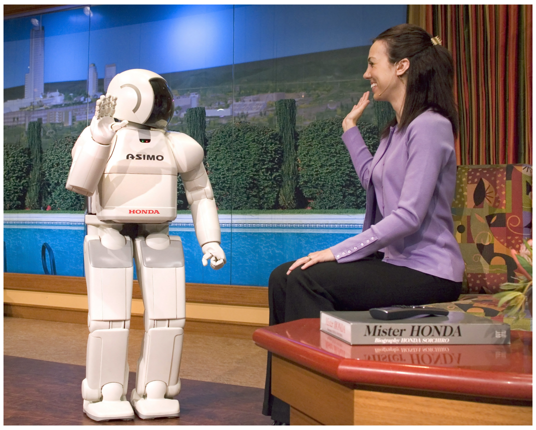
Robôs de estudo como o ASIMO (figura 1.5a), o Robomaid (figura 1.4c) ou a Robina (figura 1.4a), dão uma ideia de como serão os robôs no futuro. Mas é também graças a estes, e outros, protótipos que a robótica vem evoluindo ao longo dos anos. De facto o ASIMO, actualmente o mais avançado robô humanoide 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, caminhando para a frente e para trás, virando enquanto caminha e até subir e descer 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.



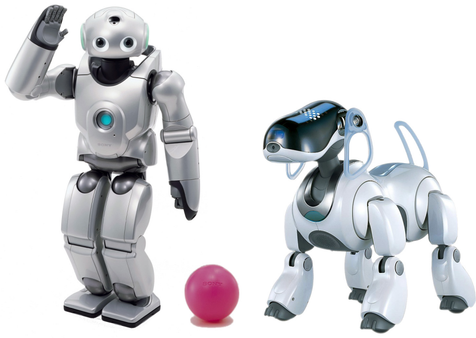
(a) (b) (c)

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.

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. Alem 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 quanto 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 se concluíram vários 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[30]. 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 outro com as mesmas caracte-



(a)



(b)

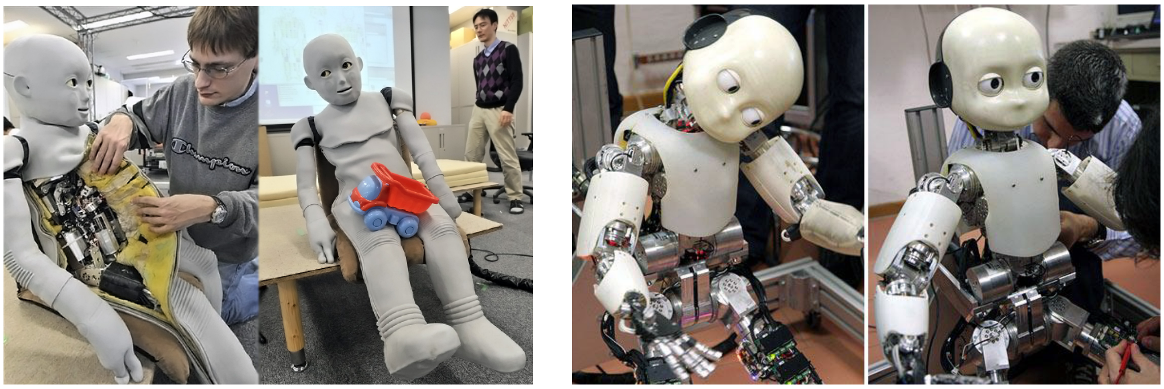
(c)

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

rísticas, vários tipos de aplicações, nomeadamente na ajuda com 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.

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*[3] 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,

[3] verificar o uso do estrangeirismo



(a) (b)

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

expressões e gestos, nos últimos dois anos o CB^2 tem vindo 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 a 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[31]. 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 mão 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);

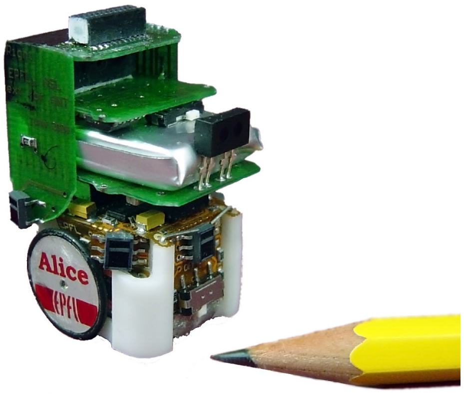
- Lagartas (figuras 1.2d, 1.3a e 1.3b);

1.3.2 Pequenos Robôs móveis para aprendizagem

Não sendo tão avançados, a nível de *hardware*, como os tipos robôs da área da aprendizagem anteriormente mencionados, os pequenos robôs para aprendizagem serão sempre utilizados para investigação nesta área. Pois, como estes robôs, não sendo 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 os robôs em cima de bancadas laboratoriais ou em espaços mais reduzidos, devido às suas reduzidas dimensões.

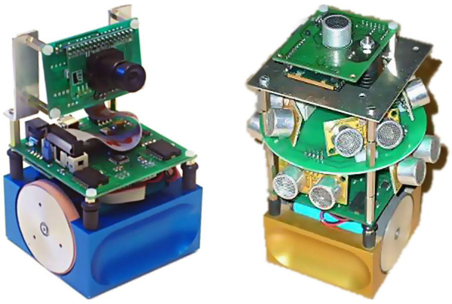
Existem várias plataformas *standard* deste tipo de robôs prontas para programar que, normalmente, são adquiridas para investigação na área. Como por exemplo o Khepera (figura 1.7a), o Alice (figura 1.7b), o e-puck (figura 1.7c), o Miabot Pro (figura 1.7d), o AmigoBot™ (figura 1.8a) e o pioneer 3-DX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

(a)



(b)

(c)



(d)

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.

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

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. Têm vários módulos de expansão, nomeadamente uma câmara digital linear de 102 pixels, sensores tácteis, ZigBee®, uma garra e um painel solar.

(a)

(b)

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

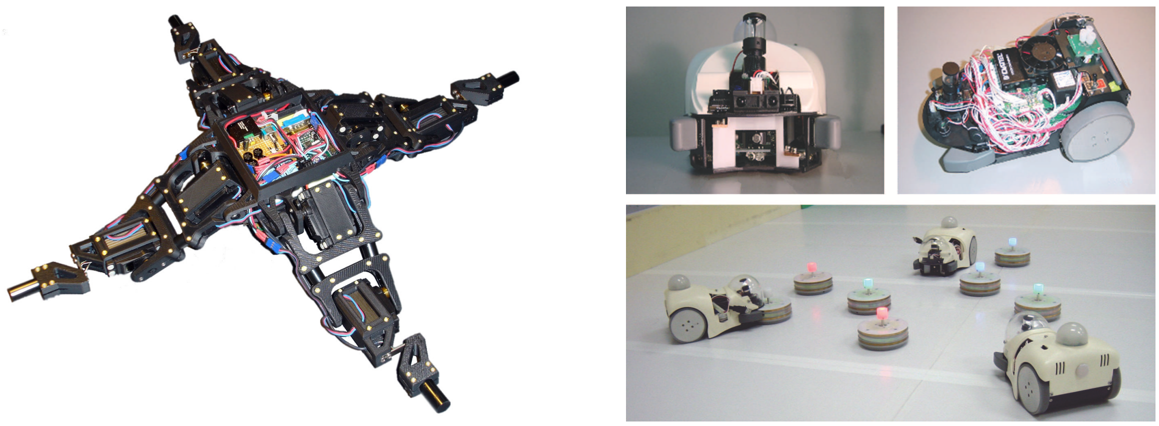
Criado em 2004 no ASL da EPFL, pelos mesmos criadores do Khepera, o e-puck (figura 1.7c) foi desenvolvido para poder operar em cima de uma bancada de laboratório, medindo apenas 70 mm de diâmetro por 50 mm de altura (sem módulos de expansão), em diversas áreas de desenvolvimento.[22] O e-puck está equipado 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 que se pode 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 bit, permitindo que seja possível a introdução de sensores para aprendizagem, o Miabot é muito inferior ao nível de *hardware* em relação aos outros. 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, que é uma vantagem em relação aos outros robôs.

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] O que 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 com restantes já descritos, no entanto não deixam de ser uma referência para esta área de pesquisa já que possuem dimensões suficientes para se poderem operar em espaços como o de um laboratório de pesquisa robótica.



(a) (b)

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

O AmigoBot™ (figura 1.8a) está equipado com um microprocessador de 32-bit 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 bit, só tem capacidade para receber dados de mais oito dispositivos pois 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 conecta-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-bit 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 encaixar um computador, que o torna o Pioneer 3-DX completamente autónomo. A maior vantagem deste robô é a sua capacidade de carga, até 25 kg, que lhe permite ser equipado com sensores mais sofisticados, como por exemplo um sensor de laser industrial ou câmaras de video digitais de maior resolução.

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 em forma de estrela (figura 1.9a) do CCSL (*Cornell Computational Synthesis Laboratory*) da Universidade de Cornell. Estes robôs embora não sejam muito flexíveis, quanto às áreas de desenvolvimento, possuem as características mais indicadas para a área que 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 tiveram 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 duas microfones para comunicação acústica, uma porta de comunicações por infra-vermelhos, para a comunicação entre dois robôs próximos, e 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 só ainda tinha 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 *starfish robot* 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-bit, 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 que são necessárias para que 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, quer não volátil (com por exemplo EEPROM ou *flash*) para guardar os dados da aprendizagem.

1.3.3 *Reinforcement learning*

O *Reinforcement Learning* estuda de que forma 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 que vai aprendendo. 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 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 de formas 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 mas 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.[29]

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.[29]

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]

Em anexo 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 para-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 para-choques não é imperial para um robô poder aprender, no entanto, um sensor de contacto físico, como é um sensor passivo, é sempre mais fiável do que qualquer tipo de sensor activo. Isto porque, um sensor de contacto não é volátil a factores externos que possam interferir com a sua mecânica, ao contrário da maioria dos sensores, 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 só fornecem 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 robô saber a distância a que se encontram obstáculos.

Numa primeira abordagem, a ideia era pegar num robô já existente e melhorar o seu equipamento 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*. Então, optou-se por se basear num projecto *open source* de um robô concebido para competições de mini-sumo, o Mark III[2], e conceber um robô novo de raiz de baixo custo usando, como inspiração, o Mark III.

O Mark III 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 robô apresenta um chassis 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.

