
Universidade de Évora

Escola de Ciências e Tecnologia

**Mestrado em Engenharia Informática**

# NXT Mindstorms e Aprendizagem por Reforço

João Paulo Carracha Coelho

**Orientador**
Teresa Cristina Freitas Gonçalves

Évora, Outubro 2011

**Mestrado em Engenharia Informática**

# NXT Mindstorms e Aprendizagem por Reforço

João Paulo Carracha Coelho

**Orientador**
Teresa Cristina Freitas Gonçalves

# Sumário

A aprendizagem por reforço é uma aprendizagem por tentativa e erro, onde o agente, através da interação com o ambiente, aprende a realizar uma tarefa com base em recompensas positivas e negativas.

Esta dissertação pretende analisar o comportamento de um robô implementado com um algoritmo de aprendizagem por reforço cujo objetivo consiste em seguir um percurso. Para tal, foi utilizado o robô educacional criado pela Lego, o *NXT Mindstorms*, implementado com um algoritmo de aprendizagem por reforço, o *Q-learning*, utilizando os métodos de pesquisa *Softmax* e $\epsilon$-*greedy*. Para programar o robô utilizou-se a linguagem de programação *lejOS NXJ*.

Realizaram-se várias experiências com o objetivo de determinar a influência das variáveis do algoritmo *Q-learning* (taxa de aprendizagem e fator de desconto) e dos métodos de pesquisa *Softmax* (temperatura) e $\epsilon$-*Greedy* (taxa de exploração), dos valores da função de recompensa e da utilização de vários percursos.

Concluiu-se, através das experiências realizadas, que um robô implementado com um algoritmo de aprendizagem por reforço consegue aprender a tarefa em poucas iterações (inferior a 100 iterações). Também é possível concluir, através da experiência para determinar a influência das variáveis do algoritmo e dos métodos de pesquisa, que o robô apresenta melhores resultados quando o valor da taxa de exploração diminui. Conclui-se que, para todas as experiências realizadas neste trabalho o método de pesquisa *Softmax* apresenta melhores resultados em relação ao $\epsilon$-*Greedy* e o robô tem melhor desempenho no percurso em linha recta.

Como trabalho futuro pretende-se implementar outro algoritmo de aprendizagem por reforço, como o *State-Action-Reward-State-Action* (SARSA) e outros métodos de pesquisa com o objetivo de determinar qual o mais indicado para esta tarefa. Pretende-se também construir um robô para outras tarefas como seguir uma fonte de luz ou som.

*NXT Mindstorms and Reinforcement Learning*

# Abstract

Reinforcement learning is learning by trial and error, where the agent through interaction with the environment, learn a task based on positive and negative rewards.

This master's thesis aims to analyze the behavior of a robot implemented with a reinforcement learning algorithm whose goal is to follow a route. For this purpose was used the educational robot created by Lego, the *NXT Mindstorms*, implemented with a reinforcement learning algorithm, the Q-learning, using the research methods, *Softmax* and $\epsilon$-*Greedy*, implemented with the programming language *lejOS NXJ*.

Were performed several experiments in order to determine the influence of the variables of *Q-learnig* algorithm (learning rate and discount factor) and research methods *Softmax* (temperature) e $\epsilon$-*Greedy* (exploration rate), the values of reward function and the uses of several routes.

It was concluded, through the experiments, that a robot implemented with a reinforcement learning algorithm can learn the task in a few iterations (less than 100 iterations). It is also possible to conclude, through the experiment to determine the influence of the variables of the algorithm and of the methods of research, that the robot produces better results when the value of exploration rate decreases. It also conclude that for all experiments in this work the research method *Softmax* produces better results compared to the $\epsilon$-*Greedy* and the robot has better performance following a straight line.

As future work we intend to implement another reinforcement learning algorithm, such as SARSA and other research methods in order to determine the best to this task. The aim is also build a robot to perform other tasks such as following a light or sound source.

# Agradecimentos

Aqui deixo o meu obrigado a todas as pessoas que contribuíram para a realização desta dissertação de mestrado.

Principalmente aos meus pais, Jorge Coelho e Cristina Coelho, pelo apoio e educação que sempre me deram ao longo da minha vida. Por me incutirem o entusiasmo pelos estudos, pois sem eles nada disto seria possível.

À minha namorada pela paciência e compreensão que teve até este trabalho estar realizado. Pelo apoio que me tem dado até agora.

A todos os meus amigos que me incentivaram a nunca desistir e pelas palavras de apoio que me deram, nos momentos mais difíceis.

Aos Professores com quem me cruzei na universidade, em especial à minha orientadora, Professora Teresa Gonçalves, que me acompanhou e apoiou ao longo da realização deste trabalho. Aos Professores Francisco Coelho e Luís Rato por me transmitirem algumas ideias para a realização deste trabalho.

# Acrónimos

| | |
|---|---|
| **AA** | Aprendizagem Automática |
| **API** | *Application Programming Interface* |
| **AR** | Aprendizagem por Reforço |
| **ATMS** | *Advanced Traffic Management System* |
| **IA** | Inteligência Artificial |
| **JVM** | *Java Virtual Machine* |
| **LADAR** | *Laser Detection and Ranging* |
| **LED** | *Light Emitting Diode* |
| **MIT** | *Massachusetts Institute of Techonology* |
| **PDM** | Processos de Decisão de Markov |
| **PD** | Programação Dinâmica |
| **MC** | Monte Carlo |
| **DT** | Diferença Temporal |
| **RE** | Robótica Evolucionária |
| **RCX** | *Robotic Command Explorer* |
| **RGB** | *Red Green Blue* |
| **SARSA** | *State-Action-Reward-State-Action* |
| **SDK** | *Software Development Kit* |
| **USB** | *Universal Serial Bus* |

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

A aprendizagem por reforço é um campo da aprendizagem automática onde um agente, através de um processo de tentativa e erro, "aprende"a realizar uma tarefa específica através da maximização da recompensa que recebe ao interagir com o ambiente [24].

Segundo Mance e Stephanie [23], a aprendizagem por reforço combina os campos da programação dinâmica e da aprendizagem supervisionada para produzir sistemas mais fortes da aprendizagem automática. Por ser uma área da aprendizagem automática é considerada um ramo da inteligência artificial [59].

A programação dinâmica[1], de uma fomra simples, é um conjunto de algoritmos que podem ser usados para calcular políticas ótimas dado um modelo completo e perfeito do ambiente [21, 63].

Na aprendizagem supervisionada são fornecidos exemplos de entrada e saída correspondente, e através do processo de treino o algoritmo consegue aprender essas relações [63].

Na aprendizagem por reforço as tarefas são descritas através de funções de recompensa, em substituição de instruções específicas para cada situação ou estado, ou seja, em vez de indicar ao agente qual é a melhor acção para uma determinada situação, o agente, através de interação com o ambiente, aprende qual é essa ação.

Acerca de vinte anos surgiu um enorme interesse na utilização da aprendizagem por reforço na robótica, o que tem beneficiado a investigação relacionada com estes temas. Um dos principais objectivos da investigação na robótica era reproduzir inteligência humana em robôs, mas na altura só era possível recriar inteligência a nível de um inseto [83, 6].

[1] Do inglês, *dynamic programming*.

Por volta de 2004 [19], a aprendizagem por reforço foi usada em muitos robôs, mas a maioria foi realizada apenas em ambientes simulados. Na altura, o escasso número de experiências em ambientes reais devia-se aos problemas associados: o atraso existente na informação recolhida pelos sensores, a existência de ações não determinísticas e a alterações no ambiente.

A aprendizagem por reforço não necessita de uma entidade externa que forneça exemplos ou de um modelo da tarefa a ser executada: a única fonte informação é a experiência do agente que lhe permite adquirir uma política de ações que maximiza seu desempenho geral, como já referido.

## 1.1 Motivação

Cada vez mais, têm surgido robôs programáveis no contexto educacional, dando aos alunos oportunidade de construir e programar robôs com o objectivo de estimular a sua imaginação e criatividade para resolver problemas reais.

O fascínio pelo mundo da robótica e a ideia de construir um robô autónomo capaz de realizar inúmeras tarefas foi uma das motivações para a escolha deste tema.

A principal motivação para a realização deste trabalho foi dupla:

- por um lado, estudar uma área desconhecida (para mim) da inteligência artificial – a aprendizagem por reforço;
- por outro aplicá-la a um dispositivo físico – o robô educacional *NXT Mindstorms*.

## 1.2 Objetivos

Este trabalho pretende atingir os seguintes objetivos:

- construir um robô cujo comportamento evoluísse com o tempo através da implementação de algoritmos de aprendizagem por reforço, utilizando, como plataforma de teste, o robô educacional da Lego, o *NXT Mindstorms* e o algoritmo de aprendizagem Q-learning;
- analisar a evolução do comportamento do robô verificando a influência das diversas variáveis do problema: métodos de pesquisa e valores das variáveis associadas, função de recompensa e meio ambiente.

## 1.3 Abordagem proposta

Para atingir os objetivos propostos escolheu-se uma tarefa que o robô fosse capaz de aprender: seguir um percurso marcado.

Como plataforma de teste utilizou-se o kit educacional *NXT Mindstorms* alterado com uma nova linguagem – a *leJOS NXJ* [40], em substituição da fornecida pelo *kit* básico do *Lego NXT Mindstorms*, a *NXT-G*. Foi construído um veículo capaz seguir percursos com auxílio de dois motores e dois sensores infravermelhos, para além do bloco *NXT*.

Como algoritmo de aprendizagem implementou-se o *Q-learning* [80] e experimentaram-se dois métodos de pesquisa, o *Softmax* e o $\epsilon$-*greedy*, e realizaram-se três conjuntos de experiências para analisar a influência:

- da variação dos valores das variáveis do algoritmo *Q-learning* e dos métodos de pesquisa *Softmax* e $\epsilon$-*greedy*;
- da função de recompensa no processo de aprendizagem;
- de diferentes tipos de percurso no comportamento do robô.

## 1.4 Principais contribuições

O trabalho realizado nesta dissertação faz o estudo do comportamento de um agente inteligente implementado com um algoritmo da aprendizagem por reforço. As suas principais contribuições são:

- verificação da aplicabilidade dos algoritmos de aprendizagem na robótica. Nas diferentes experiências realizadas o robô conseguiu sempre aprender a tarefa proposta;
- o estudo da influência do método de pesquisa no processo de aprendizagem. Nas diferentes experiências realizadas o método *Softmax* obteve melhores resultados (aprendizagem em menos iterações) sistematicamente;
- o estudo da influência dos valores das variáveis e da utilização de vários percursos no processo de aprendizagem. Na experiência de verificar a influência das variáveis do algoritmo, mostrou que o robô tem um comportamento positivo quando o valor da variável da taxa de exploração (método de pesquisa $\epsilon$-*Greedy*) diminui. Nos diferentes percursos testados, o robô apresentou melhores resultados no percurso de linha recta do que nos outro percursos utilizados.

De uma forma geral, este trabalho contribuiu para uma melhor compreensão da área da aprendizagem por reforço e da influência dos diversos fatores do processo de aprendizagem aplicados à robótica.

## 1.5 Estrutura da dissertação

O Capítulo 2 apresenta um breve resumo histórico da área de aprendizagem por reforço, faz uma introdução ao tema e apresenta os seus conceitos fundamentais: a política, a função de recompensa, a função de valor e o modelo do ambiente. São apresentados os métodos de pesquisa, $\epsilon$-*greedy* e *Softmax* que permitem o equilíbrio entre as fases de exploração e examinação. São introduzidos os métodos e respetivos algoritmos utilizados na aprendizagem por reforço: a programação dinâmica, o método de Monte Carlo e método de diferença temporal. Finalmente é introduzido um dos mecanismo básicos da aprendizagem por reforço, os traços de elegibilidade.

O Capítulo 3 dedica-se à descrição de alguns dos conceitos principais sobre a robótica, sendo feito um breve resumo histórico e uma introdução ao tema, e são apresentados alguns dos robôs autónomos. É ainda descrito o robô educacional da lego, *NXT Mindstorms*, os seus componentes e as linguagens de programação com suporte do kit.

O Capítulo 4 introduz o robô construído e a configuração experimental. Segue-se a apresentação e descrição das experiências realizadas para estudar a influência dos valores das variáveis do algoritmo, dos métodos de pesquisa, da função de recompensa e do meio ambiente no comportamento do robô. Os resultados obtidos são apresentados e avaliados.

O Capítulo 5 analisa comparativamente os resultados obtidos e apresenta conclusões. É também descrito algum trabalho futuro no seguimento deste trabalho.

# Capítulo 2

# Aprendizagem por reforço

Neste capítulo é apresentado um breve resumo histórico da área de aprendizagem por reforço, fazendo uma introdução ao tema e apresenta os seus conceitos fundamentais. São apresentados métodos de pesquisa, $\epsilon$-*greedy* e *Softmax*, métodos de implementação para resolver o problema da aprendizagem por reforço e traços de elegibilidade.

## 2.1 Breve histórico

A história da aprendizagem por reforço tem três linhas distintas de investigação [63], sendo que duas delas foram desenvolvidas separadamente, a **tentativa e erro** e a **teoria do controlo óptimo**. As exceções destas duas levaram a uma terceira, conhecida como **método de diferença temporal**. Estas três linhas juntaram-se nos finais do ano de 1980 para darem origem à aprendizagem por reforço [63].

