CENTRO UNIVERSITÁRIO DA FEI
DANILO HERNANI PERICO

USO DE HEURÍSTICAS OBTIDAS POR MEIO DE DEMONSTRAÇÕES PARA ACELERAÇÃO DO APRENDIZADO POR REFORÇO

São Bernardo do Campo
2012

DANILO HERNANI PERICO

Uso de Heurísticas obtidas por meio de Demonstrações para Aceleração do Aprendizado por Reforço

Dissertação de Mestrado apresentada ao Centro Universitário da FEI para obtenção do título de Mestre em Engenharia Elétrica, orientado pelo Prof. Dr. Reinaldo Augusto da Costa Bianchi.

São Bernardo do Campo
2012

Perico, Danilo Hernani.
Uso de heurísticas obtidas por meio de demonstrações para aceleração do aprendizado por reforço / Danilo Hernani Perico. São Bernardo do Campo, 2012.
87 f. : il.

Dissertação (Mestrado) - Centro Universitário da FEI.
Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi

1. Inteligência Artificial. 2. Aprendizagem de máquina. 3. Aprendizado por reforço. 4. Aprendizado por demonstração. 5. Robótica. I. Bianchi, Reinaldo Augusto da Costa, orient. II. Título.

CDU 62-52

Centro Universitário da **FEI**

APRESENTAÇÃO DE DISSERTAÇÃO
ATA DA BANCA JULGADORA

PGE-10

Programa de Mestrado de Engenharia Elétrica

Aluno: Danilo Hernani Perico **Matrícula:** 110117-9

Título do Trabalho: Uso de heurísticas obtidas por meio de demonstrações para aceleração do aprendizado por reforço.

Área de Concentração: Inteligência Artificial Aplicada à Automação

Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi

Data da realização da defesa: 27/11/2012

ORIGINAL ASSINADA

A Banca Julgadora abaixo-assinada atribuiu ao aluno o seguinte:

APROVADO ☒ REPROVADO ☐

São Bernardo do Campo, 27 de Novembro de 2012.

MEMBROS DA BANCA JULGADORA

Prof. Dr. Reinaldo Augusto da Costa Bianchi

Ass.: _____

Prof. Dr. Paulo Eduardo Santos

Ass.: _____

Prof. Dr. Renê Pegoraro

Ass.: _____

VERSÃO FINAL DA DISSERTAÇÃO

ENDOSSO DO ORIENTADOR APÓS A INCLUSÃO DAS RECOMENDAÇÕES DA BANCA EXAMINADORA

_____

Aprovação do Coordenador do Programa de Pós-graduação

_____

Prof. Dr. Carlos Eduardo Thomaz

A Deus, aos meus pais e à minha esposa Lucivânia.

AGRADECIMENTOS

Agradeço inicialmente a Deus, por tudo.

Aos meus pais, Valter e Ana Clara, por sempre terem me apoiado em todas as minhas decisões e por sempre terem intercedido a Deus, pela minha felicidade.

Agradeço também a minha esposa, Lucivânia, por todo amor, incentivo e carinho. Por ter compreendido as minhas ausências durante a realização deste trabalho, dedico-lhe essa conquista com gratidão e amor.

Ao meu orientador, Professor Dr. Reinaldo Augusto da Costa Bianchi, pela paciência, pelas sugestões, pelos conselhos e pela motivação.

Aos professores, amigos e todos os integrantes do mestrado pela excelente companhia durante toda esta fase de intenso aprendizado que o mestrado me proporcionou.

Agradeço ao Centro Universitário da FEI por fornecer o necessário para a minha progressão pessoal e intelectual.

"*Não tentes ser bem sucedido, tenta antes ser um homem de valor.*"

Albert Einstein

RESUMO

O Aprendizado por Reforço é um método bastante conhecido para resolução de problemas em que o agente precisa aprender por meio de interação direta com o ambiente. Porém, esta técnica tem o problema de não ser eficiente o bastante, devido ao seu alto custo computacional. Este trabalho tem como objetivo propor e testar o algoritmo de Aprendizado por Reforço acelerado por heurísticas obtidas por meio de demonstrações, no domínio da navegação robótica, de modo que exemplos de caminhos são utilizados para a construção da heurística. Experimentos foram efetuados no meio conhecido como Mundo de Grades e em ambiente simulado no *software* Player/Stage. Os resultados experimentais permitiram concluir que o algoritmo proposto pode melhorar o desempenho do aprendizado, diminuindo consideravelmente o número de passos necessários para se alcançar o objetivo, quando comparado aos algoritmos convencionais de Aprendizado por Reforço.