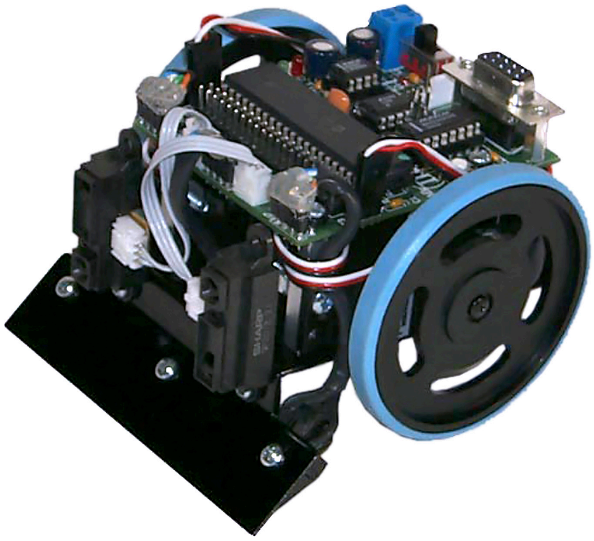


Figura 2.1: Robôs MarkIII

Assim, concebeu-se um robô com um chassis robusto, um sistema de para-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*), que significa pequeno robô autónomo para aprendizagem. 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, duas rodas, accionadas pelos servos, no centro do chassis 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

Como já foi referido, foram concebidos um chassis e um sistema de para-choques suficientemente robustos para que a aprendizagem possa ser realizada eficientemente. A base do chassis foi fabricado 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 chassis. Para colmatar a fraca resistência mecânica do alumínio foi concebido uma peça em acrílico que, para além de aumentar a resistência mecânica do chassis, também aloja os sensores de contacto que fazem parte do sistema de para-choques. O chassis 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 o sensores de ultra-sons e infra-vermelhos de distância frontais e, dois pilares que suportam, cada um, um sensor de distância infra-vermelho e dois apoios esféricos livres.

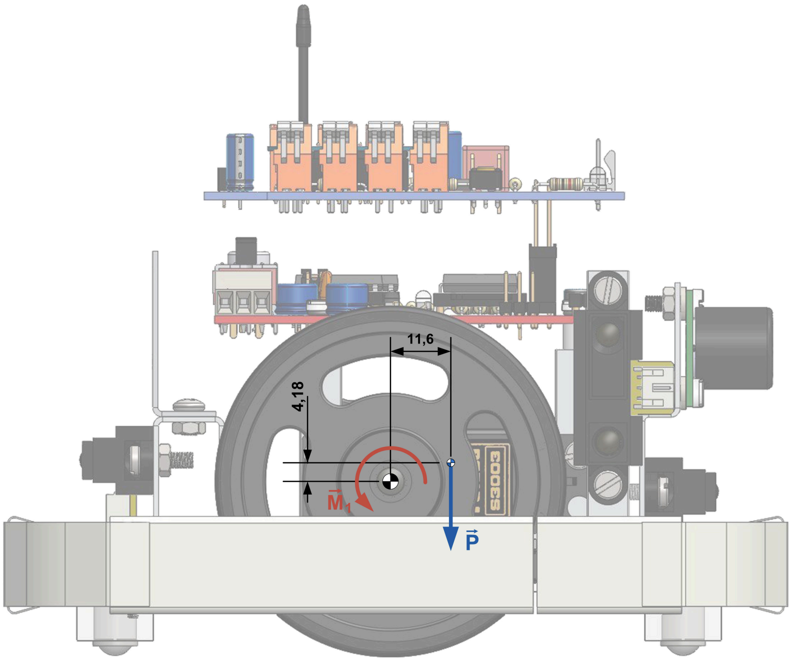


Figura 2.2: Diagrama de corpo livre para o cálculo do somatório dos momentos??!!!

A base do chassis tem uma área plana grande no centro, onde são alojados os servos, nas extremidades laterais estão dois "braços" onde são colocados as peças de acrílico, onde são alojados os sensores de contacto do sistema de para-choques, na parte de trás a chapa é dobrada em forma de "U" para dar mais alguma resistência mecânica ao chassis e para a colocação de um sensor de distância de infra-vermelhos traseiro, e nas extremidades frontal e traseira a chapa é dobrada para alojar dois apoios esféricos livres.

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 torque máximo dos servos. Devido à complexa distribuição de massas dos diversos componentes do SALbot, o centro de massa foi calculado 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 chassis. Das características do servo da Futaba® S3003 retiraram-se os seguintes valores:

$$\textbf{Velocidade:} \left\{ \begin{array}{lll} 0.23\ \mathrm{s}/60^\circ & @ & 4.8\ \mathrm{V}; \\ 0.19\ \mathrm{s}/60^\circ & @ & 6\ \mathrm{V}. \end{array} \right.$$

$$\textbf{Torque:} \left\{ \begin{array}{lll} 3.2\ \mathrm{kgf} \cdot \mathrm{cm} & @ & 4.8\ \mathrm{V}; \\ 4.1\ \mathrm{kgf} \cdot \mathrm{cm} & @ & 6\ \mathrm{V}. \end{array} \right.$$

Para a verificação de o momento criado pelo peso do robô ser suficiente para anular o binário dos servos primeiro calculou-se o binário dos dois servos e, uma vez que estes operam a 5V, usou-se interpolação linear para calculo:

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

e como,

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

então,

$$\begin{aligned}
\overrightarrow{M}_1 &= 6.7\times 9.8\times 10^{-2}\\
&= 65.660\times 10^{-2} \quad \mathrm{N}\cdot\mathrm{m}
\end{aligned}$$

depois determinou-se o peso do robô a partir do modelo 3D com uma majoração de 50%, e calculou-se o momento criado no eixo de rotação dos servos:

$$\begin{aligned}
\overrightarrow{P} &= 0.462\times 9.8\\
&= 4.5276 \ \mathrm{N}\ ;\\
\overrightarrow{M}_P &= \overrightarrow{P}\times \mathrm{d}\\
&= 4.5276\times 11.6\times 10^{-3}\\
&= 5.252\times 10^{-2}\mathrm{N}\cdot\mathrm{m}
\end{aligned}$$

como

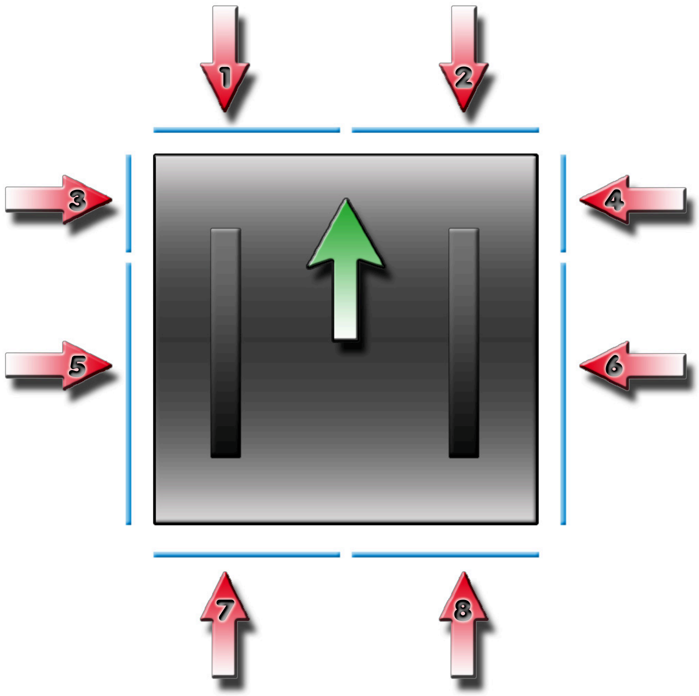
$$\overrightarrow{M}_1 > \overrightarrow{M}_P$$

então, é necessário um apoio na parte traseira do robô para que este não levante no arranque para trás.

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 8x8 mm de alumínio corrente para garantir uma fixação rígida entre os servos e o chassis e também para suportar as placas de circuito impresso. Estes são fixados à base do chassis por uma ligação aparafusada, cada pilar possui um furo roscado no qual um parafuso M3 garante a fixação do pilar à base do chassis. 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 colocar o sensor numa determinada orientação desejada. Permitindo, assim, várias configurações diferentes do SALbot que podem ser testadas, em vários tipos de aprendizagem ou vários tipos de ambiente com o fim de descobrir qual 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, garantido por dois furos roscados onde apertam dois parafusos M3, permite também a fixação dos servos aos pilares frontais do chassis 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 resistente o suficiente para a sua função.

O sistema de para-choques foi concebido tendo em conta a sua robustez e flexibilidade, mas também que permitisse 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,



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 contacto.

um para cada zona de contacto excepto nas zonas laterais esquerda e direita, que contém dois em paralelo para uma maior estabilidade. Esses sensores de contacto são fins-de-curso com uma haste que contém uma roda na ponta para poderem deslizar na face de contacto. Além dos sensores de contacto o sistema de para-choques é constituído por barras de contacto, que fazem a transferência do contacto exterior com os sensores de contacto e representam cada uma das zonas de contacto, chapas arredondadas para os cantos do robô, que permite a transferência do contacto exterior nos cantos para os sensores de contacto para uma ou duas zonas de contacto 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 barra possuem um perfil em forma de um "T" duplo para que a chapa de alumínio possa deslizar em duas direcções, longitudinalmente e transversalmente, com uma 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. E estão fixadas, por um adesivo, a uma tela fina de latex na face de contacto, que por sua vez é fixada ao chassis 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 chassis.

Uma vez que as telas de latex são flexíveis, estas, 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.

Para que um robô saiba quando bateu necessita de um dispositivo de detecção que detecte um embate em qualquer direcção, por isso concebeu-se o sistema de para-choques para que quando existisse um contacto nos cantos, este, fosse 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 e o comportamento do sistema de para-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 o canto 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 para-choques do SALbot permite a detecção de um embate qualquer que seja a direcção do mesmo.

Os cantos forma 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, impedindo o canto de rodar.

Os apoios esféricos livres são compostos por uma esfera de aço de ϕ 7 mm uma mola e uma estrutura cilíndrica com ϕ 8 mm. A mola pressiona a esfera contra a estrutura, para que esta apenas só 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 um adesivo, à base do chassis.