A aprendizagem por **tentativa e erro** foi iniciada no campo da psicologia, na aprendizagem animal onde, segundo Sutton e Barto [63] em 1911, surgiu a primeira investigação, feita por Thorndike [67] e as primeiras investigações computacionais da aprendizagem por tentativa e erro foram feitas por Farley e Clark [17] e por Minsky [38]. Existem dois aspetos importantes da aprendizagem por tentativa e erro: o primeiro é a **seleção**, onde através de tentativas é possível escolher ações e saber as suas consequências; o segundo a **associação**, associa as alternativas que foram encontradas pela "seleção", a situações particulares que aconteceram.

Harry Klopf [30, 31, 32] transformou esta aprendizagem, numa aprendizagem por reforço com inteligência artificial [63].

Durante as décadas de 80 e 90 a investigação na aprendizagem por tentativa e erro, tornam-se raras[63]. Isto devido às confusões, ainda hoje existentes, entre esta aprendizagem e a aprendizagem supervisionada pois ambas se baseiam no erro no processo de aprendizagem [21].

A **teoria do controlo óptimo** surgiu nos finais dos anos 50, destacando-se pelo uso da programação dinâmica, considerada como uma forma viável de resolver problemas estocásticos [63].

A programação dinâmica sofreu um enorme desenvolvimento desde essa data, pois houve muitos trabalhos importantes como o de Bellman [2, 3] e os de Bertsekas [4, 5].

Por último, temos a aprendizagem por **métodos de diferença temporal** e esta linha também surgiu na psicologia da aprendizagem animal, em particular, na noção de reforço secundário. Este reforço secundário é, por assim dizer, um estímulo associado a um reforço primário como alimento ou dor [21].

Os primeiros trabalhos de aprendizagem por reforço foram realizados no início da Inteligência Artificial (IA) e da Cibernética e contou com o conhecimento de outras áreas, tais como, teoria do controlo, teoria de informação, estatística, psicologia, algoritmos genéticos, economia, teoria de jogos, neurociência e ciência dos computadores [27].

As áreas da teoria do controlo e da estatística, que ficavam entre IA e a Engenharia Convencional estão agora entre as mais ativas incluindo novos campos, como redes neuronais, controlo inteligente e aprendizagem por reforço [63].

Segundo Sutton e Barton [63], o conceito de aprendizagem por reforço foi utilizado pela primeira vez, na literatura de engenharia, nos anos 60. Em 1961, Minsky [39] publicou um artigo onde examina alguns problemas da aprendizagem por reforço [36, 37, 63, 78].

Existem diversas áreas de aplicação da aprendizagem por reforço. Essas aplicações vão desde a robótica, à fabricação industrial, a problemas de pesquisas combinatórios, tais como jogos de computador [27].

No campo dos jogos de computador existem vários jogos implementados com esta aprendizagem, como por exemplo, o jogo de damas [27, 55], o jogo do gamão usando diferença temporal [64, 65, 66], o jogo go [57] e o xadrez [68].

No campo da robótica existem vários trabalhos, um deles, de Schaal e Atkeson [56], relata a construção de um robô com dois braços, com a tarefa aprender a manobrar um bastão chinês[1]. Outro robô implementado com aprendizagem por reforço foi o Obelix que tinha como tarefa aprender a empurrar caixas para que este conseguisse andar livremente pelo ambiente; este trabalho foi feito por Mahadevan [34]. Mataric [35] descreve uma experiência robótica onde quatro robôs, se encontram num ambiente fechado, tendo a tarefa de reunir pequenos discos e transportá-los para um determinado destino.

[1] Do inglês, *devil-stick*.

A aprendizagem por reforço também foi utilizada para otimizar a performance de elevadores [12, 48, 85] e no controlo de tráfego [10, 33, 44].

Outros trabalhos interessantes no campo da aprendizagem por reforço são [19, 20, 50, 51, 62, 70, 71, 72, 75, 76, 81].

## 2.2 Introdução

A aprendizagem por reforço é uma área da aprendizagem automática[2]. Segundo Haykin [24], existem duas formas de aprendizagem: a aprendizagem com professor e a aprendizagem sem professor. Na aprendizagem com professor existe a aprendizagem supervisionada; o método da aprendizagem sem professor divide-se em dois grupos: a aprendizagem não supervisionada e a aprendizagem por reforço.

Na aprendizagem supervisionada são fornecidos exemplos de entrada e saída correspondente, e o objetivo do algoritmo é aprender essas relações, isto é, a partir destes exemplos de entrada e saída, o algoritmo aprende a associar cada vetor de entrada de $\boldsymbol{x}$ ao vetor de saída $\boldsymbol{y}$. Este processo é conhecido como treino [24]. Como é uma aprendizagem com professor, este necessita de ter um conhecimento prévio sobre o ambiente [21, 24]. Esta aprendizagem só está concluída quando o algoritmo responde de forma correta a situações novas ou desconhecidas. A figura 2.1 ilustra o processo de aprendizagem supervisionada:

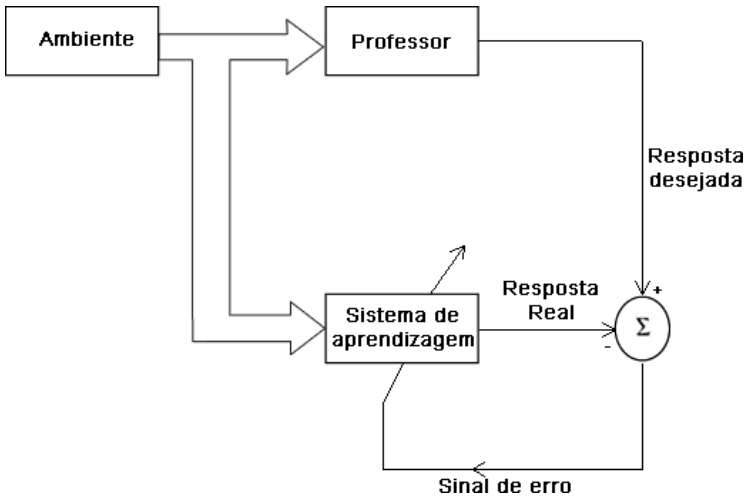



Figura 2.1: Aprendizagem supervisionada

Na aprendizagem não supervisionada não é fornecido ao agente qualquer tipo informação, não existindo professor para orientar o processo de aprendizagem [21, 71]. O agente tem como principal objetivo descobrir as relações, as caraterísticas ou padrões, ou mesmo as

[2]Do inglês, *machine learning*.

diferenças em agrupamentos, a partir dos quais consegue tirar conclusões com base em processos de auto-organização[3] por tentativa e erro.

Este tipo de aprendizagem é apropriado para aplicações de mineração de dados[4], pois os valores dos dados não são conhecidos antecipadamente. A figura 2.2 ilustra o processo de aprendizagem não supervisionada.

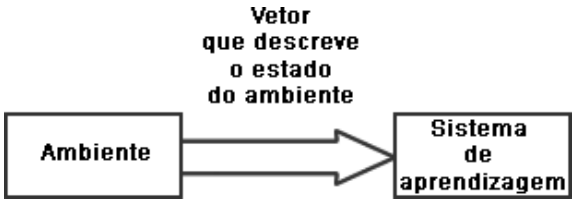



Figura 2.2: Aprendizagem não supervisionada

A aprendizagem por reforço é uma aprendizagem por tentativa e erro, onde o agente, através da interacção com o ambiente, aprende a realizar uma tarefa com base em recompensas positivas e negativas [21, 24, 63, 71]. Segundo Sutton e Barton[63], esta aprendizagem é um formalismo da IA que permite ao agente aprender através da interação com o ambiente; Na Aprendizagem por Reforço (AR) o agente aprende através experiências diretas.

As tarefas na aprendizagem por reforço são descritas através de funções de recompensa, em substituição de instruções específicas para cada situação ou estado, isto é, ao em vez de indicar a melhor ação para uma determinada situação, o agente através de interação com o ambiente, aprende qual a melhor ação para essa situação [54, 63].

Pode-se considerar que a aprendizagem por reforço abrange toda a IA: um agente é colocado num ambiente e este tem de aprender a comportar-se com sucesso nele [54]. Na aprendizagem por reforço o agente tem objetivos explícitos: este analisa as caraterísticas do ambiente que o rodeia e consegue escolher ações que influenciam esse mesmo ambiente [63].

Segundo Ribeiro [51], esta aprendizagem contém um calcanhar de Aquiles, pois, para o agente, diferentes estados podem ter a mesma representação. Como consequência ser-lhes-á atribuído o mesmo custo, mesmo quando não deveria ser feito.

Uma das mais surpreendentes descobertas na aprendizagem por reforço, é o facto de existirem algoritmos simples onde o agente pode aprender uma política óptima sem que seja capaz de prever os efeitos dessas ações ou das respetivas recompensas [13].

Resumindo, na aprendizagem por reforço o aluno é o agente que decide qual ação, ou ações a tomar num ambiente desconhecido e recebe uma recompensa ou uma penalidade por essa ação na tentativa de resolver o problema. Após um conjunto de tentativas, o

[3] Do inglês, *self-organization*.
[4] Do inglês, *data mining*.

agente deve aprender qual a melhor política, ou seja, a sequência de ações que maximizam a recompensa total. A figura 2.3 ilustra o processo de aprendizagem por reforço:

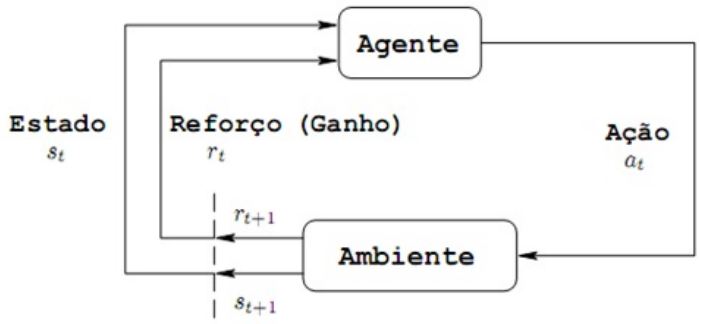



Figura 2.3: Aprendizagem por Reforço

Um agente executa uma ação, $a_t$, num estado, $s_t$, seguindo uma política, $\pi$, para avançar para o próximo estado, $s_{t+1}$, com base na recompensa, $r_t$, que essa ação irá receber.

## 2.3 Conceitos fundamentais da aprendizagem por reforço

A **política**[5], a **função de recompensa**[6], a **função de valor**[7] e o **modelo do ambiente**[8] são alguns conceitos fundamentais da aprendizagem por reforço:

### 2.3.1 Política

A política é uma regra estocástica que define o comportamento do agente, ou seja, indica ao agente qual a melhor ação a tomar num determinado estado. Esta é representada por uma função $\pi(s, a)$ que analisa o estado, $s$, e a ação, $a$ e que corresponde à probabilidade do agente tomar a ação $a \in A(S)$ quando este se encontrar no estado $s \in S$ [63]. Por outras palavras, uma política é uma distribuição de probabilidades da ações $a$ em cada estado $s$. Esta pode ser simplesmente uma função ou uma tabela, ou noutros casos, pode envolver cálculos complexos. A política constitui o núcleo do agente no sentido em que sozinha é capaz de determinar o comportamento deste.

### 2.3.2 Função de recompensa

A função de recompensa define o objetivo a atingir num sistema de aprendizagem por reforço. O agente associa um estado (ou par estado-ação) do ambiente a uma recompensa.

[5]Do inglês, *policy*.
[6]Do inglês, *reward function*.
[7]Do inglês, *value function*.
[8]Do inglês, *model of the environment*.

Esta função indica o que é melhor no sentido imediato, enquanto que a **função de valor** específica o que é melhor a longo prazo. Resumindo o principal objetivo de um agente é maximizar a recompensa total que vai acumulando em longo prazo. Esta função pode ser estocástica.

### 2.3.3 Função de valor

A função de valor indica o ganho total acumulado no futuro. Todos os algoritmos de aprendizagem por reforço são baseados na estimativa de funções de valor. Existem dois tipos de funções de valor, a função estado-valor, $V^{\pi}(s)$, e a função ação-valor, $Q^{\pi}(s,a)$. Nos processos de decisão de Markov [63], a função $V^{\pi}(s)$, define-se como:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}, \tag{2.1}$$

onde $V^{\pi}(s)$ representa o valor esperado que o agente recebe quando segue a política $\pi$. A função ação-valor, $Q^{\pi}(s,a)$, considera o par estado-ação $(s,a)$, e é definida como

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}, \tag{2.2}$$

onde $a$ representa uma ação no estado $s$ sobre a política $\pi$, num determinado instante $t$.

Um processo de decisão de Markov é uma forma de configurar processos, onde as transições entre os estados são consideradas probabilísticas.

Problemas com recompensas atrasadas[9] são bem modelados com Processos de Decisão de Markov (PDM). Diz-se que é recompensa atrasada quando o agente está num determinado ambiente e recebe uma recompensa igual zero, excepto quando este alcança um dos estados ótimos ou finais. O agente tem de aprender a sequência das ações que o levam a recompensas futuras no estado óptimo.