Palavras-chave: Inteligência Artificial, Aprendizagem de Máquina, Aprendizado por Reforço, Aprendizado por Demonstração, Robótica

ABSTRACT

The Reinforcement Learning is a well known method for solving problems where the agent needs to learn through direct interaction with the environment. However, this technique is not efficient enough, due to its high computational cost. This work aims to propose and test the Reinforcement Learning accelerated by heuristics obtained through demonstrations algorithm, in the domain of robotic navigation, so that examples of paths are used to build the heuristic. Experiments were made in the area known as Grid World and in the simulated environment using software Player/Stage. Experimental results allowed to conclude that the proposed algorithm can improve the learning performance, reducing significantly the number of steps needed to achieve the goal, when compared to the conventional Reinforcement Learning algorithms.

Keywords: Artificial Intelligence, Machine Learning, Reinforcement Learning, Learning from Demonstration, Robotics

LISTA DE FIGURAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS

AI Inteligência Artificial (*Artificial Intelligence*)

CMAC *Cerebellar Model Articulation Controller*

DP Programação Dinâmica (*Dynamic Programming*)

HAL Camada de Abstração de Hardware (*Hardware Abstraction Layer*)

HAQL Q-Learning Acelerado por Heurísticas (*Heuristically Accelerated Q-Learning*)

HARL Aprendizado por Reforço Acelerado por Heurísticas (*Heuristically Accelerated Reinforcement Learning*)

HfDAQL Q-Learning Acelerado por Heurísticas obtidas por meio de Demonstrações (*Heuristics from Demonstration to Accelerate Q-Learning*)

HfDARL Aprendizado por Reforço Acelerado por Heurísticas obtidas por meio de Demonstrações (*Heuristics from Demonstration to Accelerate Reinforcement Learning*)

LfD Aprendizado por Demonstração (*Learning from Demonstration*)

MCL Localização de Monte Carlo (*Monte Carlo Localization*)

MDP Processo Markoviano de Decisão (*Markov Decision Process*)

QS *Q-Learning* combinado com Espalhamento Espacial (*Q-Learning combined with Spatial Spreading*)

QSx QS com inserção de informações prévias sobre o domínio no formato de graus de similaridade e a inclusão de restrições

RL Aprendizado por Reforço (*Reinforcement Learning*)

RNA Redes Neurais Artificias

TD Aprendizado por Diferenças Temporais (*Temporal Difference Learning*)

LISTA DE SÍMBOLOS

α Taxa de aprendizado
β Variável de influência da heurística
γ Fator de desconto sobre o reforço futuro
ε Taxa de exploração do ambiente
η Valor máximo predefinido concedido a H
λ Fator de desconto de diferenças temporais futuras
ξ Variável de influência da heurística
π Política
π^* Política ótima
σ Função de espalhamento
τ Fator de similaridade
n Grau de diversidade
H Função heurística
Q Função ação-valor
Q^* Função ação-valor ótima
V Função valor
V^* Função valor ótima
$\mathcal{A}$ Conjunto de ações
$\mathcal{R}$ Conjunto de reforços
$\mathcal{S}$ Conjunto de estados
$\mathcal{T}$ Conjunto de transições

SUMÁRIO

1 INTRODUÇÃO

A capacidade de um agente robótico móvel aprender tem sido buscada por pesquisadores de Inteligência Artificial (*Artificial Intelligence* – AI) há vários anos. Dentre as diversas formas de aprendizados desenvolvidos com este intuito, pode-se citar o Aprendizado por Reforço (*Reinforcement Learning* – RL) como uma das técnicas mais usuais.

O RL é uma técnica de aprendizado bastante usual, pois o agente aprende por meio de interação direta com o ambiente e seu algoritmo sempre converge para uma situação de equilíbrio (SUTTON, 1988). No RL, um agente pode aprender em um ambiente não conhecido previamente, por intermédio de experimentações. Dependendo de sua atuação, o agente recebe uma recompensa ou uma penalização e, desta forma, o algoritmo encontra um conjunto de ações que levam o agente a percorrer o caminho ótimo. A este conjunto, formado pelas melhores ações, dá-se o nome de política ótima (π^*).