Em cada servo está fixada uma roda, de ϕ 65 mm, por um parafuso. Cada roda tem um elástico de borracha à volta, o que permite a aderência da roda 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 foi necessário 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 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. Como já foi referido cada servo é fixado ao pilares do chassis 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

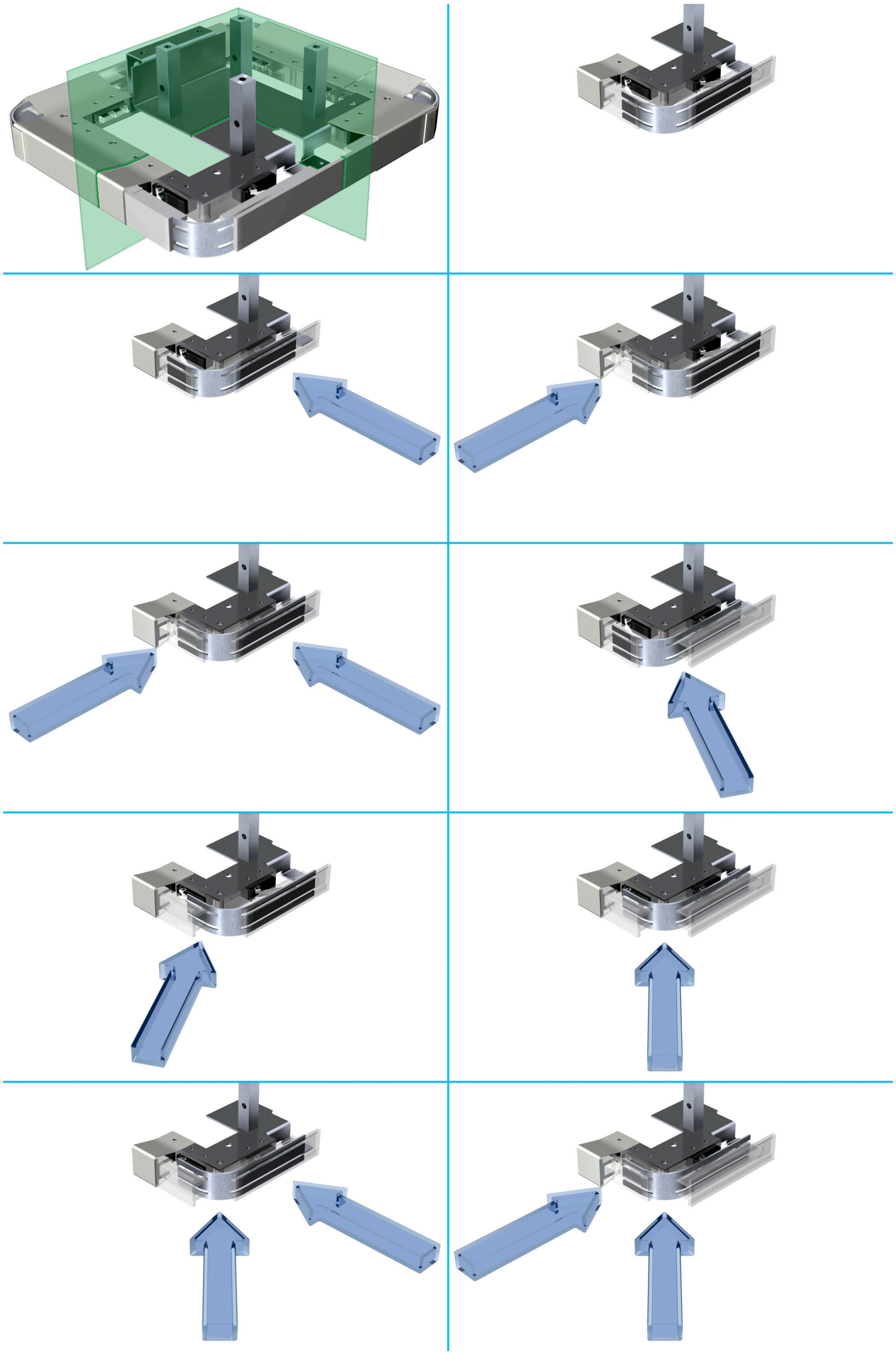


Figura 2.4: Comportamento do sistema de para-choques nos cantos.

o suporte dos sensores frontais, na parte traseira o parafuso é apertado a uma porca M3.

Existe ainda um suporte, adicional, traseiro para o suporte de um sensor ultra-sons de distância adicional, no entanto achou-se que não seria necessária a implementação desse sensor, pelo que, posteriormente esse suporte passou a suportar a pilha que alimenta o circuito electrónico das placas.

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

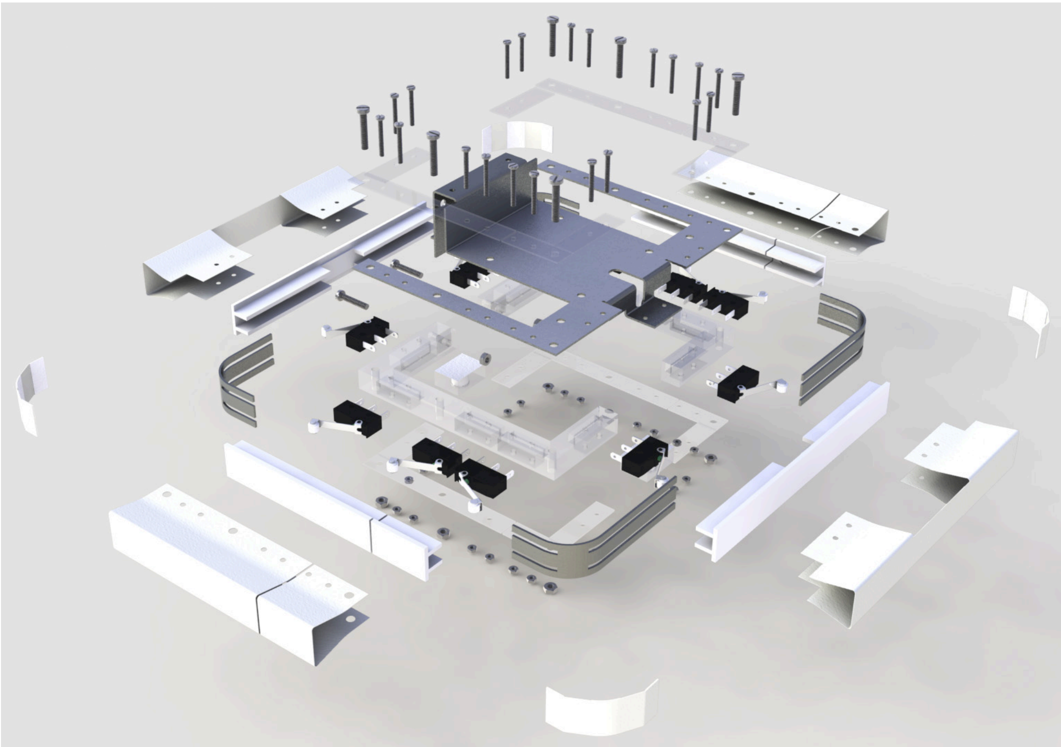


Figura 2.5: Perspectiva explodida do chassis do SALbot.

A sequência de montagem para o chassis é a seguinte:

1. Fixar as peças que suportam os sensores de contacto à base do chassis, através de dois parafusos e duas porcas M3 na parte traseira;
2. Fixar os apoios esféricos nos suportes respectivos e fixá-los na base do chassis;
3. Colocar cada sensor de contacto em cada caixa, das duas peças que suportam os sensores, para o efeito;
4. Colar cada tela de latex à barra de contacto correspondente;
5. Orientar os furos de cada tela de latex, da parte superior, com os furos dos “braços” da base do chassis, colocar as respectivas tiras de PVC orientando-as pelos furos e aplicar os parafusos M3 e M2;

6. Orientar os furos de uma tela de latex, 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, colocar as respectivas tiras de PVC orientando-as pelos furos e aplicar as porcas M3 e M2;

7. Encaixar os cantos nas barras de contacto já fixadas;

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

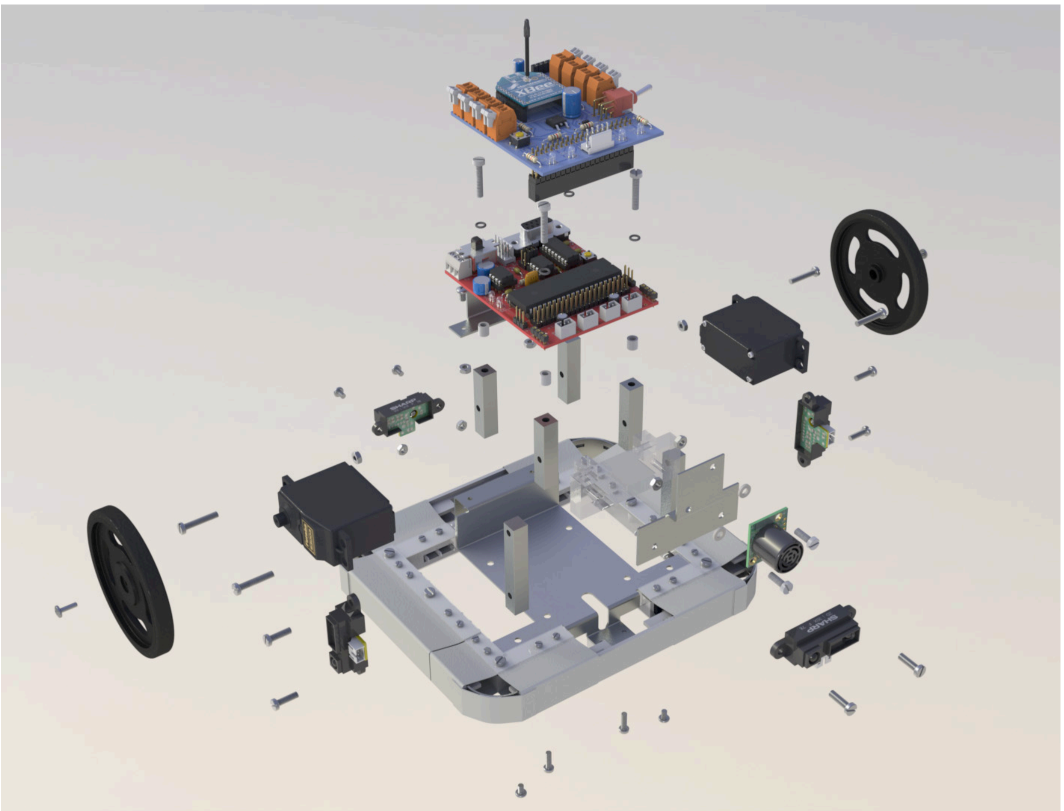


Figura 2.6: Perspectiva explodida do SALbot.