Segundo Ribeiro [51], uma das principais características de um PDM é a sua condição[10]. Diz-se que um processo estocástico tem a condição de Markov, se as distribuições de probabilidades condicionais de estados futuros, dependerem apenas do estado presente, isto é, o passado é irrelevante, pois não interessa como o estado actual foi obtido[63]. A condição de Markov é muito importante na aprendizagem por reforço porque as decisões e os valores estão assumidos serem uma função apenas do estado atual.

Um **processo de decisão de Markov** é um tuplo $(S, A, T, R)$, onde:

- $S$ é o conjunto de todos os estados finitos do ambiente;

[9] Do inglês, *delayed reward*.
[10] Do inglês, *Markov condition*.

- $A$ é o conjunto de todas as ações finitas possíveis;
- $T : SxA \rightarrow \pi(S)$: é a função de transição de estados, onde $\pi(S)$ é a distribuição de probabiliade sobre o conjunto $S$. Segundo Kaelbling e Moore[27], escreve-se $T(s, a, s')$ para a probabilidade fazer um transição do estado $s$ para o estado $s'$ com a ação $a$;
- $R : SxA \rightarrow R$ é a função de recompensa imediata.

### 2.3.4 Modelo do ambiente

O modelo do ambiente é algo que imita o comportamento do ambiente, por exemplo, dados um estado e uma recompensa, o modelo pode prever o próximo estado e a sua recompensa [63]. O modelo do ambiente é usado para decidir que ação é tomada pelo agente, considerando a possibilidade de situações futuras antes de serem realmente exprimentadas. O ambiente deve ser parcialmente observável pelo agente através de sensores, descrições simbólicas ou a informação relevante sobre ele.

## 2.4 Métodos de pesquisa

Um dos desafios existentes na aprendizagem por reforço e não em outros tipos de aprendizagens foi o facto de haver um conflito entre a Exploração[11] e a Examinação[12][63].

Na aprendizagem por reforço deve existir um equilíbrio entre a exploração[13] e a examinação[14].

Na examinação o agente escolhe a ação com maior recompensa para um determinado estado; à primeira vista parece ser uma boa opção, mas impede o agente de procurar melhores ações.

A exploração permite obter novas informações sobre o ambiente com o objetivo de alcançar melhores níveis de desempenho no futuro, ou seja, o agente explora ações ainda não experimentadas ou estados não visitados, para ter uma visão mais abrangente do ambiente. Sem exploração o melhor objetivo não será encontrado. Em contrapartida, se o agente explorar muito não consegue manter um caminho [11].

Se for selecionada uma das ações diferente da ação com maior recompensa, diz-se que o agente está a explorar, permitindo assim, melhorar a estimativa dos valores das ações que não tenham a maior recompensa. Para maximizar a recompensa esperada numa iteração escolhe-se a examinação; no entanto com a exploração pode-se produzir uma maior recompensa total a longo prazo.

[11] Do inglês, *exploration*.
[12] Do inglês, *exploitation*.
[13] Do inglês, *exploration*.
[14] Do inglês, *exploitation*.

Uma das grandes diferenças entre aprendizagem por reforço e aprendizagem supervisionada, consiste no simples facto do agente da aprendizagem por reforço para "aprender"ter que, explicitamente, explorar o ambiente em que se encontra [27].

Existem vários métodos que permitem o equilíbrio entre a examinação e a exploração. Os mais conhecidos são o método $\epsilon$-*Greedy* e o *Softmax* [21, 63].

O método $\epsilon$-*Greedy* na maioria das vezes escolhe a ação com maior recompensa mas, de vez em quando, aleatoriamente é escolhida uma ação independentemente da sua recompensa. A ação com maior recompensa, é escolhida com a probabilidade de $1-\epsilon$, e a ação aleatória é escolhida com a probabilidade $\epsilon$. Isto significa que para a ação $a^*$, ação com maior recompensa, a política é $\pi_t(s, a^*) = 1 - \epsilon$.

Uma desvantagem do método $\epsilon$-*Greedy* é que ao escolher uma ação aleatória, tanto pode escolher a ação com a melhor recompensa como a ação com a pior. O método *Softmax* foi desenvolvido para ultrapassar esta falha atribuindo um peso para cada ação, de acordo com a sua estimativa ação-valor [21]. Assim, uma ação é selecionada de acordo com o peso que lhe está associado tornando improvável a escolha da pior ação já que a ação com maior valor de recompensa, tem maior probabilidade de ser escolhida. Para calcular as probabilidades é utilizada a distribuição de Gibbs, sendo a política dada pela equação:

$$\pi_t(a) = \frac{e^{\frac{Q_t(a)}{T}}}{\sum_{b=1}^{n} e^{\frac{Q_t(a)}{T}}} \tag{2.3}$$

onde $T$ é um valor positivo designado por temperatura. Quando a temperatura for alta, qualquer ação tem (quase) a mesma probabilidade de ser escolhida. No caso de a temperatura ser baixa, essa probabilidade desaparece, fazendo com que as ações sejam escolhidas de acordo com as suas estimativas de valor.

## 2.5 Métodos de implementação

Existem três métodos para resolver o problema da aprendizagem por reforço: o método da programação dinâmica, o método de Monte Carlo e por último método da diferença temporal [63].

### 2.5.1 Métodos de programação dinânimca

A programação dinâmica[15] refere-se a um conjunto de algoritmos que podem ser usados para calcular políticas ótimas dado um modelo completo e perfeito do ambiente, como os modelos de PDM [63].

A programação dinâmica é muito importante, pois fornece uma base essencial para uma

[15] Do inglês, *dynamic programming*.

melhor compreensão dos restantes métodos. Na realidade, os métodos seguintes podem vistos como uma tentativa de chegar ao mesmo efeito da programação dinâmica, mas com menos cálculos e sem assumir um modelo completo e perfeito [24, 63].

A principal ideia deste método é usar funções de valor para organizar e estruturar a procura de boas políticas.

Pode-se facilmente obter políticas ótimas quando são descobertas as funções de valor, $V^*$ ou $Q^*$, que satisfaçam as equações de Bellman[2]:

$$\begin{aligned} V^*(s) &= \max_a E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\right\} \\ &= \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s')\right] \end{aligned} \tag{2.4}$$

ou

$$\begin{aligned} Q^*(s,a) &= E\left\{r_{t+1} + \gamma \max_a Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\right\} \\ &= \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_a Q^*(s', a')\right], \end{aligned} \tag{2.5}$$

para todo o $s \in S$, $a \in A(S)$, e $s' \in S^+$. Os algoritmos de programação dinâmica são obtidos por transformação das equações de Bellman.

### Iteração da política

As ações escolhidas por um agente implementado com um sistema de aprendizagem por reforço são feitas a partir de uma função chamada política $\pi$ [63].

Na iteração da política, a política é avaliada e depois melhorada[13]. A **avaliação da política**[16] consiste em calcular o valor de todos os estados, $s$, seguindo uma política, $\pi$:

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \mid s_t = s\right\} \\ &= E\left\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\right\} \qquad (2.6) \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s')\right], \qquad (2.7) \end{aligned}$$

onde $\pi(s,a)$ é a probabilidade de uma ação, $a$, ser executada num estado, $s$, sobre a política $\pi$. $V^\pi$ expressa a relação entre o valor de um determinado estado e dos valores dos próximos estados.

A **política melhorada**[17] usa os valores de $V^*(s)$[18] para especificar a política nova $\pi'$ que é tão boa, ou melhor que a política atual $\pi$ [13].

[16] Do inglês, *policy evaluation*.
[17] Do inglês, *policy improvement*.
[18] Valor do estado $s$ nos termos da política optima.

A avaliação da política não permite saber se a política é melhor ou pior que a atual. Para isso basta realizar uma ação $a$ no estado $s$ seguindo a política $\pi$ sendo o valor esperado:

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s, a_t = a\right\} \\ &= \sum_{s'} P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma V^{\pi}(s')\right]. \end{aligned} \tag{2.8}$$

A nova política é

$$\pi'(s) = \arg max_a \left\{Q^{\pi}(s,a)\right\}, \tag{2.9}$$

O critério fundamental é maior ou menor que $V^{\pi}(s)$. Este resultado chama-se melhoramento da política. Sejam $\pi$ e $\pi'$ um par de políticas determinista de tal forma que, para todo o $s \in S$ [63],

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s). \tag{2.10}$$

Se a política $\pi$ e $\pi'$ forem iguais então é uma política óptima. Os valores $V^{\pi}(s)$ e $Q^{\pi}(s,a)$ associados à política ótima são chamados valores ou ações ótimas e são denominados de $V^{*}(s)$ e $Q^{*}(s,a)$ [13].

O método da iteração da política vai melhorando a cada passo, até que este encontre uma política ótima. No pior caso, a iteração de política tem um número exponencial de passos, mas normalmente é muito rápido [63].

### Iteração do valor

Este método é a principal alternativa à iteração de política onde, em cada iteração, tem que ser feita uma avaliação da política, o que pode ser um cálculo demorado [63].

Na iteração do valor a função do valor ótimo é aproximado por um processo de iteração que atualiza um valor estimado de $V^{\pi}(s)$, que vai eventualmente convergir para $V^{*}$ [63].

A iteração de valor pode ser escrita como uma simples operação de reserva, que combina a política melhorada e os passos incompletos da avaliação política:

$$\begin{aligned} V_{k+1}(s) &= \max_a E\left\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, a_t = a\right\} \\ &= \max_a \sum_{s'} P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma V_k(s')\right], \end{aligned} \tag{2.11}$$

Para $V_0$ arbitrário, a sequência $\{V_k\}$ converge para $V^{*}$ nas mesmas condições que garantem $V^{*}$ [63].

A iteração da equação, (2.11), encontra os valores óptimos. A equação, (2.9), pode ser utilizada para depois encontrar a política ótima associada a esses valores.

### 2.5.2 Método de Monte Carlo

Ao contrário da programação dinâmica, o método de Monte Carlo não precisa de ter informação completa do ambiente. Este apenas precisa de experiências, uma amostra de sequências de estados, ações e recompensas, realizadas através da interação com o ambiente.

Para garantir que as recompensas sejam bem definidas, o método de Monte Carlo é um método por tarefas episódicas, isto é, as experiências são divididas em episódios, onde todos eventualmente têm um fim, não importando as ações que são escolhidas. A política e a estimativa de valor são alteradas quando um episódio termina.

É um método incremental num sentido de episódio-por-episódio, mas não no sentido de passo-por-passo [63].

#### Avaliação da política de Monte Carlo

Este método vai aprender a função estado-valor para uma política dada. Em particular estima $V^{\pi}(s)$, o valor de um estado $s$ seguindo a política $\pi$ [63].

Uma forma óbvia de estimar o valor de um estado, é calcular a média das recompensas observadas após a visita do estado em questão [9].

À medida que trajetórias são produzidas, os ganhos são observados e a média deve convergir para o valor esperado.

Existem dois tipos dos métodos de Monte Carlo, em relação à avaliação de política: o método de Monte Carlo cada-visita[19] e o método de Monte Carlo primeira-visita[20]. No método cada-visita será feita uma estimativa de $V^{\pi}(s)$, função valor-estado, através da média das recompensas após todas as visitas ao estado $s$ num conjunto de episódios; no método primeira-visita é feita uma média das recompensas após a primeira visita ao estado $s$.

### 2.5.3 Método de diferença temporal

Combinando as vantagens da programação dinâmica e do método de Monte Carlo, a aprendizagem por Diferença Temporal (DT)[21] destaca-se como uma ideia central e singular na aprendizagem por reforço [9, 21, 63].

Este método baseia-se no equilíbrio entre exploração e examinação, e exsitem dois métodos:

[19] Do inglês, *every-visit*.
[20] Do inglês, *first-visit*.
[21] Do inglês, *Temporal-Difference*.

política-ligada[22], conhecida como SARSA, e política-desligada [23], conhecida como *Q-learning*.

Como o método de Monte Carlo, os métodos de diferença temporal podem aprender diretamente de uma experiência sem precisar de um modelo de ambiente [63]. Como a programação dinâmica, os métodos de diferença temporal atualizam estimativas baseadas, em parte, em outras estimativas aprendidas, sem esperar pelo resultado final [24, 63].

### Previsão do método de diferença temporal

Os métodos de diferença temporal e de Monte Carlo usam experiências para resolver problemas de previsão. O método de Monte Carlo espera até que o ganho total seja conhecido, para depois atualizar $V(s_t)$. Uma simples equação do método de Monte Carlo de cada-visita apropriada para um ambiente não estacionário é:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t)\right], \tag{2.12}$$

Onde $R_t$ é o actual ganho obtido a partir do instante $t$ e $\alpha \in ]0, 1[$.

Os métodos de diferença temporal apenas precisam de esperar até ao próximo instante [63], pois é nesse instante, $(t+1)$, que é formado um objetivo e é feita uma atualização usando as recompensas $r_{t+1}$ e valor estimado de $\mathrm{V}(s_{t+1})$ [9].

O método de diferença temporal, conhecido por $TD(0)$, é:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1} - V(s_t)\right], \tag{2.13}$$

O objetivo dos métodos de diferença temporal é atualizar o valor $r_{t+1}+\gamma V(s_{t+1})$, enquanto que o objetivo do método de Monte Carlo é atualizar o valor $R_t$.