O principal problema encontrado no RL é referente à grande quantidade de iterações necessárias para que o algoritmo possa convergir, ou seja, o RL tem alto custo computacional, o que o torna um processo bastante lento. Assim, quando levamos este método de aprendizado para um modelo dinâmico e não determinístico, como é o caso da robótica móvel autônoma, são necessárias algumas técnicas para se acelerar este método de aprendizado.

1.1 Motivação

A principal motivação deste trabalho é tornar a navegação de robôs móveis autônomos mais eficiente, uma vez que os métodos existentes de RL, incluindo os acelerados, podem não apresentar bons resultados quando programados em robôs dinâmicos e não determinísticos cuja atuação deve acontecer em ambientes complexos e extensos.

1.2 Proposta

A proposta desta dissertação é apresentar uma classe de algoritmo que mantenha as principais propriedades do Aprendizado por Reforço, mas que seja mais rápida, eficaz e de simples implementação no domínio da robótica móvel autônoma. O uso de heurísticas pode ser considerado um dos métodos para se acelerar o RL (BIANCHI, 2004) no domínio proposto, mas apresenta a desvantagem de não ter um estudo completo de seu desempenho em ambientes mais complexos e dinâmicos.

Outra técnica de aprendizagem existente, conhecida por tornar a comunicação de seres humanos com robôs móveis mais intuitiva, é o Aprendizado por Demonstração (SCHAAL,

1997), que consiste na técnica de se desenvolver políticas através de exemplos de sequências de estado-ação, que são gravados durante a demonstração. Porém, este modelo de aprendizado tem a desvantagem de ser bastante limitado pela qualidade dos dados recebidos nas demonstrações.

Este trabalho tem como propósito a fusão entre o Aprendizado por Reforço Acelerado por Heurísticas (BIANCHI, 2004) e o Aprendizado por Demonstração (SCHAAL, 1997), tendo como inspiração o aprendizado em duas fases, proposto por Smart e Kaelbling (2002), que basicamente é um RL que utiliza demonstrações como método de obtenção de conhecimento prévio do domínio e tem a vantagem de tornar desnecessário o fato do programador ter que conhecer em detalhes o funcionamento dos sensores do robô, mas que, por outro lado, necessita de muitas demonstrações para exercer alguma melhora no aprendizado.

Portanto, espera-se alcançar, por meio da proposta deste trabalho, o aprendizado de robôs móveis de uma maneira mais rápida e simples, utilizando-se para isso poucas demonstrações de caminhos ao robô.

1.3 Objetivo

O objetivo deste trabalho é propor, analisar e testar o Aprendizado por Reforço Acelerado por Heurísticas obtidas por meio de Demonstrações no domínio da robótica móvel autônoma.

1.4 Domínio

O domínio abordado neste trabalho é o dos agentes robóticos inteligentes móveis. Este domínio normalmente apresenta agentes dinâmicos e não determinísticos.

Neste trabalho o ambiente é representado por dois domínios. O primeiro, e mais simples, é o ambiente conhecido como Mundo de Grades (*Grid World*), que considera um espaço formado por células de tamanhos fixos que constituem uma grade. Neste domínio a posição do robô e a posição relativa dos objetos que o cercam é sempre exata. O segundo, e mais complexo, é o ambiente simulado no *software* Player/Stage, que é capaz de reproduzir, de maneira virtual, um ambiente fechado com uma população de robôs. Ao contrário do que acontece no Mundo de Grades, neste domínio o robô depende dos seus sensores para identificar os obstáculos e se localizar, e isto faz com que a sua posição e a posição relativa dos obstáculos não sejam tão exatas.

1.5 Organização do Trabalho

Este trabalho está organizado da seguinte maneira: o capítulo 2 mostra uma revisão bibliográfica sobre o RL, apresentando os algoritmos *Q-Learning* e $SARSA$. No capítulo 3 são apresentados os métodos já conhecidos para aceleração do RL, como as Generalizações, as Abstrações e o uso de Heurísticas.

No capítulo 4 é apresentado o Aprendizado por Demonstração e suas técnicas de aquisição de dados e de políticas. O capítulo 5 exibe a proposta deste trabalho em detalhes.

Já no capítulo 6 são apresentados os resultados obtidos nos experimentos realizados para o domínio dos robôs móveis, tanto no Mundo de Grades, como em ambiente simulado com o uso do *software* Player/Stage.