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

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

2. Fixar cada servo aos pilares frontais com um parafuso apertando-o à peça de acrílico frontal, onde posteriormente são montados os sensores frontais;

3. Fixar as rodas nos servos com um parafuso;

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

5. Fixar os pilares, já fixados aos servos e à peça de acrílico frontal, à base do chassis com parafusos M3;
6. Fixar os sensores de infra-vermelhos nos pilares respectivos para o efeito, com parafusos M3, e o sensor de ultra-sons frontal na chapa de suporte dos sensores frontais, com dois parafusos M3 de nylon, duas porcas M3 e com um espaçador de nylon entre o sensor e a chapa de suporte;
7. Fixar o sensor de infra-vermelhos traseiro na parte de trás do chassis com dois parafusos e porcas M3, fixar os pilares de suporte dos sensores de infra-vermelhos à base do chassis, com a orientação desejada, e fixar o sensor de infra-vermelhos frontal à chapa de suporte dos sensores frontais, fixando também a chapa de suporte dos sensores frontais à peça de acrílico frontal;
8. Fixar a chapa de suporte traseira com dois parafusos e porcas M2.5 ao chassis;
9. Fixar a bateria de alimentação dos servos por baixo da base do chassis e a bateria de alimentação das placas de circuito impresso na chapa de suporte traseira, utilizando fitas adesivas de velcro;
10. Fixar a placa de circuito impressa principal aos pilares com parafusos M3 em nylon, colocando um espaçador de plástico entre cada pilar e a placa, e um o-ring de borracha entre a cabeça de cada parafuso e a placa;
11. Fazer as ligações dos servos, baterias, sensores de infra-vermelhos e ultra-sons com a placa de circuito impresso principal;
12. Encaixar a placa de circuito auxiliar 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 concebidas duas placas de circuito impresso, a placa de circuito principal e a placa de circuito auxiliar para o SALbot e uma placa de interface de comunicação *wireless* 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 apenas responsável, apenas, 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. As placas foram concebidas no Altium Designer, um programa ECAD (*Electronic Computer-Aided Design*) da Altium.

A placa de circuito principal foi desenvolvida com base na placa do Mark III. Nesta encontra-se 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, um unidade de memória EEPROM 24LC1025 com 1 Mbit, uma porta de expansão da interface I2C, que é configurada por 6 pinos, um botão de reset e on-off, e 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. Este 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.

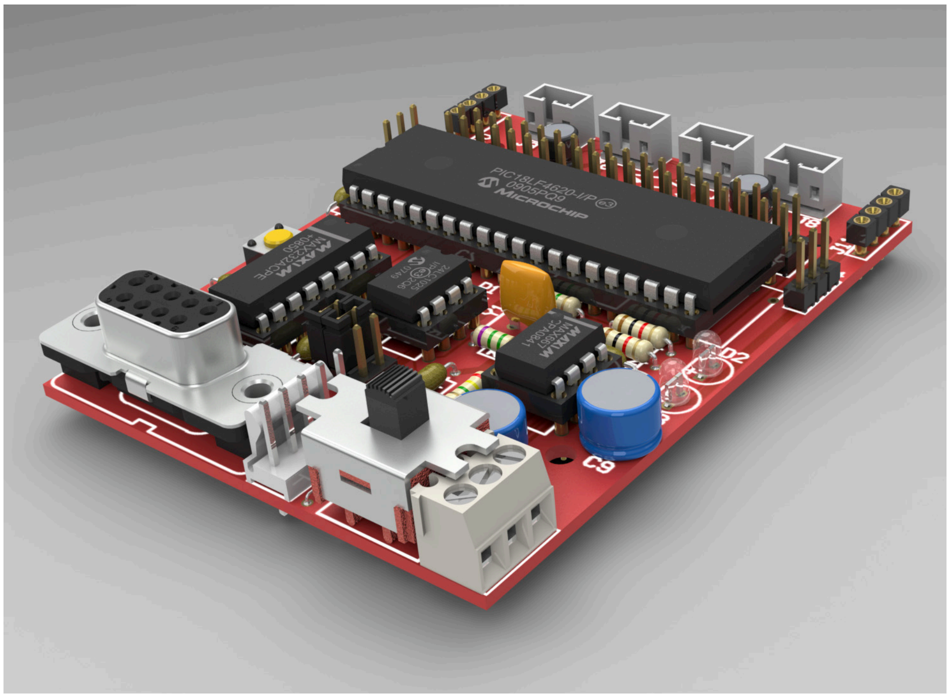


Figura 2.7: Placa de circuito principal do SALbot.

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* e 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.

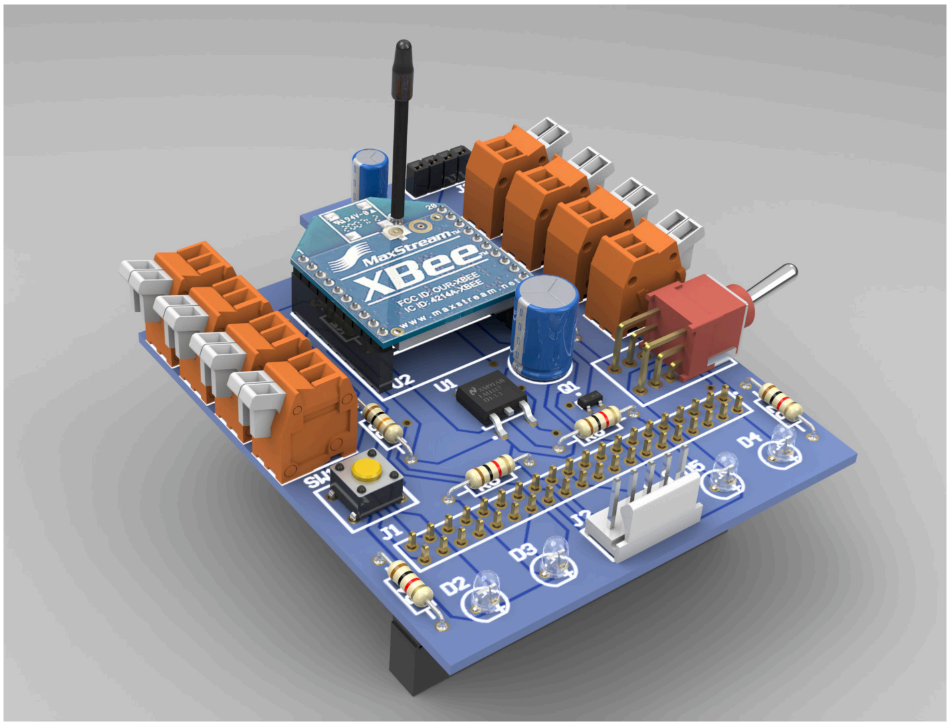


Figura 2.8: Placa de circuito auxiliar do SALbot.

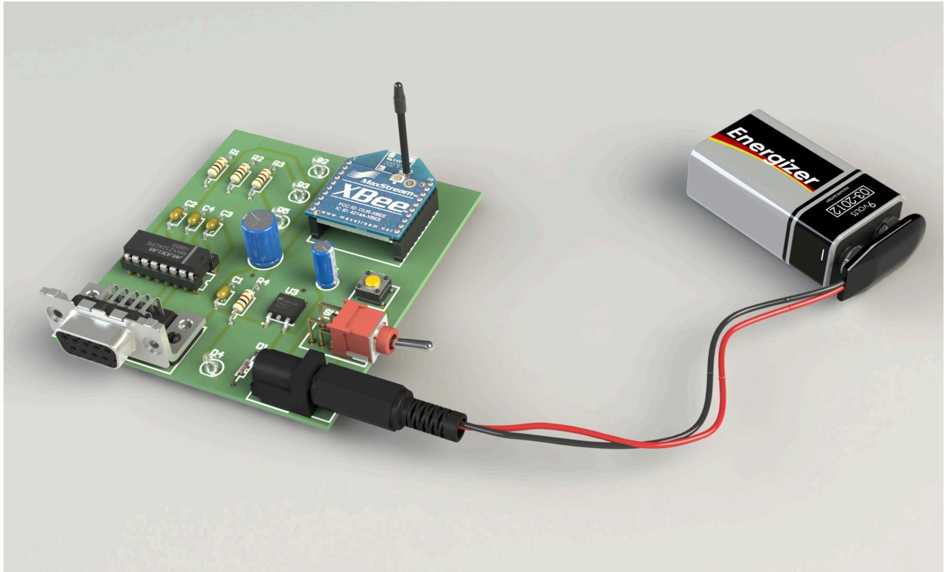


Figura 2.9: Placa de interface de comunicação.

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 descrito a arquitectura do sistema dos robôs, isto é de que forma é que os dispositivos estão interligados e como é que são feitas as comunicações. Seguidamente é abordado o firmware e o funcionamento dos diversos dispositivos, isto é, a forma como os dispositivos estão programados para interagir com outros dispositivos e como é feita essa interacção. Depois, são descrito 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 a EEPROM externa e o PWM, utilizado para controlar os servos. Por fim é exposto software de aprendizagem implementado no microcontrolador, isto é o programa desenvolvido para a aprendizagem dos robôs.

3.1 Arquitectura do SALbot

Antes de mais, é importante explicar o funcionamento da arquitectura do sistema do SALbot (ver figura 3.1). Como já foi referido do capítulo anterior, o SALbot possui duas placas de circuito impresso, a placa de circuito principal e a auxiliar, sendo a auxiliar uma expansão à principal. Na placa de circuito principal está implementado um microcontrolador programado para aprendizagem. Além dos cálculos específicos do algoritmo de aprendizagem, o microcontrolador é encarregue de 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 conectados 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 de 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 802.15.4, 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.