Resumindo, a vantagem mais óbvia do método de previsão de diferença temporal, em relação aos métodos programação dinâmica, é que não precisa de modelos de ambiente, das recompensas e das probabilidades de transições entre os estados [9, 63]. Em relação ao método de Monte Carlo, os métodos de previsão de diferença temporal encontram as atualizações das estimativas no estado seguinte, enquanto o método de Monte Carlo aguarda até ao estado final para obter o verdadeiro ganho e atualizar as suas estimativas.

### SARSA

A política ligada, ***SARSA***, foi introduzida em 1992 por Rummery e Niranjan [52]. Tem este nome pois a função principal para atualizar o valor de $Q^\pi$ depende do estado onde se encontra o agente ***"S1"***, a ação que este escolhe ***"A1"***, a recompensa que ganha por

[22] Do inglês, *on-policy*.
[23] Do inglês, *off-policy*.

escolher a ação ***"R"***, o estado seguinte ***"S2"*** onde o agente se encontra por ter executado aquela ação e, finalmente, a próxima ação ***"A2"*** escolhida neste novo estado, ou seja depende do tuplo $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, o que gera o termo ***SARSA***.

O agente interage com o ambiente e atualiza a política, baseado nas ações que foram realizadas:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right], \tag{2.14}$$

Onde a atualização do valor de $Q(s_t, a_t)$ é feita após cada transição de um estado não terminal, denominado por $s_t$. Se o próximo estado, $s_{t+1}$, for o estado final, então $Q(s_{t+1}, a_{t+1})$ é definido como zero. Esta regra usa todos os elementos do quíntuplo de eventos, isto faz com que haja uma transição de um par de estado-ação para o próximo.

Neste método, as transições são idênticas às cadeias de Markov, mas existe um processo de recompensa[63].

### *Q-learning*

O algoritmo Q-learning foi apresentado por Watkins em 1989 [80], sendo considerado um dos maiores avanços na área da aprendizagem por reforço já que se trata de um algoritmo que aprende a política ótima sem criar um modelo do ambiente [63].

Este algoritmo amplia a função valor tornando-a numa função ação-valor que, para cada estado, armazena o valor de todas ações.

Esta função, $Q^*(s, a)$, expressa, num comportamento ótimo, o valor da ação $a$ realizada no estado $s$. A relação entre a função valor e a função ação-valor é definida como

$$V^*(s) = \max_a Q^*(s, a) \tag{2.15}$$

A política é definida de acordo com a equação

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{2.16}$$

e a atualização da função ação-valor é dada pela expressão

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right] \tag{2.17}$$

onde $Q$ é a tabela de valores dos pares estado-ação, $a_t$ é a ação actual, $s_t$ o estado atual, $s_{t+1}$ é o novo estado que resulta da ação no estado atual, $\alpha \in ]0, 1[$ corresponde à taxa de aprendizagem, $\gamma \in ]0, 1[$ é o fator de desconto, $\max_a Q(s_{t+1}, a_t)$ é a ação com maior recompensa da tabela $Q$ e $r_{t+1}$ é a recompensa que o agente recebe no próximo estado.

Teoricamente a taxa de aprendizagem, $\alpha \in ]0,1[$, determina em que medida as novas informações irão substituir as informações antigas. Quando o valor for igual a 0, o agente não consegue aprender nada, contudo se o valor for igual a 1 vai fazer com que o agente só considere as informações mais recentes.

O fator de desconto, $\gamma \in ]0,1[$, determina a importância das recompensas futuras. Se o valor do fator de desconto for igual a 0, o agente considera, apenas as recompensas atuais, se o fator de desconto for igual a 1 o agente irá lutar por uma recompensa alta, a longo prazo.

Segundo Jaakkola [26], este método é bastante conhecido pois é um método pioneiro que verdadeiramente estuda o método da aprendizagem por reforço pelo propósito do controlo.

Para Ribeiro [51], este apresenta a técnica mais simples que calcula diretamente a política ótima da ação sem uma fase de custo intermediário de avaliação[24] e sem precisar de um modelo do ambiente, como já foi referido anteriormente.

O algoritmo *SARSA* e *Q-learning* são muitos semelhantes e as suas diferenças são mínimas: o *SARSA* usa a política para atualizar os valores de $Q(s,a)$, ou seja, os valores de $Q(s,a)$ são atualizados com base na política que está a ser seguida; O *Q-learning* atualiza os valores utilizando a política de examinação.

### 2.5.4 Considerações Finais

Cada método de implementação tem as suas vantagens e desvantagens. A programação dinâmica está matematicamente bem desenvolvida, mas necessita de um modelo de ambiente completo e preciso. O método Monte Carlo não precisa de modelo do ambiente e é relativamente simples, mas não é adequado para cálculos incrementais passo-a-passo. O método de diferença temporal não precisa de modelo de ambiente e é completamente incremental, mas é mais complexo de analisar.

## 2.6 Traços de elegibilidade

Os traços de elegibilidade é um dos mecanismos básicos da aprendizagem por reforço [63].

A ideia por detrás dos traços de elegibilidade é muito simples: cada vez que um estado é visitado, é armazenada informação temporária da ocorrência do evento, um traço, que gradualmente diminui ao longo do tempo.

Qualquer algoritmo de diferença temporal, como *Q-learning* e *SARSA*, podem ser combinados com traços de elegibilidade para obter um algoritmo que possa aprender eficazmente.

O algoritmo *Q(λ)*, proposto por Peng e Williams [47] é caraterizado por ser uma adaptação

[24] Do inglês, *intermediate cost evaluation step*.

de uso de traços de elegibilidade para o algoritmo *Q-learning*.

Quando os traços de elegibilidade são combinados com o *Q-learning*, o algoritmo resultante é chamado de Q($\lambda$) [6].

Os traços de elegibilidade, introduzidos por Klopf [30], foram usados numa variedade de sistemas de aprendizagem por reforço. É um mecanismo muito simples, pois cada vez que um estado é visitado, é iniciado um processo de memória a curto prazo, ou seja temporário, que enfraquece gradualmente com o tempo [63].

Os traços de elegibilidade ajudam a fazer ligação entre os eventos e a informação guardada do treino. Também fazem ligação entre os métodos de diferença temporal e Monte Carlo [63].

# Capítulo 3

# Conceitos de robótica

Este capítulo apresenta um breve resumo histórico e uma introdução à robótica. São apresentados alguns dos robôs autónomos, também introduz os componentes e linguagens de programação do *Lego NXT Mindstorms* e descreve trabalhos de aplicação de aprendizagem por reforço à robótica.

## 3.1 Breve Histórico

Em 1921 [73, 28], o dramaturgo Karel Capek introduziu, pela primeira vez, a palavra robô na sua peça "*R.U.R, Rossum's Universal Robots*".

O termo robótica foi utilizado pela primeira vez pelo cientista e escritor Isaac Asimov, em 1941, no romance "*Runaround*" [28]. Também publicou pequenas histórias de ficção científica onde apareciam robôs, sendo elas "*I Robot*"(1950), "*The Foundation Trilogy*"(1951-52), "*Foundation's Edge*"(1982) e "*The Gods Themselves*"(1972). Isaac Asimov tomou como provável que os robôs viessem a possuir inteligência e, por isso, formulou três leis [1], para que os robôs podessem conviver com os humanos, sendo elas:

- 1ª Lei – um robô não pode fazer mal a nenhum ser humano, ou permitir que sofra algum mal;
- 2ª Lei – um robô deve obedecer às ordens dadas por um ser humano, exceto quando as ordens entram em conflito com a 1ª Lei;
- 3ª Lei – um robô deve proteger-se desde que não ponha em causa as duas leis anteriores.

Em 1993 o cientista Hans Moravec publicou um artigo, *The Age of Robots* [42], onde dizia que a primeira geração de robôs só ficaria disponível no ano 2000 e seria incapaz de aprender. A segunda geração só ficaria disponível no ano 2020, com uma inteligência comparada com a de um rato. A terceira geração de robôs só ficaria disponível no ano 2030, com uma inteligência comparável com a de um macaco. A quarta geração de robôs teria uma inteligência comparada com a do ser humano, mas esta geração só seria possível de alcançar, segundo ele, por volta dos anos 2040 ou 2050.

Em seguida serão apresentados alguns dos trabalhos que contribuiram para uma evolução da robótica:

- Em 1898, Nikola Tesla apresentou um barco controlado por ondas rádio na Exposição Eléctrica em Nova Iorque. É considerado como o pai da robótica e da rádio [45].
- Em 1961, surgiu o primeiro robô industrial o *Unimate*, desenvolvido por George Devol e Joe Engleberger [43], com a função de empilhar bocados de metal quente.
- Em 1975 foi apresentado no Japão um robô humanóide[1], o *Wabot-I*. Este tinha a capacidade de comunicar com uma pessoa em japonês e medir a distância de um objeto usando recetores externos. A partir desta data iniciou-se o desenvolvimento de robôs humanóides [54].
- Em 1994 o laboratório do MIT desenvolveu um robô humanóide chamado *Cog* [8] cuja função era aprender de forma semelhante à cognição humana.
- Em 2000 a Honda revelou o *ASIMO* [25], um robô humanóide autónomo, capaz de correr, andar, comunicar com os humanos. Este robô tem a capacidade de fazer um reconhecimento facial, ambiental e vocal.
- Em Abril de 2001 o robô *Canadarm2* foi lançado para o espaço para auxiliar os astronautas na construção e na manutenção da Estação Espacial Internacional [53].
- Em 2002 foi apresentado um aspirador robótico pela companhia iRobot, o robô *Roomba*.
- *Starfish*, um robô de 4 pernas com a capacidade de aprender a andar mesmo quando "perdia"uma perna, foi apresentado pela Universidade de Cornell em 2006 [46].
- O *I-sobot*, lançado pela empresa TOMY em 2007, é um robô humanóide que caminha, faz flexões, dança, dá murros e pontapés. Tem quatro modos de controlo: controlo remoto, de programação, de ação especial e comando de voz [7].
- Em 2011, o *Robonaut2* foi lançado para uma estação espacial a bordo do vaivém Discovery. Este foi o primeiro robô humanóide no espaço e tem a tarefa ser um assistente autónomo para ajudar os astronautas com tarefas complexas para manter a estação espacial funcional [14].

[1] Robô de aparência humana.

## 3.2 Introdução

A robótica, de uma forma simples, é a ciência que estuda os robôs, a sua construção e o seu comportemento [54]. A palavra robô é de origem checa que significa "trabalho forçado" [73, 28]. São utilizados em tarefas que constituem perigo para o ser humano: em locais com pouco luminosidade, tratamento de lixo tóxico, exploração espacial e subaquática, principalmente para buscas e resgates ou localizar minas terrestres. Alguns destes robôs têm a capacidade de aprender, através da interação com o ambiente, a realizar a melhor ação para uma dada situação para alcançar o seu objetivo. Um robô é composto por:

- Controlador – cérebro do robô e tem três fases, a percepção, o processamento e a acção. Os sensores dão informação sobre o ambiente, esta informação é processado e é realizada a acção;
- Atuadores – são motores que movimentam o robô;
- Manipuladores – são comparados com os membros dos seres humanos;
- Sensores – transmitem a informação sobre o ambiente, são comparados com os nossos sentidos;
- Por último, e mais importante a fonte de energia para o robô ganhar "vida".

Segundo DiSalvo [15], grande parte das pesquisas na robótica não se centra apenas nas tarefas industriais, mas sim em novos tipos de robôs, em outras alternativas de conceber robôs e novas maneiras de fabricá-los. Os robôs tornaram-se uma ferramenta de interesse, são cada vez mais utilizadas nas escolas secundárias e universidades. A robótica tem como objetivo a automatização de tarefas que podem ser executadas pelo homem.

Cada vez mais, os robôs precisam de possuir inteligência para terem a capacidade segurar ou agarrar objetos, navegarem sem se perderem, para observar o que está ao seu redor e aprender como chegar a um determinado local [49, 53].

Os avanços da tecnologia e da inteligência artificial, permitiu uma evolução no campo da robótica, tornando os robôs mais inteligentes e com a capacidade adquirir conhecimentos. Estes ainda estão longe de possuírem uma inteligência comparável com a do ser humano [22]. Ainda serão precisos muitos anos para que os investigadores compreendam os processos envolvidos no pensamento, e mesmo quando chegarem a alguma conclusão têm de encontrar uma maneira de implementá-los nos robôs [42].

## 3.3 Robôs Autónomos

Os robôs autónomos realizam as tarefas em ambientes desconhecidos sem ajuda humana, ou seja, conseguem atuar por eles próprios independentemente de qualquer controlador.

Para que um robô seja considerado autónomo tem que possuir certas capacidades: ter sensores que conseguem receber informações do ambiente onde se encontra, ter a capacidade de trabalhar sozinho sem intervenção humana durante um determinado período, ser capaz de deslocar-se sem intervenção humana e ser capaz de fazer uma auto-manutenção [58].

Cientistas criaram novos programas e sistemas de sensores para tornar os robôs mais inteligentes e mais percetivos [16]. Os robôs mais avançados usam visão tridimensional para "verem" melhor o ambiente que o rodeia. Duas câmaras dão ao robô uma perceção de profundidade, com programas de reconhecimento, o robô tem a capacidade de localizar e classificar vários objetos. Em seguida vão ser apresentados alguns dos robôs autónomos em ambientes reais:

- O robô *Zoe*, ilustrado na figura 3.1(a), tem a capacidade de explorar o deserto de Atacama em busca de sinais de vida, com a ajuda de sensores. É um robô autónomo que usa energia solar e pode vir a ser usado para explorar Marte [82].