Finalmente, o capítulo 7 conclui este trabalho com uma discussão sobre os resultados obtidos e sugere possíveis propostas de extensão de estudo do tema.

2 APRENDIZADO POR REFORÇO

Historicamente, o Aprendizado por Reforço (*Reinforcement Learning* – RL) teve como percursor o modelo de aprendizagem proposto por Samuel (1959), que utilizou algumas técnicas de aprendizagem de máquina para criar um algoritmo capaz de aprender a jogar damas. Após um longo período com poucas pesquisas sendo realizadas nesta área de estudo, Sutton (1988) apresentou o Aprendizado por Diferença Temporal (*TD–Learning*) que é um dos algoritmos mais impotantes para o entendimento do RL. Inspirado pelo algoritmo TD, Watkins (1989) propôs o algoritmo *Q-Learning* que, até hoje, é uma das maneiras mais populares de implementar o RL. Depois disso, diversos outros trabalhos foram realizados utilizando os conceitos básicos deste aprendizado como, por exemplo, o estudo de Aprendizagem de Máquina realizado por Mitchell (1997), o desenvolvimento do algoritmo $SARSA$ realizado por Sutton (1996) e o abrangente estudo sobre o Aprendizado por Reforço apresentado por Sutton e Barto (1998). Todas estas pesquisas contribuíram significativamente para a fundamentação desta técnica.

O Aprendizado por Reforço é um método de aprendizagem de máquina que consiste na técnica pela qual um agente, sem conhecimento prévio, possa aprender através de interações com o ambiente, ativamente, por experimentação e, desta forma, descobrir uma política ótima de atuação.

Uma política de atuação (π) pode ser definida como uma coleção de ações que determina por quais estados o agente deve percorrer; logo a política ótima de atuação (π^*) representa a coletânea das ações que resultam no menor custo para o agente completar a sua tarefa.

Este algoritmo funciona através do uso de um sinal escalar de reforço $r_{s,a}$, que pode ser uma recompensa ou uma penalização, recebido em cada alteração no estado s_t. Cada ação a_t do agente é determinada pelo conhecimento obtido até então e indica qual ação deverá ser tomada visando obter a maior recompensa possível (figura 2.1). Assim, o RL é capaz de encontrar, após diversas iterações, qual a melhor ação a ser tomada em cada estado.

Logo, no RL, o agente aprendiz tem como meta aprender uma política ótima de ações que maximize as recompensas recebidas durante o processo de execução, não importando o seu estado inicial. Esse aprendizado pode ser formalizado através do conceito do Processo Markoviano de Decisão (*Markov Decision Process* – MDP), detalhado a seguir.

2.1 Processo Markoviano de Decisão

Uma maneira muito usual de se formalizar o RL é por meio do Processo Markoviano de Decisão (*Markov Decision Process* – MDP).

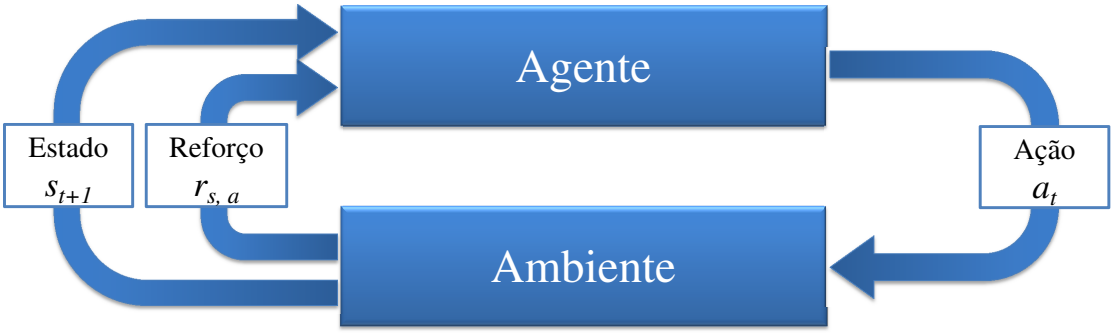


Figura 2.1 – Interação entre o agente e o ambiente no Aprendizado por Reforço.
Fonte: Autor

O MDP é uma teoria matemática bem estabelecida e que tem fundamentos sólidos, portanto a formalização do RL através dos conceitos de MDP dá o suporte necessário para a estruturação deste aprendizado. Um Processo Markoviano de Decisão é definido pela quadrupla $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$ (LITTMAN, 1994; KAELBLING; LITTMAN; MOORE, 1996; MITCHELL, 1997), onde:

a) $\mathcal{S}$ é um conjunto finito de estados;

b) $\mathcal{A}$ é um conjunto finito de ações;

c) $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ é uma função recompensa;

d) $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(S)$ é uma função de transição de estados, onde $\Pi(S)$ é um mapeamento da transição de estados em probabilidades.