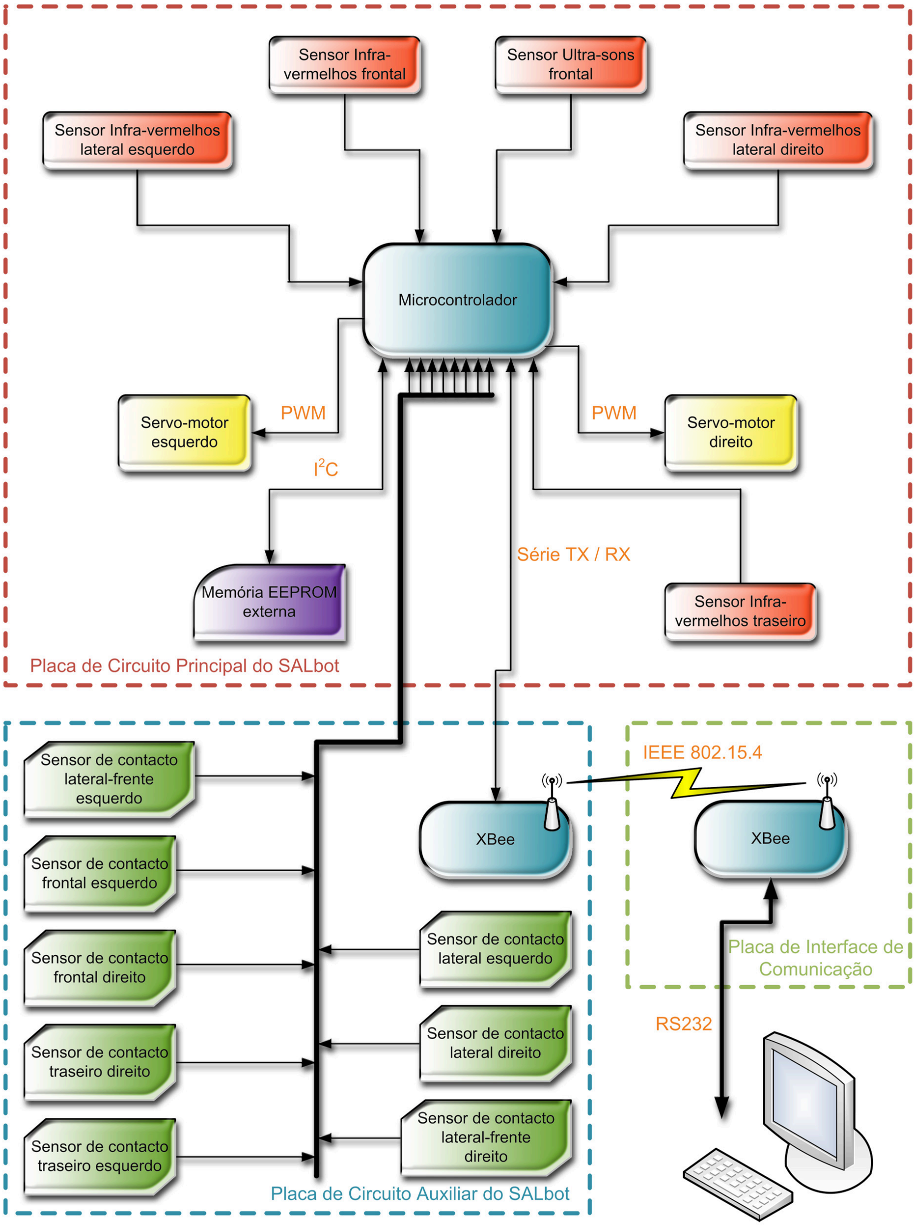


Figura 3.1: Arquitectura do sistema do SALbot.

3.2 Firmware e funcionamento dos dispositivos

3.2.1 Sensores de contacto

Os sensores de contacto implementados foram os DB3C-B1RC da Cherry. Estes sensores de contacto não são mais do que sensores de fim de curso, accionado por uma haste com uma roda na extremidade. Estes funcionam funcionam como um botão, um circuito, normalmente aberto, entre os 5V de alimentação e o pino do microcontrolador do respectivo sensor é fechado quando o sensor é activado. Assim os 5V são enviados para o pino do microcontrolador indicando-o da existência de um contacto. Para evitar o flutuamento da tensão nos pinos do microcontrolador, o que induziria o microcontrolador em erro, existem "*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 os 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 1.95V e 2.55V, e representa a distância medida pelo sensor.

Este tipo de sensores, basicamente, 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 pelos 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][27][14]

Apesar de este tipo de sensores ser quase imune à interferência da luz ambiente e não depender da cor do objecto detectado, é extremamente difícil detectar objectos que permitam a passagem da luz, o sinal analógico enviado pelo sensor não é linearmente proporcional à distância e o valor do ângulo medido pelo CCD varia com a temperatura ambiente. Assim para uma melhor precisão na medição das distâncias retiraram-se vários valores do sinal enviado pelo sensor para diversas distâncias medidas, a partir dos quais se fez um gráfico do valor do sinal analógico em função da distância medida, que é possível verificar na figura 3.4. Com esses valores, usando uma regressão potencial, que era a que mais se adequava aos valores obtidos, obteve-se uma função aproximada da distância medida em função do sinal analógico enviado pelo sensor.

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

Embora a distância precisa em milímetros da distância entre o robô e um objecto não seja fundamental para o algoritmo de aprendizagem, foi necessário fazer a correspondência de algumas medidas de distância para valores analógicos, lidos pelo microcontrolador, para criar intervalos de distâncias, a que correspondem certos estados do robô.

GP2D120X

Sinal analógico (V)	valor (contagens)	distância(mm)
0,325	66,56	400
0,375	76,8	350
0,425	87,04	300
0,525	107,52	250
0,65	133,12	200
0,735	150,528	180
0,8125	166,4	160
0,925	189,44	140
1,05	215,04	120
1,25	256	100
1,4	286,72	90
1,575	322,56	80
1,775	363,52	70
2,025	414,72	60
2,35	481,28	50
2,75	563,2	40
3,5	716,8	30

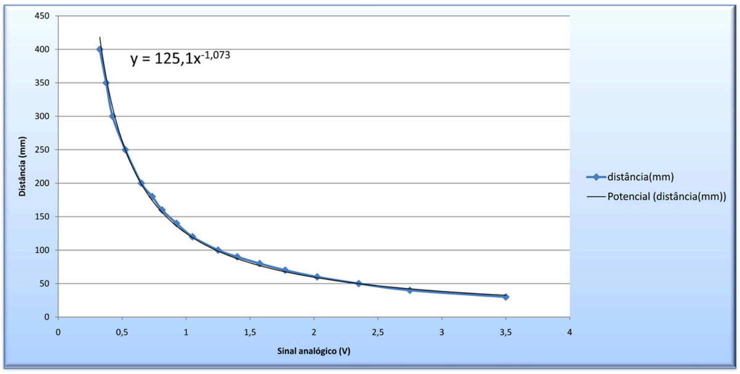


Figura 3.4: Gráfico dos valores do sinal analógico em função da distância do sensor GP2D120.

3.2.3 Sensores de distância ultra-sons

Os sensores de ultra-sons implementados nos robôs foram os LV-MaxSonar®-EZ0™, estes sensores permitem a medição de distâncias entre 150 mm e 6450 mm. Cada sensor têm vários tipos de interface para a leitura da distância medida, nomeadamente porta série, PWM e sinal analógico. Por ser mais 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 ligações, uma 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.

Isto é, o sinal analógico apresenta uma resolução de $3.845 \times 10^{-4}\ V \cdot mm^{-1}$.

Figura 3.5: 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 som ultra-sónico. No sensor existe um emissor de sons 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 detectada pelos ouvidos humanos, essas onda mecânica viaja pelo ar à velocidade do som (cerca de 343 $m \cdot s^{-1}$ em ar seco a 20°), e quando encontra um obstáculo no seu caminho esta vai ser reflectida de volta para o sensor onde é captada pelo receptor, a onda reflectida designa-se por eco. Assim o sensor emite um feixe de ultra-sons e fica à espera do seu eco, quando receber o eco vai calcular o tempo que demorou desde que enviou o feixe até à recepção do eco e sabendo a velocidade do som, a distância do sensor ao objecto que reflectiu o som é facilmente determinada. O sensor envia a cada 49 ms um feixe de ultra-sons e coloca o PWM no nível alto, que se mantém durante 37.5 ms se não for recebido um eco, caso seja recebido um eco o PWM é colocado no nível baixo. No restante tempo, menos de 4.7 ms é usado para corrigir o nível do sinal analógico e os últimos 4.7 ms são para enviar dados pela porta série.[27]

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 alto. Se tal acontecer o sensor irá correr um ciclo de calibração de 49 ms e seguidamente uma leitura da distância, demorando assim aproximadamente 100 ms.

Embora este sensor possa detecta objectos entre os 0 mm e 150 mm, enviado o valor correspondente a 150 mm nesses casos, só mede distâncias absolutas a partir dos 150 mm ele. Outra desvantagem é o facto de o sensor não realizar uma reajustamento da velocidade do som a cada medição relativamente à temperatura e humidade, que a influenciam. Em vez disso, a calibração do sensor consiste na compensação em relação ao padrão do efeito *ringdown* do sensor. 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*.

3.2.4 Servos

Para a movimentação do SALbot implementaram-se dois servos, com a arquitectura 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 3.6: Servo Futaba® S3003.

Os servos implementados, como já foi referido anteriormente, foram os S3003 da Futaba®. Estes servos estão preparados para uma amplitude de movimentos restrita a 90° para cada lado, pelo que, para permitir a 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". 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, este 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 o tamanho deste tipo de memória disponível nos microcontroladores é insuficiente, sendo no máximo 1024 *bits* para a família da série PIC18 e 4096 para a família dsPIC, para a quantidade de dados relativos à aprendizagem que são necessários armazenar, mesmo para os métodos de aprendizagem mais simples.

O dispositivo de armazenamento de memória EEPROM implementado foi o 24LC1025 da Microchip com 1 Mbit de memória, como já foi referido. O microcontrolador acede a este dispositivo, escrevendo e lendo dados em posições específicas da memória disponível, através da comunicação por I^{2}C™, que será descrita na secção de "Protocolos de comunicação e comando". A memória deste dispositivo agrupada em páginas de 128 *bytes*, cada *byte* é composto por oito *bits* ao qual

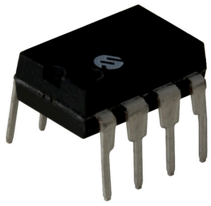


Figura 3.7: Unidade de EEPROM de interface série 24LC1025 da Microchip.

está associados 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 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.

Apesar de este dispositivo ser o que mais capacidade de memória EEPROM apresenta de todos os outros dispositivos disponíveis no mercado, o dispositivo EEPROM implementado 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 implementar uma unidade EEPROM externa, existe uma porta de expansão $I^2C^{™}$, é possível implementar até um máximo de quatro dispositivos EEPROM no mesmo bus $I^2C^{™}$.

3.2.6 XBee

Figura 3.8: Módulo XBee XB24-AWI-001 da Digi®.

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 que comunicam através do protocolo *wireless standard* IEEE 802.15.4.

Estes módulos possuem uma interface série, a partir da qual se enviam e recebem 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, que os envia para o max232 que 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.