(a) Robô *Zoe* (b) Robô *Nomad*

Figura 3.1: Robô *Zoe* e Robô *Nomad*

- O *Nomad*, ilustrado na Figura 3.1(b), tem como objetivo recolher amostras dos meteoritos na Antártida. Este robô explorou a remota região da Antártida, *"Elephant Moraine"*, em busca de amostras de meteoritos. Segundo Wagner [77], o Nomad encontrou e classificou corretamente no local, três meteoritos e dezenas de rochas terrestres.

- O *Tractor*, ilustrado na Figura 3.2(a), tem a capacidade de recolher informação do ambiente que o rodeia, analisando as imagens em tempo real, retirando delas as principais caraterísticas com o objetivo de plantar e colher, sozinho, e evitar a colisão com o gado. Estes dados são combinados com estimativas de alcance dos lasers, Laser Detection and Ranging *(LADAR)*, para criar mapas precisos.

(a) Robô *Tractor*


(b) Robô *SandStorm*

Figura 3.2: Robô *Tractor* e Robô *SandStorm*

- *SandStorm*, ilustrado da Figura 3.2(b), é um veículo Hummer reconfigurado com a capacidade de traçar um mapa do terreno e tomar as decisões de condução inteligente. Este consegue viajar ate 56 km/h no deserto.
- *Grace*, ilustrado na Figura 3.3, é um robô móvel que possui um enorme conjunto de sensores: microfone, vários sensores de toque, de infravermelhos, de ultra-som e um laser. Este é capaz de interagir com as pessoas, utilizando o seu microfone e o software de reconhecimento de voz [60].

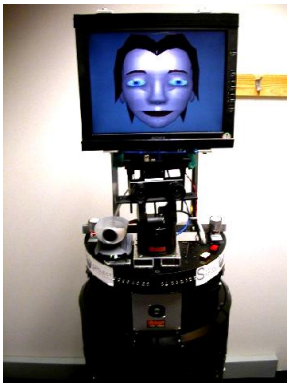


Figura 3.3: Robô *Grace*

## 3.4 Lego NXT Mindstorms

A colaboração entre a *Lego Group* e o Massachusetts Institute of Techonology *(MIT)* resultou no desenvolvimento de um robô com um bloco inteligente, conhecido por *Lego Mindstorms*, em 1988 [40, 41]. Nasceu assim, a primeira geração do *Lego Mindstorms*, mais conhecida por *Robotic Command Explorer* (RCX). Este robô trazia apenas dois motores, dois sensores de toque e um sensor de luz. O RCX foi substituído, nos finais de Julho de

2006, pela segunda geração da *Lego Mindstorms* conhecida por *NXT*. O *NXT* é mais ágil, resistente, intuitivo, utiliza as mais recentes tecnologias.

O kit básico do *NXT Mindstorms* é constituído por um bloco, três motores e quatro sensores (luz, ultra-som, som e toque), mas podem ser adquiridos outros tipos de acessórios.

Este robô evolui com o passar dos anos e atualmente já vai na terceira geração, lançada em Agosto de 2009 [41].

### 3.4.1 *Kit* básico do *NXT*

O *kit* básico do *NXT* contém os seguintes componentes:

- O bloco *NXT*, ilustrado na Figura 3.4(a), conhecido por *smart brick*, é o "cérebro" do robô. É um bloco controlado por computador que dá vida ao robô, efetuando diferentes operações [18]. Este bloco contém quatro entradas na parte inferior e três na parte superior e uma entrada *Universal Serial Bus* (USB).


(a) Bloco do *Nxt Mindstorms*

(b) Motor do *Nxt Mindstorms*

Figura 3.4: Bloco do *Nxt Mindstorms* e motor do *Nxt Mindstorms*

- Os motores, ilustrados na Figura 3.4(b), permitem ao robô movimentar-se pelo ambiente. Têm a capacidade de ajustar a velocidade e calculam com precisão as voltas ou graus que o motor efetuou, podem chegar às 170 rotações por minuto.

- O kit básico, da versão 1.1, inclui quatro sensores, vai ser apresentado outro sensor que pertence à versão 2.0:

  – O sensor de luz, ilustrado na Figura 3.5(a), permite ao robô distinguir a luminosidade ambiente e através de um *Light Emitting Diode* (LED) vermelho incorporado, consegue detetar a intensidade das cores. As leituras são indicadas em percentagem (%), e variam de 0 a 100% .


(a) Sensor de luz (b) Sensor de cor

Figura 3.5: Sensor de luz e sensor de cor

– O sensor de cor, ilustrado na Figura 3.5(b), pertence à versão 2.0 e tem a capacidade de distinguir até seis cores através de um LED *Red Green Blue* (RGB), detecta o nível da luminosidade ambiente, também consegue detetar a intensidade das cores e funciona como uma lâmpada que emite luz vermelha, verde ou azul.
– O sensor de toque, ilustrado na Figura 3.6(b), tem a capacidade de indicar se este está a ser pressionado ou não, os valores podem ser 0, caso o sensor não estiver pressionado, ou 1, caso o sensor esteja a ser pressionado.

(a) Sensor ultra-sónico (b) Sensor de toque

Figura 3.6: Sensor ultra-sónico e sensor de toque

– O sensor de ultra-som, ilustrado na Figura 3.6(a), tem a capacidade de medir a distância a que o robô se encontra de um obstáculo. Este sensor mede distâncias desde 0 a 255 cm com precisão de ± 3 cm. O sensor de ultra-som utiliza os mesmos princípios científicos dos morcegos, a ecolocalização que mede a distâncias e calcula o tempo que o som demora a atingir o objecto e regressar
– O sensor de som, ilustrado na Figura 3.7, tem a capacidade de determinar se o nível do som detetado é alto ou baixo mede até 90 decibéis. Como os diferentes níveis de som são complicados de gerir, as leituras são indicadas em percentagem (%).
  * 4-5%, ambiente silencioso;
  * 5-10%, som a alguma distância;
  * 10-30%, som normal perto do sensor;
  * 30-100%, som alto ou ambiente muito ruidoso.

Figura 3.7: Sensor ultra-sónico e sensor de som

### 3.4.2 NXT Software

O bloco *NXT* pode ser programado com recurso a diversas linguagens:

***NXT-G.*** Linguagem de programação disponibilizada pelo *NXT Mindstorms*, é recomendável para quem não tem grandes conhecimentos na área da programação. O seu interface é de fácil compreensão e, por sua vez, bastante simples de usar [29].

***Not eXactly C.*** Conhecida por *NXC*, é uma linguagem *open-source* de alto nível semelhante à linguagem *C*. Esta linguagem é destinada para utilizadores com mais experiência e que queiram dar uma nova dinâmica ao robô [].

***ROBOTC.*** Desenvolvida pela *Carnegie Mellon Robotic's Academy*, é uma linguagem de programação baseada em *C* para o *Lego NXT Mindstorms*. Executa um *firmware* otimizado que permite ao *NXT* executar programas mais rapidamente, comprimindo-os para que o bloco consiga armazenar uma maior quantidade de programas.

***leJOS NXJ.*** É uma linguagem *open-source* de alto nível que usa *Java Virtual Machine* e fornece uma poderosa *Application Programming Interface* (API), assim como as ferramentas necessárias para descarregar o código para o bloco NXT. Esta linguagem é destinada a programadores intermédios e experientes, e fornece bibliotecas extensivas que suportam várias funções de alto nível, tais como, a navegação e comportamento baseados em robótica.

***MATLAB.*** Linguagem programação de alto nível para cálculos numéricos e aquisição de dados e análise. Pode ser usada para controlar os robôs *NXT* através de bluetooth ou por USB.

**PyNXC.** É um projeto que converte o código em *python* para a linguagem *NXC*, *Not eXactly C*, para fazer o download do programa para o bloco *NXT*.

***C# com Microsoft Robotics Developer Studio.*** As ferramentas livres, *Visual Studio Express* e *Robotics Developer Studio*, permitem ao utilizador programar na linguagem *C#*.

Existem ainda outras linguagens com suporte ao bloco NXT: o *BricxCC, a Next Byte Codes, o Robolab, o NXTGCC, o nxtOSEK, o ICON, a Lua, Ada, o URBI, o FLL NXT Navigation, o ruby-nxt, o Robotics.NXT, o LibNXT e o Simulink.*

## 3.5 Trabalho relacionado

Existem poucos trabalhos relacionados com a tarefa realizada nesta dissertação. No trabalho de Vamplew [74] é construído um robô com base no Lego RCX é utilizado o algoritmo SARSA. O robô utiliza dois sensores de infravermelhos, postos lado a lado na parte frontal, sendo o objetivo medir o número de iterações que o robô necessita para aprender a seguir o percurso sem o perder. Neste trabalho, o robô demora em média 20 minutos a aprender a seguir a linha com a melhor combinação($\alpha = 0.1$ e o $\epsilon = 0.9$);

Em [61] é usado um robô diferente do *Lego NXT Mindstorms*, o *Micom Car* [84], com o algoritmo Q-learning. O *Micom Car* pode usar até 8 sensores infravermelhos enquanto o *NXT* só pode usar no máximo 4 sensores. As experiências apresentadas, consistem em testar o robô em vários percursos com o objetivo de verificar o desempenho do robô, sendo registado o tempo que o robô leva a completar o percurso e a média do número de voltas sem perder a linha. Os melhores resultados foram obtidos com o valor de 13.6 segundos dado numa volta e dá 8.4 voltas sem perder a linha com a melhor combinação ($\alpha = 0.8$ e o $\epsilon = 0.999$);

# Capítulo 4

# Aprender a seguir uma linha

Para estudar a implementação do algoritmo *Q-learning* no Lego *NXT Mindstorms* foi construído um veículo capaz de seguir um percurso marcado numa superfície. Foram realizados três conjuntos de experiências, para verificar o desempenho do robô: variação dos valores das variáveis, do valor das recompensas e a utilização de vários percursos.

Este capítulo introduz o agente, descreve a configuração experimental e apresenta e discute os resultados obtidos.

## 4.1 O Veículo

Para avaliar as técnicas de aprendizagem por reforço foi construído o veículo autónomo ilustrado na Figura 4.1. Este é composto pelo bloco *NXT*, dois motores responsáveis pelo movimento e dois sensores de infravermelhos colocados na parte frontal para indicar a sua posição relativamente ao percurso marcado. Utilizou-se dois sensores para facilitar a aprendizagem do robô.

Figura 4.1: Veículo autónomo construído

O bloco *NXT* contém o programa responsável pelo comportamento do veículo: recebe a informação analisada pelos sensores, processa essa informação e dá indicações de atuação aos motores.

Para programar o bloco *NXT* utilizou-se a linguagem *lejOS NXJ* [40], descarregando o código através de ligação USB.

Embora com a mesma finalidade, os sensores infravermelhos utilizados são distintos, pois na altura só estavam disponíveis estes sensores: um tem a capacidade de fazer uma leitura da luz refletida e verificar luminosidade do ambiente[1] e o outro, para além destas capacidades, é capaz de distinguir as cores do objeto refletido[2]. Estes sensores estão ilustrados na Figura 3.5(a) e na Figura 3.5(b).

A utilização dos dois sensores infravermelhos colocam o veículo num de 4 estados possíveis:

- `F`: os dois sensores fora da linha
- `E`: o sensor esquerdo sobre a linha e o direito fora da linha
- `D`: o sensor direito sobre a linha e o esquerdo fora da linha
- `S`: os dois sensores sobre a linha

O movimento dos dois motores permite realizar as quatro ações básicas: andar para a frente, andar para trás, virar à direita e virar à esquerda. Estas ações são conseguidas através da imposição de diferentes velocidades de rotação dos motores. Foram testadas diversas velocidades, tendo sido escolhida a velocidade 6 graus/segundo porque foi aquela que demonstrou ser a velocidade indicada para que o robô conseguisse aprender, a seguir a linha, mais rapidamente.

Para seguir corretamente a linha, foi implementada uma função de calibração de valores: basicamente esta função, através dos sensores infravermelhos, faz várias leituras dos valores da luz refletida quando sobre a linha e fora da linha, utilizando o valor médio como limiar.

[1] Sensor *NXT Mindstorms* da versão 1.1.
[2] Sensor NXT Mindstorms versão 2.0.

## 4.2 Configuração experimental

Com o veículo construído escolheu-se o percurso para o estudo, uma linha reta com 100 cm de comprimento e 5 de largura, e fizeram-se algumas experiências iniciais para escolher os valores de recompensa para cada estado, tendo sido utilizados os seguintes:

- estado `F`: $r_t = -15$. Com os dois sensores fora da linha a recompensa deve ser negativa;
- estados `E` e `D`: $r_t = -5$. Com um dos sensores fora da linha a recompensa deve ser maior mas também negativa;
- estado `S`: $r_t = 10$. Com os dois sensores sobre a linha a recompensa é positiva.

Definiram-se, depois, diversos conjuntos de experiências para estudar a influência dos diferentes elementos da aprendizagem por reforço no seu desempenho.