A resolução de um MDP consiste em gerar uma política ($\pi : \mathcal{S} \times \mathcal{A}$) que maximiza (ou minimiza) uma função. Como o objetivo do RL é aprender uma política que maximiza a função de recompensa acumulada ao longo do tempo (KAELBLING; LITTMAN; MOORE, 1996), podemos utilizar o MDP para modelar matematicamente este objetivo.

Dependendo da maneira como as transições ocorrem, o MDP pode ser determinístico ou não-determinístico. Quando o MDP apresenta apenas uma transição válida para o par estado-ação, pode-se dizer que o sistema é determinístico; porém se uma transição puder resultar em mais do que um estado, considerando que cada estado possível apresenta uma probabilidade, o sistema é não-determinístico.

É possível computar o valor da recompensa esperada através da função de custo esperado $V^{\pi}(s_t)$, criada quando se utiliza uma política arbitrária π, a partir de um estado inicial s_t. Esta

função concede um valor numérico, representado pela somatória dos reforços futuros seguindo uma política ótima, para cada estado.

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{2.1}$$

Onde:

a) r_{t+i} é a sequência de recompensas recebidas a partir do estado s_t, utilizando a política π para selecionar as ações;

b) γ é um fator de desconto que determina o quanto as recompensas futuras serão consideradas, admitindo intervalo $0 \leqslant \gamma < 1$.

O agente aprendiz tem como objetivo determinar a política ótima (π^*). Para isso o RL deve aprender uma política π que maximize $V^{\pi}(s_t)$, para todo estado $s \in S$.

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), \forall s \tag{2.2}$$

Um método bastante conhecido de se encontrar a política ótima π^* é através do uso dos algoritmos de Programação Dinâminca (*Dynamic Programing* – DP) (BERTSEKAS, 1987).

2.2 Programação Dinâmica

A Programação Dinâmica (*Dynamic Programing* – DP) é um conjunto de algoritmos iterativos que podem encontrar a política ótima de atuação, porém ela precisa de um modelo completo de mundo, ou seja, conhecer a função de transição de estados, e tem alto custo computacional.

Um dos pontos fundamentais da DP é o Princípio da Otimalidade de Bellman (BELLMAN, 1957). Segundo este princípio, um agente que segue uma política ótima a partir de um estado inicial até um estado final, passando por um estado intermediário, é equivalente a um agente que segue a melhor política a partir de um estado inicial até um intermediário e, em seguida, segue a melhor política do estado intermediário até o final.

Partindo deste princípio, podemos concluir que para encontrar a política ótima de um sistema no estado s_i, basta encontrar a ação a_i que leva ao melhor estado s_{i+1}, e assim por diante até o estado final.

Dois algoritmos de Programação Dinâmica bastante utilizados para resolver MDPs são: o algoritmo de Iteração de Valor e o algoritmo de Iteração de Política.

O algoritmo de Iteração de Valor (BERTSEKAS, 1987) utiliza a equação 2.3 para calcular a função valor ótima V^* de forma iterativa:

$$V(s_t) \leftarrow \max_{\pi(s_t)} \left[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in S} T(s_t, \pi(s_t), s_{t+1}) V'(s_{t+1}) \right] \tag{2.3}$$

Onde:

a) o estado atual é representado por s_t;

b) a ação realizada em s_t é $\pi(s_t) = a_t$;

c) o próximo estado é representado por s_{t+1};

d) a função valor é V e a função valor na iteração anterior é V'.

A política ótima π^* é encontrada no final do processamento, pela equação 2.4:

$$\pi^*(s_t) \leftarrow \arg\max_{\pi(s_t)} \left[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in S} T(s_t, \pi(s_t), s_{t+1}) V^*(s_{t+1}) \right] \tag{2.4}$$

O algoritmo de Iteração de Valor completo é apresentado no algoritmo 2.1.