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, podendo aceitar taxas de *baud rate* não estandardizadas. A comunicação sem fios que através do protocolo IEEE 802.15.4 opera na gama de frequências de 2.4 GHz, num dos dezasseis canais previstos para este protocolo. A transmissão de dados sem fios é feita com uma potência de 1 mW, ou 0 dBm, e a sensibilidade recepção de dados é de -92 dBm.[10]

Como os módulos XBee operam a 3.3 V, foi colocar nas placas 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. Uma vez que a comunicação sem fios é feita no 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 o número de um dos canais, disponíveis pelo protocolo IEEE 802.15.4, 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-bit mais significativos e DL os 32-bit 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, 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 0-7 correspondem às taxas de transferência de dados estandardizadas de 1200 bps a 115.2 kbps e de 0x80 a 0x3D090 para taxas de transferência de dados não estandardizadas até 250 kbps.[10]

Embora os módulos XBee só 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 oito entradas e saídas digitais, das quais seis podem ser configuradas como entradas analógicas, um pino exclusivo para entrada digital e outro para saída digital e duas de PWM. O que possibilita o controlar e aquisição de dados de dispositivos remotamente sem a necessidade de um microcontrolador ou qualquer outro dispositivo 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, mesmo os XBee-PRO ou XBee com o protocolo ZigBee. 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. E 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.[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 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 servos

Os servos, como já foi referido, são comandados através de um sinal PW enviado pelo microcontrolador. O servo implementado foi concebido para se mover 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 3.9, 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]

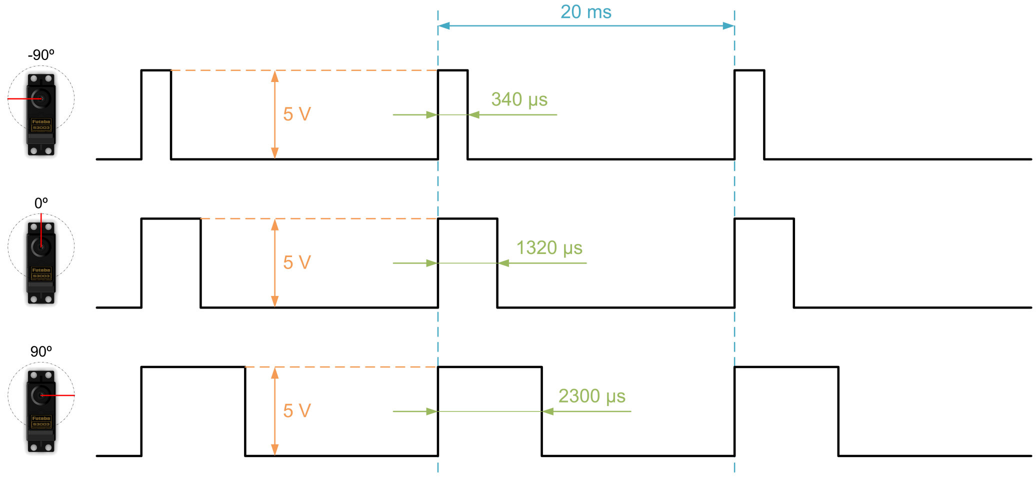


Figura 3.9: Sinal PW do servo S3003 da Futaba®.

No entanto, como foi necessário alterar os servos para possam rodar continuamente, o controlador do servo têm 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°, o servo vai tentar posicionar-se nessa posição, mas como o controlo interno do servo indica que o servo está na posição central, o servo continua a rodar. Assim, para rodarmos o servo no sentido horário basta enviar um sinal PW com um pulso entre 1320 μs e 2300 μs, em que a velocidade aumenta quanto maior for o comprimento do pulso. Para rodarmos o servo no sentido anti-horário deve-se enviar um sinal PW com um pulso entre 1320 μs e 320 μs, em que a velocidade aumenta quanto menor for o comprimento 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.

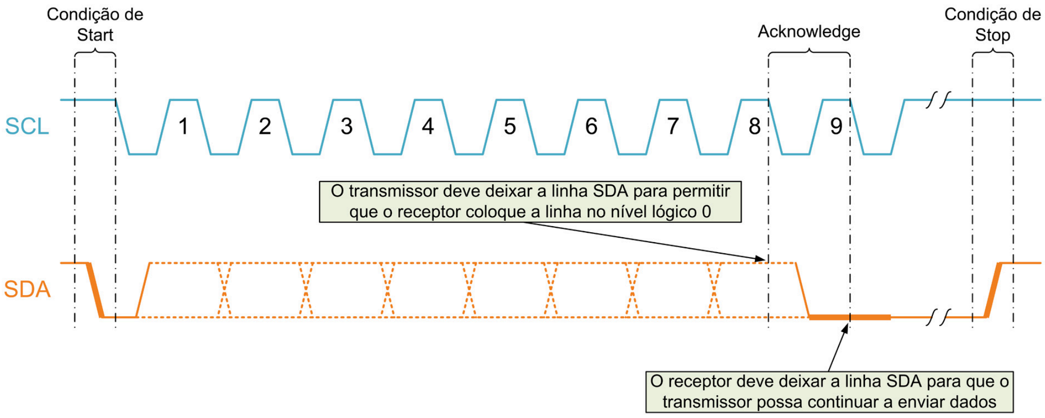


Figura 3.10: Composição da sequência de transferência de dados no I²C™.

A interface de comunicação I²C™ requer que a comunicação seja controlada por um dispositivo *Master*, que neste caso é o microcontrolador, 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*. Os dispositivos, tanto o *Master* como o *Slave* podem ser transmissores (enviam dados) ou receptores (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]

Endereçamento dos dispositivos

Em todos os comandos o primeiro *byte* a enviar, após a condição de *Start*, é o *byte* de controlo. Como é possível verificar na figura 3.11 o *byte* de controlo é composto por quatro *bits* com o código de controlo, um *bit* relativo à selecção do bloco, 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 kbit, 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. E o bit que indica o tipo de operação é 0 para operações de escrita e 1 para operações de leitura.

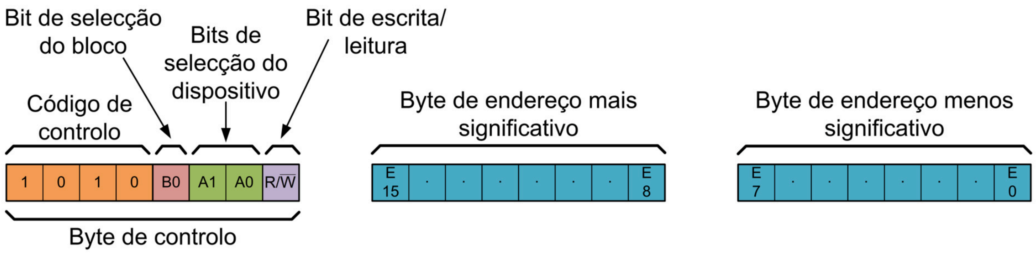


Figura 3.11: Composição da sequência de *bits* do endereçamento de uma mensagem por I^{2}C™.

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

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 da 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 enviado correspondem ao primeiro *byte* 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 originar com que o *byte* de endereço menos significativo ultrapasse o valor 128, o *byte* do endereço volta ao inicio da página e vai escrever os dados restantes por cima dos valores que se encontravam nessas posições de memória anteriormente.

Operações de leitura

Existem três tipos de operações de leitura possíveis de realizar, 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* ao 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.

Na 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 primeiro o *byte* mais significativo e depois da 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 enviada. 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 que for desejada 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, existindo a possibilidade de os SALbot receberem hipotéticas mensagens vindas de outros dispositivos *wireless* que poderiam interferir com o funcionamento normal dos robôs. Assim houve a necessidade de implementar um protocolo que só fosse descodificado pelos SALbot ou pela 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* e tendo em conta a possibilidade da perda de informações pelo meio da mensagem, já que é as comunicações *wireless* são permeáveis a interferências externas. Para permitir o dinamismo do comprimento da mensagem foi criado um campo na mensagem que contém a informação do tamanho da mensagem. E para evitar a leitura de mensagens com erros foi criado um campo na mensagem que indica se a mensagem é verdadeira ou falsa.

A mensagem, como é visível na figura 3.12, é constituída, basicamente, por quatro secções: o *byte* de início, um *byte* com o comprimento e tipo, a mensagem propriamente dita e um *byte* de verificação.

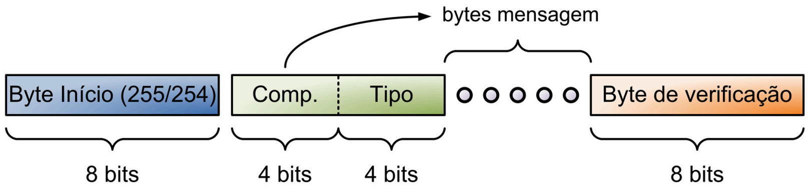


Figura 3.12: Topologia da mensagem de comunicação entre o SALbot e o computador.

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. No segundo *byte* temos 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.

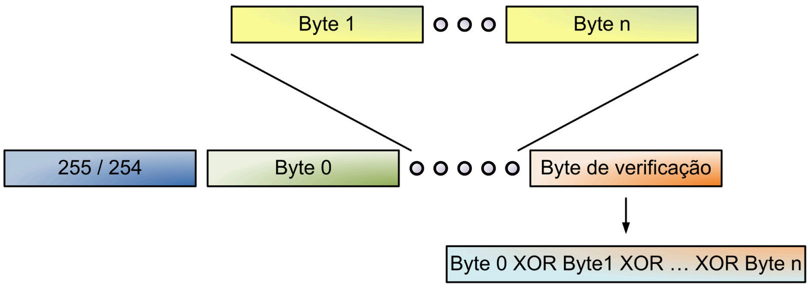


Figura 3.13: Descrição gráfica do *byte* de verificação.

Na figura 3.13 é possível verificar 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), isto é, faz-se a operação XOR do primeiro *byte* com o segundo, depois repete-se a operação para o 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.

Mensagem de modo de operação

A mensagem de modo de operação (figura 3.14) é enviada pelo programa que gere a 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.

No modo Manual o SALbot fica à espera que o utilizador envie mensagens para enviar dados ou para ser comandado.

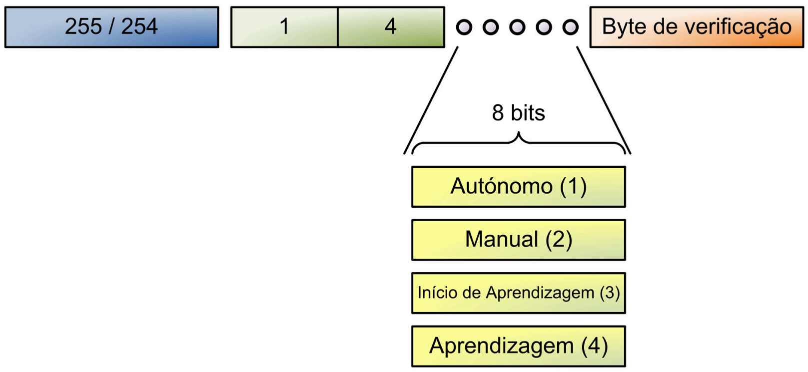


Figura 3.14: Mensagem de alteração do modo de operação do SALbot.