## 4.3 Experiências

O primeiro conjunto de experiências visa estudar a influência das variáveis do algoritmo *Q-learning* com o método de pesquisa $\epsilon$-*Greedy*, nomeadamente a taxa de aprendizagem ($\alpha$), o fator de desconto ($\gamma$) e a taxa de exploração ($\epsilon$). O segundo conjunto pretende estudar a influência da variação da taxa de aprendizagem ao longo do processo de aprendizagem e o terceiro conjunto compara os métodos de pesquisa $\epsilon$-*Greedy* e *Softmax*.

Foram ainda feitas experiências para estudar a influência da função de recompensa e do ambiente no processo de aprendizagem.

Para cada cenário foram realizadas 20 experiências, sendo contabilizado o número de iterações necessárias para que o veículo complete o percurso sem perder a linha. A próxima seção descreve as experiências e apresentam os resultados obtidos.

### 4.3.1 Influência dos valores das variáveis

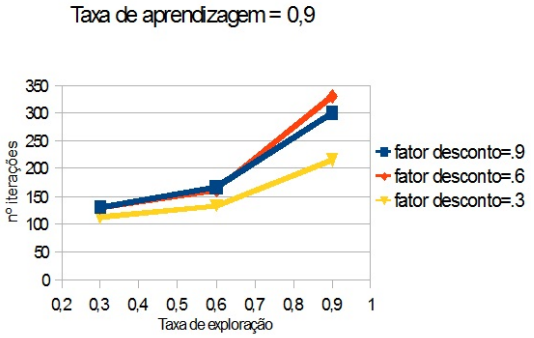
Para estudar a influência das diferentes variáveis do algoritmo *Q-learning* implementado com o método de pesquisa $\epsilon$-*Greedy* realizaram-se experiências com todas as combinações dos valores 0.3, 0.6 e 0.9 para a taxa de aprendizagem ($\alpha$), fator de desconto ($\gamma$) e taxa de exploração ($\epsilon$).

Uma vez que as taxas de aprendizagem e de exploração devem diminuir ao longo do processo de aprendizagem, aos seus valores iniciais são decrementados 0.1 a cada 50 iterações.

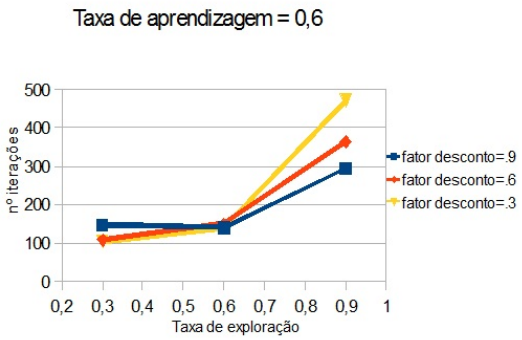
A Tabela 4.1 mostra os resultados obtidos (média e desvio padrão do nº de número de iterações do algoritmo até o veículo completar o percurso sem perder a linha).

| $\alpha$ | $\gamma$ | $\epsilon$ | Iterações |
|---|---|---|---|
| 0.9 | 0.9 | 0.9 | 300.4 ± 24.3 |
| 0.9 | 0.9 | 0.6 | 166.6 ± 8.5 |
| 0.9 | 0.9 | 0.3 | 130.5 ± 31.1 |
| 0.9 | 0.6 | 0.9 | 329.9 ± 48.3 |
| 0.9 | 0.6 | 0.6 | 162.1 ± 13.7 |
| 0.9 | 0.6 | 0.3 | 128.9 ± 13.5 |
| 0.9 | 0.3 | 0.9 | 215.1 ± 20.9 |
| 0.9 | 0.3 | 0.6 | 133.1 ± 15.9 |
| 0.9 | 0.3 | 0.3 | 112.6 ± 21.4 |
| 0.6 | 0.9 | 0.9 | 294.1 ± 28.7 |
| 0.6 | 0.9 | 0.6 | 139.8 ± 12.9 |
| 0.6 | 0.9 | 0.3 | 146.8 ± 36.6 |
| 0.6 | 0.6 | 0.9 | 364.2 ± 48.4 |
| 0.6 | 0.6 | 0.6 | 150.1 ± 14.4 |
| 0.6 | 0.6 | 0.3 | 109.0 ± 15.7 |
| 0.6 | 0.3 | 0.9 | 471.5 ± 25.2 |
| 0.6 | 0.3 | 0.6 | 139.1 ± 16.2 |
| 0.6 | 0.3 | 0.3 | 103.6 ± 10.2 |
| 0.3 | 0.9 | 0.9 | 589.4 ± 10.9 |
| 0.3 | 0.9 | 0.6 | 126.1 ± 11.4 |
| 0.3 | 0.9 | 0.3 | 156.8 ± 19.9 |
| 0.3 | 0.6 | 0.9 | 339.9 ± 38.3 |
| 0.3 | 0.6 | 0.6 | 119.7 ± 16.7 |
| 0.3 | 0.6 | 0.3 | **91.6** ± 11.0 |
| 0.3 | 0.3 | 0.9 | 304.8 ± 11.9 |
| 0.3 | 0.3 | 0.6 | 120.4 ± 12.5 |
| 0.3 | 0.3 | 0.3 | 110.6 ± 11.1 |

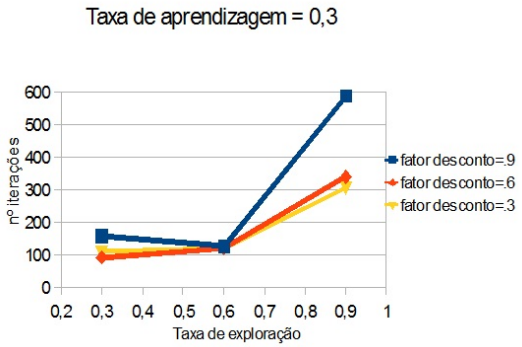
Tabela 4.1: Nº médio de iterações para o método de pesquisa $\epsilon$-*Greedy*.




(a) Taxa de aprendizagem é igual a 0.9




(b) Taxa de aprendizagem é igual a 0.6




(c) Taxa de aprendizagem é igual a 0.3

Figura 4.2: Gráficos do nº médio de iterações para o método de pesquisa $\epsilon$-*Greedy*

Pela observação da tabela e dos gráficos, é possível constatar que o veículo aprendeu a seguir a linha mais rapidamente com uma taxa de aprendizagem inicial $\alpha = 0.3$, um fator de desconto $\gamma = 0.6$ e uma taxa de exploração inicial de $\epsilon = 0.3$. Por outro lado, a configuração com $\alpha = 0.3$, $\gamma = 0.9$ e $\epsilon = 0.9$ apresenta os piores resultados.

Através dos Gráficos 4.2, é possível constatar que o comportamento do robô apresenta piores resultados quando a taxa de exploração aumenta. Também é possível constatar que a taxa de exploração para valores iguais ou inferiores a $\epsilon = 0.6$, o número de iterações é muito semelhante e o comportamento do robô não sofre alterações significativas. Através do Gráfico 4.2(c), é possível verificar que a taxa de aprendizagem para valores iguais a $\alpha = 0.3$ o robô tanto pode ter o melhor desempenho como o pior, dependendo do valor da taxa de exploração. Também possível constatar, através dos Gráficos 4.2(a) e 4.2(b) ,que a taxa de aprendizagem e o fator de desconto para valores superiores ou iguais as $\alpha = 0.6$ e $\gamma = 0.6$ o número de iterações é muito semelhante.

Através dos valores apresentados na tabela é possível confirmar que valores altos da taxa de exploração ($\epsilon = 0.9$) produzem um comportamento errático do veículo com valores médios do nº de iterações superiores a 300 (exceto quando o $\alpha = 0.9$ e $\gamma = 0.3$). Este resultado

era expetável já que com este valor 90% das vezes é escolhida uma acção aleatória que não considera o valor da função ação-valor (equação 2.17) para o estado em que o veículo se encontra.

Conforme esperado, também é possível constatar que o número de iterações descresce com a diminuição da taxa de exploração.

Para valores baixos da taxa de exploração ($\epsilon = 0.3$) o nº de iterações mínimo é obtido com o fator de desconto $\gamma = 0.6$, mas o aumento da taxa de exploração desfaz esta tendência.

### 4.3.2 Influência do método de pesquisa

Para estudar a influência do método de pesquisa no processo de aprendizagem testou-se o método *Softmax* com a configuração ganhadora do método $\epsilon$-*greedy*: $\alpha = 0.3$, $\gamma = 0.6$ e $\epsilon = 0.3$ e com valores de temperatura de 0.3, 0.6 e 0.9.

A Tabela 4.2 mostra o número médio de iterações do veículo até conseguir completar ao percurso sem perder a linha e o correspondente desvio padrão.

| $\alpha$ | $\gamma$ | $T$ | Iterações |
|---|---|---|---|
| 0.3 | 0.6 | 0.9 | 142.7 ± 9.6 |
| 0.3 | 0.6 | 0.6 | 115.4 ± 13.2 |
| 0.3 | 0.6 | 0.3 | 83.5 ±13.3 |

Tabela 4.2: Nº médio de iterações com o método de pesquisa *Softmax*.

Através da observação da tabela, é possível constatar que o veículo tem um melhor desempenho para o valor de temperatura $T = 0.3$; por outro lado, a configuração com a temperatura $T = 0.9$ apresenta os piores resultados. Assim, é possível concluir que valores altos da temperatura ($T = 0.9$) produzem um comportamento menos desejável do veículo. Este resultado era expetável já que quanto maior a temperatura, todas as ações têm probabilidades (quase) iguais de escolha.

Em relação aos métodos de pesquisa $\epsilon$-*Greedy* e *Softmax* (Tabela 4.1 e Tabela 4.2, respectivamente), pode-se constatar que o método *Softmax* necessita de menos iterações de aprendizagem que o método $\epsilon$-*Greedy* e que essa diferença é maior para valores superiores da taxa de exploração e equivalente temperatura.

### 4.3.3 Influência dos intervalos de variação das variáveis

Uma vez que a taxa de aprendizagem $\alpha$ e a temperatura $T$ devem ir diminuindo ao longo do processo de aprendizagem experimentou-se diminuir os seus valores em 0.1 unidades a diferentes intervalos: 50, 40, 30, 20 e 10 utilizando os valores iniciais destas variáveis de 0.9, 0.6 e 0.3. A Tabela 4.3 apresenta os resultados obtidos.

| intervalo | $\alpha$ | $\gamma$ | $\epsilon/T$ | $\epsilon$-*Greedy* | *Softmax* |
|---|---|---|---|---|---|
| 50 | 0.3 | 0.6 | 0.9 | 339.8 ± 38.3 | 142.7 ± 9.6 |
| 50 | 0.3 | 0.6 | 0.6 | 119.7 ± 16.7 | 115.4 ± 13.2 |
| 50 | 0.3 | 0.6 | 0.3 | 91.6 ± 11.0 | 83.5 ± 13.3 |
| 40 | 0.3 | 0.6 | 0.9 | 275.9 ± 13.9 | 118.6 ± 5.1 |
| 40 | 0.3 | 0.6 | 0.6 | 117.1 ± 5.1 | 104.1 ± 6.7 |
| 40 | 0.3 | 0.6 | 0.3 | 83.7 ± 8.8 | 72.7 ± 7.6 |
| 30 | 0.3 | 0.6 | 0.9 | 251.2 ± 6.8 | 100.8 ± 9.8 |
| 30 | 0.3 | 0.6 | 0.6 | 96.6 ± 6.9 | 81.9 ± 7.3 |
| 30 | 0.3 | 0.6 | 0.3 | 66.3 ± 6.6 | 59.9 ± 6.2 |
| 20 | 0.3 | 0.6 | 0.9 | 201.3 ± 11.6 | 77.4 ± 5.5 |
| 20 | 0.3 | 0.6 | 0.6 | 71.8 ± 6.5 | 46.5 ± 7.2 |
| 20 | 0.3 | 0.6 | 0.3 | 44.4 ± 5.9 | 43.2 ± 5.6 |
| 10 | 0.3 | 0.6 | 0.9 | 178.6 ± 4.8 | 49.4 ± 6.7 |
| 10 | 0.3 | 0.6 | 0.6 | 49.9 ± 5.8 | 34.9 ± 3.5 |
| 10 | 0.3 | 0.6 | 0.3 | 31.9 ± 6.7 | 28.6 ± 4.7 |

Tabela 4.3: Nº médio de iterações para diferentes intervalos de atualização de $\epsilon$ e $T$.

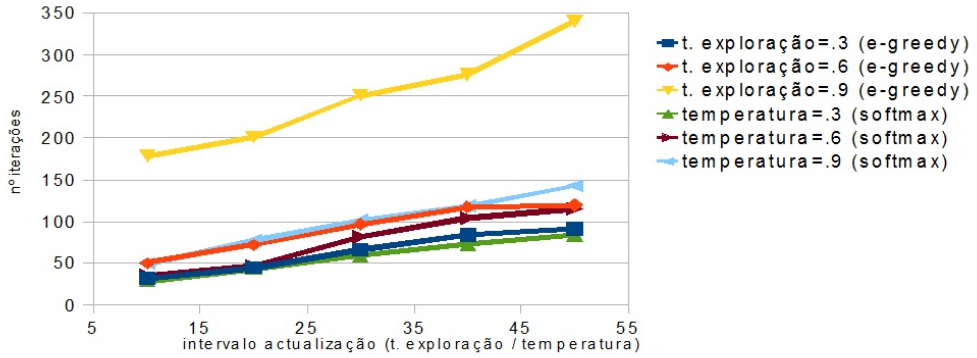



Figura 4.3: Gráfico do nº médio de iterações para diferentes intervalos de atualização de $\epsilon$ e $T$.

Pela observação da tabela e do gráfico, e como seria de esperar, é possível constatar que o número de iterações decresce com a diminuição do intervalo de atualização das taxas de aprendizagem e de exploração ($\epsilon$-*Greedy*) e temperatura (*Softmax*).

Esta tabela e o gráfico replica o mesmo comportamento referido na sub-seção anterior no que respeita à comparação entre os métodos de pesquisa:

- a aprendizagem é mais rápida à medida que a taxa de exploração e a tempertura diminuem;
- a aprendizagem é mais rápida com o *Softmax*;
- esta diferença é mais evidente para valores maiores da taxa de exploração e temperatura, como é possível verificar no Gráfico 4.3, que o método de pesquisa $\epsilon$-*Greedy* apresenta uma maior diferença no número de iterações em relação ao método de pesquisa *Softmax* quando o valor da taxa de exploração é $\epsilon = 0.9$.

### 4.3.4 Influência da função de recompensa

Este conjunto de experiências foi realizado para determinar a função de recompensa com melhores resultados no processo de aprendizagem. Estudaram-se os métodos de pesquisa $\epsilon$-*Greedy* e *Softmax* com os melhores valores das variáveis obtidos nas experiências anteriores:

- $\alpha = 0.3$, $\gamma = 0.6$, $\epsilon = 0.3$, $T = 0.3$;
- diminuição de $\alpha$, $\epsilon$ e $T$ em 0.1 unidades a cada 10 iterações.

A Tabela 4.4 ilustra as três combinações de recompensas exploradas e a Tabela 4.5 mostra o número médio de iterações necessárias para cada uma dessas funções de recompensa.

| função | `S` | `E` e `D` | `F` |
|---|---|---|---|
| $f_1$ | 10 | -5 | -15 |
| $f_2$ | 20 | 5 | -10 |
| $f_3$ | 20 | 10 | -10 |

Tabela 4.4: Funções de recompensa.

| função | $\epsilon$-*Greedy* | *Softmax* |
|---|---|---|
| $f_1$ | $31.9 \pm 6.7$ | $28.6 \pm 4.7$ |
| $f_2$ | $58.8 \pm 7.6$ | $50.9 \pm 5.2$ |
| $f_3$ | $68.9 \pm 7.7$ | $65.1 \pm 4.2$ |

Tabela 4.5: Nº médio de iterações para cada função de recompensa.

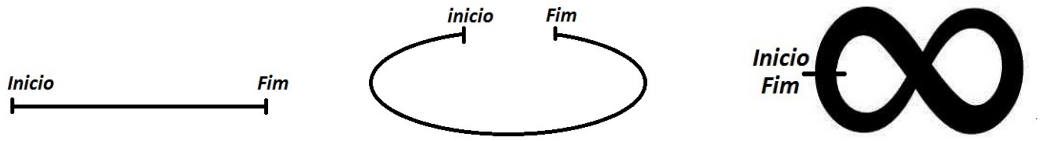
Pela observação da tabela, é possível constatar que o veículo tem um melhor desempenho com função $f_1$; por outro lado, a função $f_3$ apresenta os piores resultados para ambos os métodos de pesquisa.

Mais uma vez, o método *Softmax* apresenta melhores resultados que o método $\epsilon$-*greedy* embora as diferenças não sejam significativas.

### 4.3.5 Influência do ambiente

O ambiente é composto por uma superfície onde é desenhado o percurso que o veículo deve seguir.

Este conjunto de experiências foi realizado determinar o comportamento do veículo nos três percursos ilustrados na Figura 4.4.




(a) Percurso 1 (b) Percurso 2 (c) Percurso 3

Figura 4.4: Percursos testados

O percurso 1, ilustrado na Figura 4.4(a), tem 100 cm de comprimento e de 5 de largura, como já foi referido anteriormente. O percurso 2, ilustrado na Figura 4.4(b), tem 150 cm de comprimento e 5 de largura. O percurso 3, ilustrado na Figura 4.4(c), tem 200 cm de comprimento e 5 de largura.

Para cada percurso foram testadas as três funções de recompensa para os métodos de pesquisa $\epsilon$-*Greedy* e *Softmax* e os melhores valores para as variáveis obtidos nas experiências da secção 4.3.3.

A Tabela 4.6 mostra o número médio de iterações para cada método de pesquisa, função de recompensa e percurso.

| percurso | função | $\epsilon$-*Greedy* | *Softmax* |
|---|---|---|---|
| P1 | $f_1$ | 31.9 ± 6.7 | 28.6 ± 4.7 |
| | $f_2$ | 58.8 ± 7.6 | 50.9 ± 5.2 |
| | $f_3$ | 68.9 ± 7.7 | 65.1 ± 4.2 |
| P2 | $f_1$ | 53.2 ± 4.8 | 41.6 ± 5.1 |
| | $f_2$ | 99.4 ± 9.3 | 80.6 ± 6.2 |
| | $f_3$ | 114.9± 9.1 | 100.8± 8.4 |
| P3 | $f_1$ | 116.8 ± 8.2 | 115.2 ± 7.2 |
| | $f_2$ | 144.2 ± 6.1 | 134.8 ± 8.8 |
| | $f_3$ | 164.1 ± 8.2 | 156.2 ± 7.8 |

Tabela 4.6: Nº médio de iterações para cada percurso.

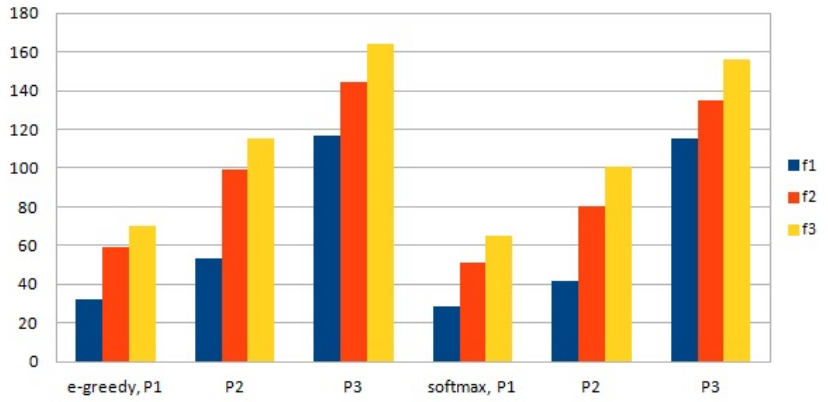



Figura 4.5: Gráfico do nº médio de iterações para cada percurso.

Pela observação da tabela e do gráfico, é possível constatar que o veículo tem um melhor desempenho para o percurso 1 e função de recompensa $f_1$. Este comportamento era o esperado, pois trata-se de um percurso simples sem interseções e curvas.

No percurso 2 existe um aumento da média das iterações de todas as combinações dos valores da recompensa, pois o percurso tem duas curvas o que dificulta a aprendizagem do robô. Este percurso é maior que o percurso 1.

O percurso 3 apresenta os piores resultados pois este percurso é composto por curvas e uma interseção. Além disso, o tamanho do percurso é maior em relação aos percursos 1 e 2.

Nesta tabela reconfirma-se que o método de pesquisa *Softmax* mantém valores mais baixos do nº de iterações relativamente ao método $\epsilon$-*Greedy*.

# Capítulo 5

# Conclusões e trabalho futuro

Neste capítulo final são apresentadas as conclusões deste trabalho e descritas possíveis extensões e melhorias como trabalho futuro.

## 5.1 Conclusão

Este trabalho estuda o algoritmo de aprendizagem por reforço *Q-learning*, e a sua interação com os métodos de pesquisa $\epsilon$-*Greedy* e *Softmax* aplicados a um veículo autónomo construído com o kit educacional *NXT Mindstorms* cujo objetivo é seguir um percurso. As diversas experiências realizadas demonstram que o robô consegue aprender a tarefa em poucas iterações do algoritmo (inferior a 100 iterações).

Foi analisada a influência de dois métodos de pesquisa – $\epsilon$-*Greedy* e *Softmax* – no comportamento do robô. Foi analisada a interação dos diversos parâmetros do algoritmo e métodos de pesquisa: a taxa de aprendizagem $\alpha$, o fator de desconto $\gamma$ (parâmetros do algoritmo *Q-learning*), a taxa de exploração $\epsilon$ (parâmetro do método de pesquisa $\epsilon$-*Greedy*) e a temperatura $T$ (parâmetro do método de pesquisa *Softmax*). Analisou-se também a influência da variação ao longo da aprendizagem das variáveis dos métodos de pesquisa e a influência da função de recompensa e da complexidade dos percursos no processo de aprendizagem.

Pelas experiências realizadas é possível concluir que o melhor desempenho do robô é obtido com a configuração $\{\alpha = .3, \gamma = .6, \epsilon = .3\}$ (taxa de aprendizagem e taxa de exploração baixas e fator de desconto médio) para o método de pesquisa $\epsilon$-*Greedy* e $\{\alpha = .3, \gamma = .6, T = .3\}$ (taxa de aprendizagem e temperatura baixas e fator de desconto médio) para o método de pesquisa *Softmax*.

É possível concluir, que o robô demonstra um melhor desempenho quando implementado com método de pesquisa *Softmax*, embora a diferença da média de iterações entre os dois métodos seja mínima, exceto para os valores da taxa de exploração $\epsilon = .9$, onde o método de pesquisa $\epsilon$-*Greedy*, apresenta os piores resultados. Este resultado seria de esperar uma vez que em 90% das vezes é escolhida uma ação aleatória. As experiências realizadas também confirmam que valores altos da taxa de exploração dão origem a piores desempenhos do robô, já que o seu comportamento torna-se errático.

Também é possível verificar que a taxa de exploração para valores iguais ou inferiores a $\epsilon = 0.6$ e taxa de aprendizagem e o fator de desconto para valores iguais ou superiores a $\alpha = 0.6$ e $\gamma = 0.6$ o número de iterações é equivalente e o desempenho do robô não sofre alterações significativas.

Confirma-se também que a aprendizagem é mais rápida quando a taxa de exploração e a temperatura diminuem.

É possível confirmar que o robô tem melhor desempenho com a função de recompensa $f_1$ quando o robô tem os dois sensores fora da linha (estado `F`) $r_t = -15$, quando um sensor está fora da linha (estados `E` e `D`) $r_t = -5$ e quando os dois sensores estão sobre a linha (estado `S`) $r_t = 10$.

Em relação à influência do ambiente, o robô apresenta melhores resultados quando está no percurso 1. Como esperado existe um aumento da média das iterações quando o robô é testado nos percursos 2 e 3, pois estes são maiores que o percurso 1 e representam uma dificuldade maior já que existem curvas e interseções. O percurso 3, aquele cujo número de iterações necessárias para a aprendizagem é maior possui uma interseção para além da sua dimensão ser maior que as dos outros percursos.

Finalmente pode-se concluir que para todas as experiências realizadas o método de aprendizagem que apresenta melhores resultados é o *Softmax*, pois este não escolhe ações aleatórias.

O maior desafio encontrado na realização desta dissertação foi conseguir que o robô aprendesse a seguir a linha, pois inicialmente a velocidade dos motores do *NXT* influenciou muito o tempo de aprendizagem, já que o robô perdia muitas vezes a linha. Após várias tentativas e várias alterações no valor da velocidade este problema foi ultrapassado.

Os objetivos deste trabalho, que consistiam em construir um robô implementado com um algoritmo de aprendizagem por reforço e analisar o seu comportamento verificando a influência das diversas variáveis do problema, foram alcançados.

## 5.2 Trabalho Futuro

Existem várias vertentes que podem ser seguidas como trabalho futuro. Relativamente às experiências poder-se-ia:

- comparar outro algoritmo de aprendizagem por reforço como por exemplo o SARSA com a finalidade comparar o seu comportamento e verificar qual o mais adequado para a tarefa estudada;
- fazer um percurso com diversos tipos de linha, ou seja, utilizar linhas com larguras e cores diferentes com a finalidade de determinar se o robô seria capaz de seguir este percurso;

Outra linha de trabalho futuro, seria a alteração do robô tornando-o mais robusto à tarefa, quer adicionando sensores infravermelhos ao robô quer adicionando um giroscópio para não permitir ao robô dar uma volta de 180º, pois este sensor dá a informação ao robô da direção está a seguir.

Indo ainda mais longe, a construção de outro robô com uma tarefa diferente para aprender o que permitiria aumentar a abrangência do estudo efetuado nesta dissertação. Tarefas possíveis seriam seguir uma fonte de luz ou som;

Ainda outra linha de investigação seria a utilização de outros kits robóticos, como por exemplo, o *khepera* [79] ou o *3pi* [69] para estudar e comparar os seus comportamentos.

# Bibliografia


[1] I. Asimov. *Notes for a memoir: on Isaac Asimov, life, and writing.* Prometheus Books, 2006.

[2] R. Bellman. Dynamic programming. In *Princeton University Press, Princeton, NJ.*, 1957.

[3] R. Bellman and S. E Dreyfus. Functional approximations and dynamic programming. In *Math Tables and Other Aides to Computation*, volume 13, pages 247–251, 1959.

[4] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, NJ., 1987.

[5] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena, Belmont, MA., 1995.