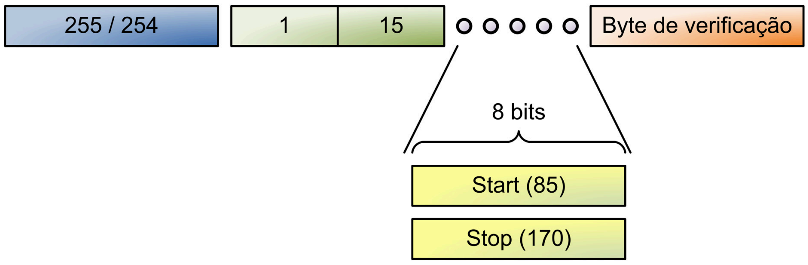


Figura 3.15: Mensagem de Início/Paragem do SALbot.

Mensagem de Início/Paragem

Uma vez que, quando o SALbot é (re)iniciado o programa encontra-se no modo Idle (não realiza qualquer operação a não ser a verificação de mensagens recebidas), a mensagem de início/paragem (figura 3.15) é usada para o utilizador iniciar o programa do SALbot para este ler e enviar os dados dos vários dispositivos. No entanto sempre que o utilizador desejar, pode enviar uma mensagem para colocar o programa do SALbot no modo Idle.

Ora é esse o motivo desta mensagem, iniciar ou terminar a leitura e envio dos dados dos dispositivos, seja qual for o modo de operação. É constituída por um *byte*, que indica se é para iniciar ou para parar.

Mensagem de pedido de dados

É através da mensagem de pedido de dados (figura 3.16) que o utilizador, quando o SALbot se encontra no modo Manual, pede ao SALbot que este lhe envie dados. Estes dados podem ser dos valores das distâncias dos sensores de distância, o estado dos sensores de contacto, ou os valores da tabela de aprendizagem. Também, apenas, constituída por um byte, a mensagem representa o tipo de dispositivo do qual queremos receber dados, como é possível verificar na figura anterior.

Mensagem de envio do estado dos sensores de distância

Sempre que o SALbot recebe um pedido de envio de dados dos sensores de distância, este envia uma mensagem com esses dados. Como é possível verificar na figura 3.17, são enviados os

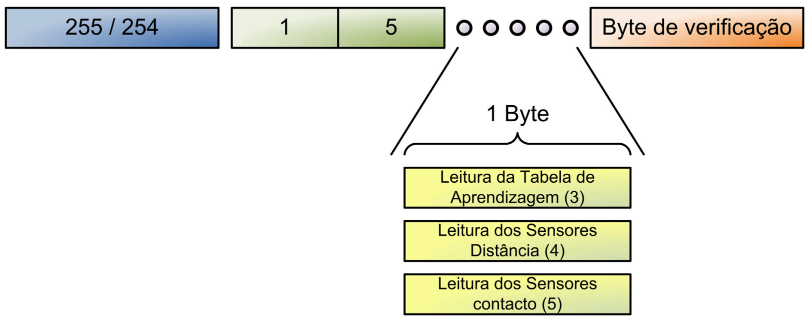


Figura 3.16: Mensagem de pedido de dados.

valores da distância dos sensores de distância em dois *bytes* para cada sensor, o primeiro *byte* com o valor mais significativo e o segundo com o valor menos significativo da distância lida do sensor. Como os valores lidos pelas portas analógicas do microcontrolador são de dez *bits*, nos *bytes* mais significativos dos valores dos sensores os últimos seis *bits* são desprezados.

Os pares de *bytes* enviados relativos aos sensores de distância são enviados pela seguinte ordem, infra-vermelhos frontal, infra-vermelhos traseiro, infra-vermelhos esquerdo, infra-vermelhos direito e, por fim, ultra-sons frontal.

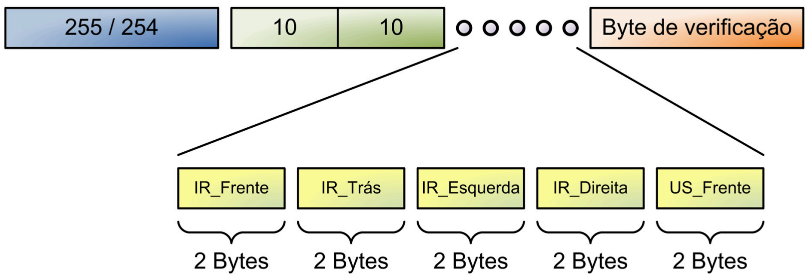


Figura 3.17: Mensagem de envio do estado dos sensores de distância.

Mensagem de envio do estado dos sensores de contacto

Uma vez que, os sensores de contacto estão activados ou desactivados e estão dispostos por oito zonas de contacto distintas, basta enviar um *byte* em que cada *bit* corresponde ao estado de um sensor. Como é possível ver na figura 3.18, o primeiro *bit* do *byte*, que é enviado por último, corresponde à zona trás-direita, o segundo à zona trás-esquerda, o terceiro à zona frente-direita, o quarto à zona frente-esquerda, o quinto à zona lateral-direita-frente, o sexto à zona lateral-esquerda-frente, o sétimo à zona lateral-direita e o oitavo à zona lateral-esquerda. Esta foi a ordem definida, visto que corresponde ao *byte* da porta do microcontrolador onde estão implementados os sensores de contacto.

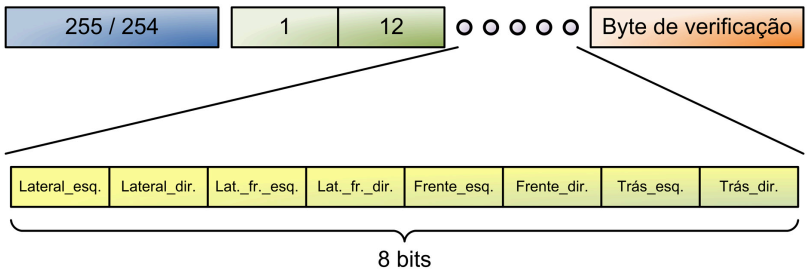


Figura 3.18: Mensagem de envio do estado dos sensores de contacto.

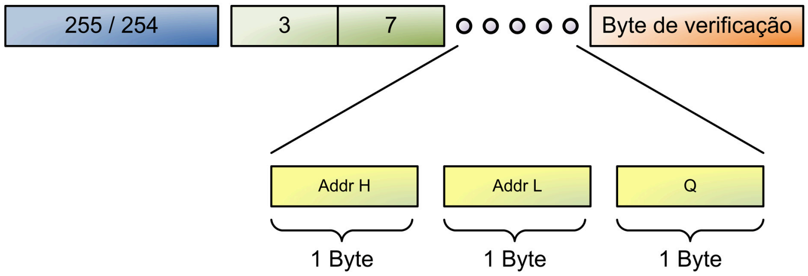


Figura 3.19: Mensagem de envio de um elemento da tabela de aprendizagem.

Mensagem de envio de um elemento da tabela de aprendizagem

3.4 Software

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 é um algoritmo *model-free* de aprendizagem, muito idêntico ao *Q-learning*, sendo que a única diferença entre eles residir no facto de o SARSA ser um método de aprendizagem *on-policy*, enquanto o *Q-learning* é um método de aprendizagem *off-line*. Isto é, 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.

Foram definidos cinco estados diferentes para o SALbot: o *Idle*, Autónomo, Manual, Inicialização da Aprendizagem e Aprendizagem. Estes estados correspondem a diferentes modos de execução do SALbot. O estado *Idle* faz com que o robô fique em repouso e 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 serve apenas para repor a tabela de aprendizagem a zeros, os parâmetros de aprendizagem nos valores originais e 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.

Na figura 3.20 é possível ver o diagrama de fluxo da rotina *Main* dos SALbot.

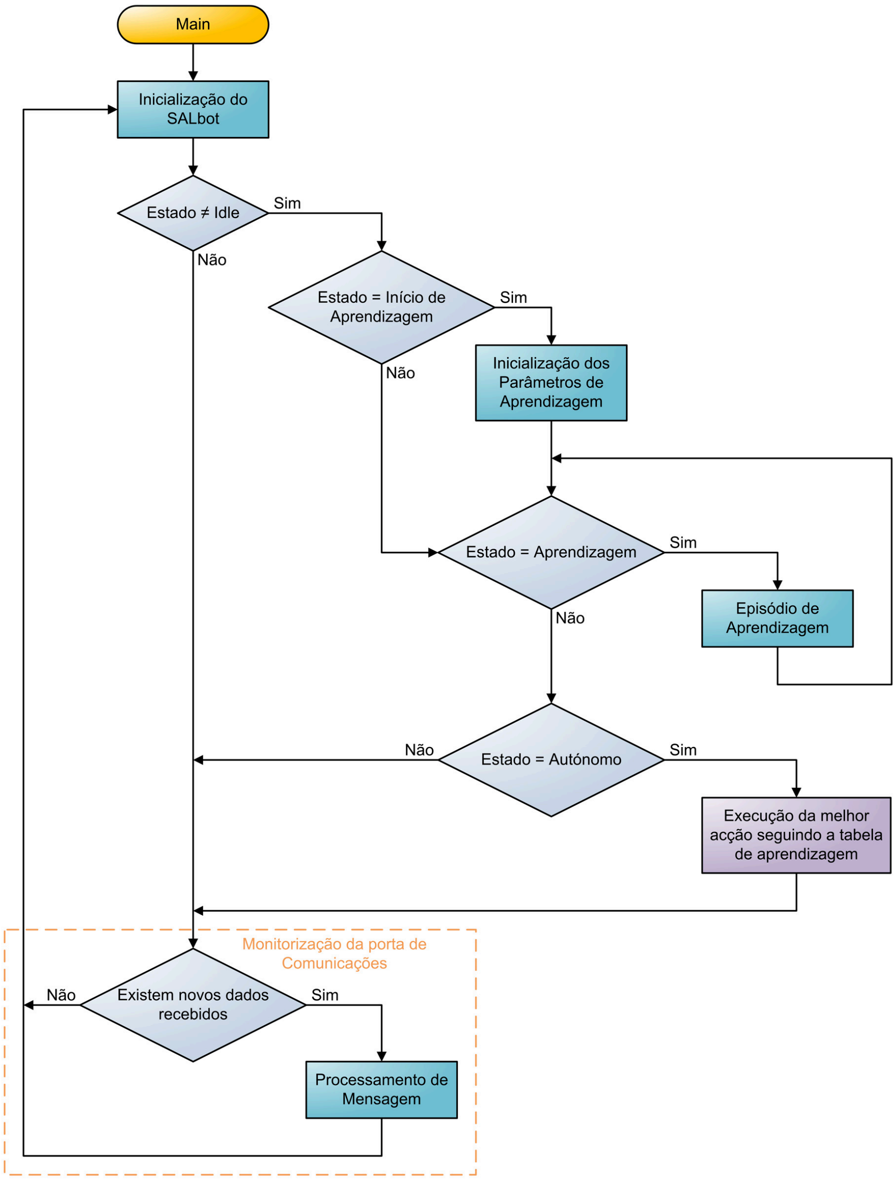


Figura 3.20: Fluxograma da rotina *Main*.