[6] Jesper Blynel. Reinforcement learning on real robots. Master's thesis, University of Aarhus, 2000.

[7] David A Bradley. Mechatronics - more questions than answers. *Mechatronics*, 20(8):827–841, 2010.

[8] Rodney A. Brooks, Cynthia Breazeal, Matthew Marjanovic, Brian Scassellati, and Matthew M. Williamson. The Cog Project: Building a Humanoid Robot. *Lecture Notes in Computer Science*, 1562:52–87, 1999.

[9] Eduardo Camponogara and Maurício Rangel Guimarães Serra. Aprendizagem por reforço: Uma primeira introdução, 2005.

[10] Jakob Carlström and Ernst Nordström. Reinforcement learning for control of self-similar call traffic in broadband networks. pages 571–580, 1999.

[11] Melanie Coggan. Exploration and exploitation in reinforcement learning. Technical report, McGill University, 2004.

[12] Robert Crites and Andrew Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*, pages 1017–1023. MIT Press, 1996.

[13] Peter Dayan and Christopher Watkins. Reinforcement learning. 2001.

[14] Myron A. Diftler, Joshua Mehling, Muhammad E. Abdallah, Nicolaus A. Radford, Lyndon B. Bridgwater, Adam M. Sanders, Roger Scott Askew, D. Marty Linn, John D. Yamokoski, Frank Permenter, Brian K. Hargrave, Robert Platt, Robert T. Savely, and Robert O. Ambrose. Robonaut 2 - the first humanoid robot in space. In *ICRA*, pages 2178–2183, 2011.

[15] Carl F. DiSalvo, Francine Gemperle, Jodi Forlizzi, and Sara Kiesler. All robots are not created equal: the design and perception of humanoid robot heads. In *DIS '02: Proceedings of the 4th conference on Designing interactive systems*, pages 321–326, New York, NY, USA, 2002. ACM.

[16] Erick Dupuis, Régent L'Archevêque, Pierre Allard, Ioannis Rekleitis, and Eric Martin. *Intelligent Space Robotics*, chapter A Framework for Autonomous Space Robotics Operations. TSI Press, 2006.

[17] B. G. Farley and W. A. Clark. Simulation of self-organizing systems by digital computer. In *IRE Transactions on Information Theory*, volume 4, pages 76–84, 1954.

[18] Octávia Raquel Gomes Figueira. Droide m.l.p potencializando a plataforma, 2008.

[19] Kary Främling. Reinforcement learning in a noisy environment: Light-seeking robot. *WSEAS Transactions on Systems*, 3(2):714–719, 2004.

[20] Er Gloye, Fabian Wiesel, Cüneyt Göktekin, Anna Egorova, Oliver Tenchio, and Raúl Rojas. Learning to drive and simulate autonomous robots with reinforcement learning. http://robocup.mi.fu-berlin.de/buch/learning3.pdf, 2004.

[21] Viviane Margarida Gomes. Controle inteligente de tempo livre em tutoria multissessão, 2009.

[22] Ricardo Ribeiro Gudwin. Novas fronteiras na inteligência artificial e na robótica. 2005.

[23] Mance E. Harmon and Stephanie S. Harmon. Reinforcement learning: A tutorial, 1996.

[24] Simon Haykin. *Neural Networks: A Comprehensive Foundation Second Edition*. Prentice Hall, 1998.

[25] Masato Hirose and Tooru Takenaka. Development of Humanoid Robot ASIMO. *Honda R&D Tech Rev*, 13(1):1–6, 2001.

[26] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. Convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

[27] L.P. Kaelbling, M.L. Littman, and Andrew Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[28] Satyam Kalan, Sanket Chauhan, Rafael F Coelho, Marcelo A Orvieto, Ignacio R Camacho, Kenneth J Palmer, and Vipul R Patel. History of robotic surgery. *Journal of Robotic Surgery*, 4(3):141–147, 2010.

[29] James Floyd Kelly. *LEGO MINDSTORMS NXT-G Programming Guide*. Apress, Berkely, CA, USA, 2nd edition, 2010.

[30] A. H. Klopf. *Brain function and adaptive systems A heterostatic theory*. Data Sciences Laboratory, Air Force Cambridge Research Laboratories, Air Force Systems Command, United States Air Force,, 1972.

[31] A. H. Klopf. A comparison of natural and artificial intelligence. *SIGART Newsletter*, 53:11–13, 1975.

[32] A. H. Klopf. The hedonistic neuron: A theory of memory, learning, and intelligence. In *Hemisphere, Washington, D.C*, volume 10, pages 390–398, 1982.

[33] Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, ECML PKDD '08, pages 656–671, Berlin, Heidelberg, 2008. Springer-Verlag.

[34] S. Mahadevan, J. Connell, C. Sammut, R. Sutton, and Temporal Phd. Automatic programming of behavior-based robots using reinforcement learning, 1991.

[35] Maja J. Mataric, Gaurav S. Sukhatme, and Esben H. Ostergaard. Multi-robot task allocation in uncertain environments, 2003.

[36] J. M. Mendel. Applications of artificial intelligence techniques to a spacecraft control problem. In *Technical Report NASA CR-755, National Aeronautics and Space Administration*, 1966.

[37] J. M. Mendel and R. W. McLaren. Reinforcement learning control and pattern recognition systems. In *In Mendel, J. M. and Fu, K. S., editors, Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, pages 287–318, 1970.

[38] M. L. Minsky. *Theory of Neural-Analog Reinforcement Systems and its Application to the Brain-Model Problem*. PhD thesis, Princeton University, 1954.

[39] Marvin Minsky. Steps toward artificial intelligence. In *Computers and Thought*, pages 406–450. McGraw-Hill, 1961.

[40] Juan Antonio Breña Moral. *Multithreading with Java leJOS*. 2008.

[41] Juan Antonio Breña Moral. *Develop leJOS programs Step by Step*. 2009.

[42] Hans Moravec. The age of robots. volume 46, pages 90–97, New York, NY, USA, October 1993. ACM.

[43] Shimon Y. Nof. *Handbook of Industrial Robotics*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 1999.

[44] Denise de Oliveira, Ana L. C. Bazzan, Bruno C. da Silva, E. W. Basso, L. Nunes, R. J. F. Rossetti, E. C. Oliveira, R. Silva, and L. C. Lamb. Reinforcement learning based control of traffic lights in non-stationary environments: a case study in a microscopic simulator. In Barbara Dunin-Keplicz, Andrea Omicini, and Julian Padget, editors, *Proceedings of the 4th European Workshop on Multi-Agent Systems, (EUMAS06)*, pages 31–42, December 2006.

[45] J.J. O'Neill. *Prodigal genius: the life of Nikola Tesla*. Adventures Unlimited Press, 2008.

[46] Mihoko Otake, Yoshiharu Kagami, Masayuki Inaba, and Hirochika Inoue. Motion design of a starfish-shaped gel robot made of electro-active polymer gel. *Robotics and Autonomous Systems*, 40(2-3):185–191, 2002.

[47] Jing Peng and Ronald J. Williams. Incremental multi-step q-learning. In *Machine Learning*, pages 226–232. Morgan Kaufmann, 1996.

[48] David L. Pepyne and Christos G. Cassandras. Optimal dispatching control for elevator systems during uppeak traffic. 5(6):1–15, 1997.

[49] D. L. Poole, Alan Mackworth, and R. G. Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, New York, January 1998.

[50] Stuart Ian Reynolds. *Reinforcement Learning with Exploration*. PhD thesis, The University of Birmingham, 2002.

[51] Carlos Henrique Costa Ribeiro. A tutorial on reinforcement learning techniques, 2002.

[52] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, 1994.

[53] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach Second Edition*. Prentice Hall, 2003.

[54] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach Third Edition*. Prentice Hall, 2009.

[55] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[56] Stefan Schaal and Christopher G. Atkeson. Robot Juggling: An Implementation of Memory-based Learning. *Control Systems Magazine*, 14(1):57–71, 1994.

[57] Nicol N. Schraudolph, Peter Dayan, and Terrence J. Sejnowski. Temporal difference learning of position evaluation in the game of go. In *Advances in Neural Information Processing Systems 6*, pages 817–824. Morgan Kaufmann, 1994.

[58] Alan C. Schultz and Illah R. Nourbakhsh. Guest editorial: Computational intelligence in robotics and automation. *Auton. Robots*, 9(1):5–6, 2000.

[59] Burr Settles. *CURIOUS MACHINES: ACTIVE LEARNING WITH STRUCTURED INSTANCES*. PhD thesis, UNIVERSITY OF WISCONSIN–MADISON, 2008.

[60] Reid Simmons, Dani Goldberg, Adam Goode, Michael Montemerlo, Nicholas Roy, Brennan Sellner, Chris Urmson, Magda Bugajska, Michael Coblenz, Matt Macmahon, Dennis Perzanowski, Ian Horswill, Robert Zubek, David Kortenkamp, Bryn Wolfe, Tod Milam, and Bruce Maxwell. Grace: An autonomous robot for the aaai robot challenge, 2002.

[61] Komei Sugiura, Makoto Akahane, Takayuki Shiose, Katsunori Shimohara, and Osamu Katai. Exploiting interaction between sensory morphology and learning. *2005 IEEE International Conference on Systems, Man and Cybernetics*, pages 10–12, 2005.

[62] Funlade T. Sunmola and Jeremy L. Wyatt. Reinforcement learning using optimistic process filtered models, 2005.

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

[64] Gerald Tesauro. Practical issues in temporal difference learning. In *Machine Learning*, pages 257–277, 1992.

[65] Gerald Tesauro. TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, 6(2):215–219, March 1994.

[66] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38:58–68, March 1995.

[67] E. L. Thorndike. Animal intelligence. 1911.

[68] Sebastian Thrun. Learning to play the game of chess. In *Advances in Neural Information Processing Systems 7*, pages 1069–1076. The MIT Press, 1995.

[69] Branislav Thurský and Gabriel Gašpar. Using 3pi robot pololu in the teaching process. In *Proceedings of the 1st international conference on Robotics in Education, RiE2010*, pages 243–245. FEI STU, Slovakia, 2010.

[70] C. Touzet, N. Giambiasi, and S. Sehad. Improving reinforcement learning of an obstacle avoidance behavior with forbidden sequences of actions. In *International Conference on Robotics and Manufacturing," Cancun*, pages 14–16, 1995.

[71] Thomas P. Trappenberg. Csci 4155: Machine learning and robotics, 2010.

[72] Eiji Uchibe, Minoru Asada, and Koh Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In *In Proc. of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1329–1336, 1996.

[73] R Valero, Y H Ko, S Chauhan, O Schatloff, A Sivaraman, R F Coelho, F Ortega, K J Palmer, R Sanchez-Salas, H Davila, and et al. Robotic surgery: History and teaching impact. *Actas Urologicas Espanolas*, (xx), 2011.

[74] Peter Vamplew. Lego$^{TM}$ mindstorms$^{TM}$ robots as a platform for teaching reinforcement learning. In *International Conference on Artificial Intelligence in Science and Technology*, pages 21–25, 2004.

[75] Sethu Vijayakumar, Tomohiro Shibata, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Autonomous Robot*, page 2002, 2003.

[76] Bram Vlist, Rick Westelaken, Christoph Bartneck, Jun Hu, Rene Ahn, Emilia Barakova, Frank Delbressine, and Loe Feijs. Teaching machine learning to design students. In *Proceedings of the 3rd international conference on Technologies for E-Learning and Digital Entertainment*, Edutainment '08, pages 206–217, Berlin, Heidelberg, 2008. Springer-Verlag.

[77] Michael D. Wagner, Dimitrios Apostolopoulos, Kimberly Shillcutt, Benjamin Shamah, Reid Simmons, William Red Whittaker, and William "red Whittaker. The science autonomy system of the nomad robot. In *In Proc. of International Conference on Robotics and Automation*, pages 1742–1749, 2001.

[78] M. D. Waltz and K. S Fu. A heuristic approach to reinforcment learning control systems. In *IEEE Transactions on Automatic Control*, volume 10, pages 390–398, 1965.

[79] L F Wang, K C Tan, and V Prahlad. Developing khepera robot applications in a webots environment. *MHS2000 Proceedings of 2000 International Symposium on Micromechatronics and Human Science Cat No00TH8530*, pages 71–76, 2000.

[80] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[81] C. J. C. H. Watkins and P. Dayan. Q-learning. In *Machine Learning*, volume 8, pages 279–292, 1992.

[82] David Wettergreen, Michael Wagner, Dominic Jonak, Vijayakumar Baskaran, Matthew Deans, Stuart Heys, David Pane, Trey Smith, James Teza, David Thompson, Paul Tompkins, and Chris Williams. Long-distance autonomous survey and mapping in the robotic investigation of life in the atacama desert, 2008.

[83] Jeremy Wyatt. Issues in putting reinforcement learning onto robots. In *Mobile Robotics Workshop, 10th Biennial Conference of the AISB.*, 1995.

[84] Yousheng Yang, Claudio Semini, Nikos G. Tsagarakis, Emanuele Guglielmino, and Darwin G. Caldwell. Leg mechanisms for hydraulically actuated robots. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, pages 4669–4675, Piscataway, NJ, USA, 2009. IEEE Press.

[85] Xu Yuan, Lucian Busoni, and Robert Babuska. Reinforcement learning for elevator control, 2008.