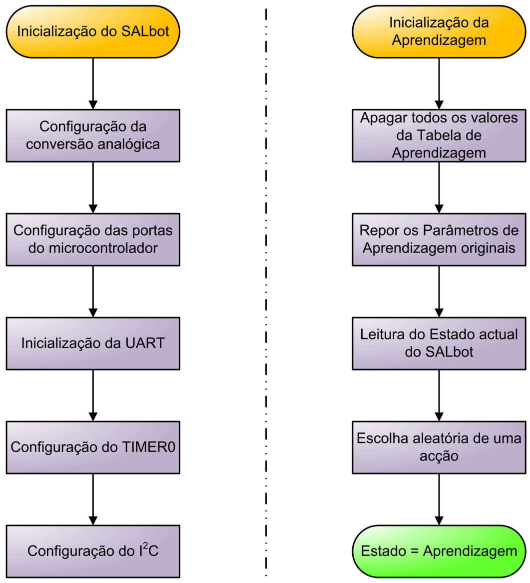


Figura 3.21: Fluxograma das rotinas: Inicialização do SALbot e Inicialização da Aprendizagem.

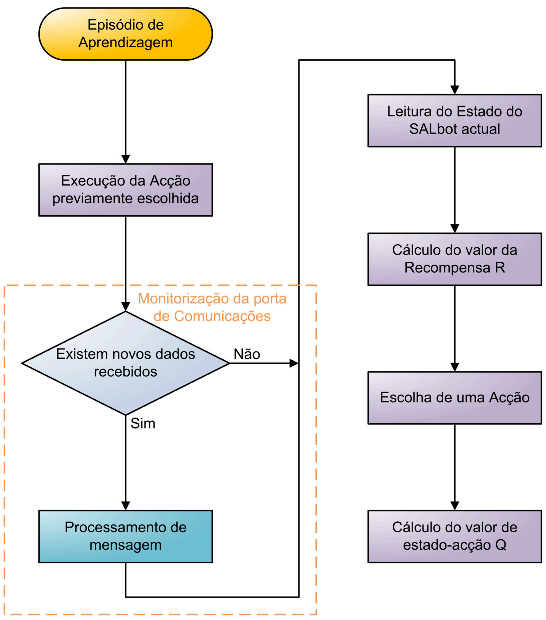


Figura 3.22: Fluxograma da rotina de Aprendizagem.

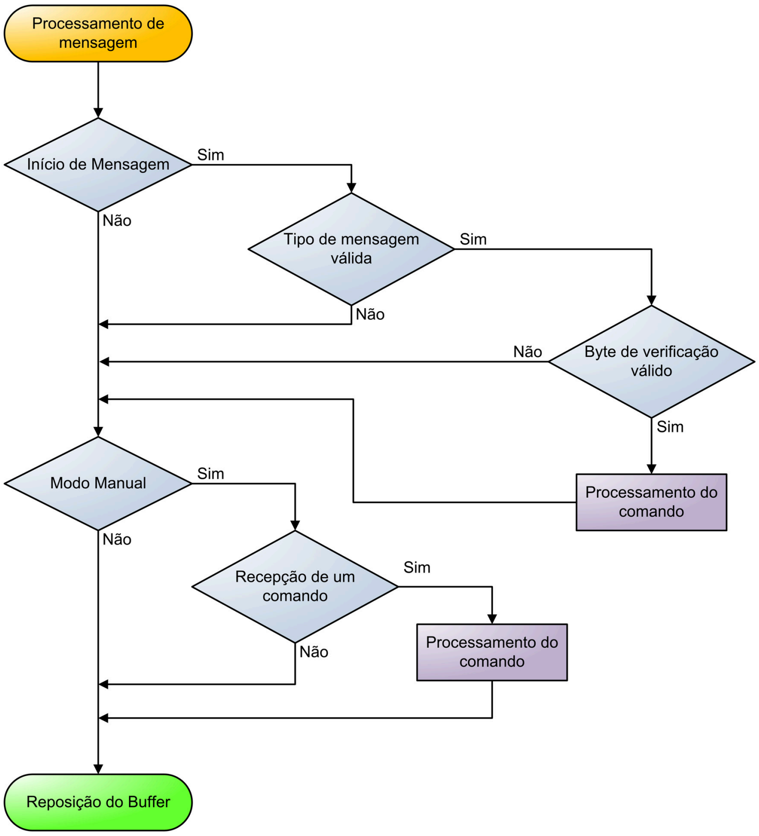


Figura 3.23: Fluxograma da rotina de Processamento de mensagens.

Capítulo 4

Testes e Ensaios

4.1 Ambiente de testes

4.2 Auto-aprendizagem

4.3 Resultados

Capítulo 5

Conclusões

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, designado por estado. Um estado, que contenha todas as informações do ambiente, é designado sendo *Markoviano*, ou tendo a propriedade *Markoviana*.[29] 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:

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

em que r é a recompensa, s o estado, s' o próximo estado e a a acção. Se o estado conter a propriedade *Markoviana*, 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:

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

Isto é, um estado inicial tem a propriedade *Markoviana*, e é um estado *Markoviano*, apenas se a equação A.1 for igual a A.2 para todo o s', r e conter 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:

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

Tanto a probabilidade de transição $\mathcal{P}^a_{ss'}$ como a recompensa esperada $\mathcal{R}^a_{ss'}$ especificam os aspectos mais importantes do dinamismo de um MDP finito.[29]

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.[29]

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:

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

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

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

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:

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

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}.[29]

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^*:

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

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:

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

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.[29] De igual forma, a equação óptima de *Bellman* para Q^* é dada por:

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

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.[29]

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.[29] 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:

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

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:

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

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 para 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.

Tabela A.1: Pseudo-código do *Monte Carlo ES*, um algoritmo do método *Monte Carlo Control* que usa a exploração inicial.[29]

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:
        $\pi(s) \leftarrow \arg\max_{a} Q(s,a)$

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 ser 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 calcula-la precisamente.

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.[29]

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

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

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

$$\epsilon\text{-}greedy \quad : \quad \pi(s,a) \begin{cases} \dfrac{\epsilon}{|\mathcal{A}(s)|} & \rightarrow \text{não maximizante} \\[2ex] 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}(s)|} & \rightarrow \textit{greedy} \text{ (ávida)} \end{cases} \tag{A.20}$$

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

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, $\tau \rightarrow 0$ 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.[28]

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:

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

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:

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

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

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 de as 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*[29].

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.

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 política. 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:

Tabela A.3: Pseudo-código do algoritmo do método *on-policy Monte Carlo Control*.[29]

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:

$a^* \leftarrow \arg\max_a Q(s,a)$

$\forall\ a \in \mathcal{A}(s)$:

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

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

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:

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

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*.[29]

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:
$s_0, a_0, r_1, s_1, a_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T$
$\tau \leftarrow$ 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 aprender através da experiência directa com o ambiente sem a necessidade ne conhecer a sua dinâmica. E os métodos de diferença temporal também estimam e actualizam baseando-se, em parte, de outras 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 tamanho do passo. O algoritmo descrito na tabela A.5 especifica o método TD(0).

Tabela A.5: Pseudo-código do algoritmo TD(0).[29]

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*[32], 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 se poder modelar a incerteza num sistema, que resulta em parte da dinâmica do ambiente que não é controlável pelo agente, é considerada a probabilidade para obter um 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 que, um agente como sendo um robô que, por exemplo, devido a variações de temperatura, por vezes se desvia da trajectória que escolheu seguir, ou avança menos do que esperava, então temos:

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

Sendo assim, não é possível fazer uma atribuição directa, 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.

O *Q-learning* tornou-se no método de *reinforcement learning* mais popular devido ao facto de ser um algoritmo "insensível à exploração", isto é, apesar de também no *Q-learning* existir a problemática *exploration-exploitation*, a convergência de aprendizagem não depende da política de exploração.[16]

Tabela A.6: Pseudo-código do algoritmo *Q-learning*.[29]

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 usando uma política derivada de Q (por ex.: ϵ-*greedy*)
Executar a acção a
Observar o próximo estado s' e a recompensa r
$Q(s, a) \leftarrow Q(s, a) + \alpha\Big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\Big]$
$s \leftarrow s'$

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 similar com o 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:

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

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.

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 $\epsilon = \frac{1}{t}$.[29]

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

Tabela A.7: Pseudo-código do algoritmo SARSA.[29]

Inicializar:

$Q(s, a) \leftarrow$ arbitrário

Repetir para cada episódio:

Inicializar s

$a \leftarrow$ tendo s usando uma política derivada de Q (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' usando uma política derivada de Q (por ex.: ϵ-*greedy*)

$Q(s, a) \leftarrow Q(s, a) + \alpha\Big[r + \gamma Q(s', a') - Q(s, a)\Big]$

$s \leftarrow s'$

$a \leftarrow a'$

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,

Tabela A.8: Pseudo-código do algoritmo *Policy Iteration*.[29]

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 $\Delta \leq \theta$ (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.

o que faz com que aumente significativamente a rapidez da sua convergência.[29] 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]

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 converja 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 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*.[29]

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

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

Repetir, até $\Delta \leq \theta$:

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 ciclo. Uma rápida convergência pode ser alcançada entrepondo vários ciclos de avaliação de política entre cada aperfeiçoamento de política.[29]

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, Setembro 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, Janeiro 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, Agosto 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] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008.

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

[28] 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.

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

[30] 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.

[31] 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.

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