Repita
Compute a função valor $V(s)$ por meio da equação:
$V(s_t) \leftarrow \max_{\pi(s_t)}[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in S} T(s_t, \pi(s_t), s_{t+1}) V'(s_{t+1})]$
Até que o valor de $V(s)$ = $V'(s)$.
Calcule π^* por intermédio da equação:
$\pi^*(s_t) \leftarrow \max_{\pi(s_t)}[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in S} T(s_t, \pi(s_t), s_{t+1}) V^*(s_{t+1})]$

Onde $s = s_t$ e $s' = S_{t+1}$.

Algoritmo 2.1 – Iteração de Valor (BERTSEKAS, 1987)

Por outro lado, o algoritmo de Iteração de Política (BERTSEKAS, 1987; KAELBLING; CASSANDRA; KURIEN, 1996) não usa a função valor ótima para encontrar a política. Ele a manipula diretamente por intermédio da escolha de uma política inicial estacionária $\pi^0 = \{a_0^0, a_1^0, ...\}$ que é utilizada para calcular V^{π^0}, com o seguinte sistema:

$$V^{\pi}(s_t) = r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in S} T(s_t, \pi(s_t), s_{t+1}) V^{\pi}(s_{t+1}) \tag{2.5}$$

Então, uma política π' é calculada pela maximização da soma do reforço $r(s, \pi(s))$ com o valor $V^{\pi}(s')$ do estado sucessor com o desconto de γ:

$$\pi'(s_t) \leftarrow \arg\max_{\pi(s_t)} \left[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in S} T(s_t, \pi(s_t), s_{t+1}) V^{\pi}(s_{t+1}) \right] \tag{2.6}$$

O processo é repetido até que a política ótima π^* seja encontrada. O primeiro sistema (equação 2.5) é chamado de Passo de Avaliação da Política, já a maximização da equação 2.6 é chamada de Passo da Melhora da Política, pois é onde a política é modificada para melhorar a função valor.

A política $\pi'(s_t)$ é gulosa, pois escolhe sempre a ação que maximiza $V^{\pi}(s_t)$. É possível, também, a utilização de outras estratégias de exploração, como as apresentadas na seção 2.6 (guardadas as diferenças entre Q e V).

O algoritmo de Iteração de Política completo é apresentado no algoritmo 2.2.

Inicialize $\pi \leftarrow \pi^0$ arbitrariamente
  Repita
    Passo 1: Compute a função valor $V^{\pi}(s)$ com a política π:
      $V^{\pi}(s_t) = r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in S} T(s_t, \pi(s_t), s_{t+1}) V^{\pi}(s_{t+1})$
    Passo 2: Melhore a política em cada estado:
      $\pi'(s_t) \leftarrow \arg\max_{\pi(s_t)} \left[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in S} T(s_t, \pi(s_t), s_{t+1}) V^{\pi}(s_{t+1}) \right]$
    $\pi \leftarrow \pi'$
  Até que $V^{\pi}(s) = V^{\pi'}(s)$.

  Onde $s = s_t$ e $s' = S_{t+1}$ e $a = a_t = \pi(s_t)$.

Algoritmo 2.2 – Iteração de Política (BERTSEKAS, 1987)

Os MDPs podem ser solucionados através de alguns métodos de Programação Dinâmica, pois, no momento em que a função valor V não puder mais melhorar, a política encontrada será a ótima.

2.3 Aprendizado por Diferença Temporal

O algoritmo mais importante para o estudo do RL é o Aprendizado por Diferença Temporal (*TD–Learning*) (SUTTON, 1988). Este método é inspirado por uma combinação entre a Programação Dinâmica e os Métodos de Monte Carlo, isto é, ele é iterativo e não precisa de conhecimento prévio sobre o ambiente.

Este algoritmo calcula a soma dos reforços em cada estado $V(s)$ de maneira iterativa, e não por um sistema de equações como acontece na Programação Dinâmica. O método de

Diferença Temporal mais simples é o $TD(0)$, que atualiza o valor de $V(s)$ usando a seguinte regra:

$$V(s_t) \leftarrow V(s_t) + \alpha[rt + 1 + \gamma V(s_{t+1}) - V(s_t)] \tag{2.7}$$

Onde:

a) o estado atual é representado por s_t;

b) o reforço ao se realizar uma ação em um determinado estado é r_{t+1};

c) o próximo estado é representado por s_{t+1};

d) a taxa de aprendizado é α, admitindo intervalo $0 \leqslant \alpha \leqslant 1$.

O $TD(0)$ completo utilizado para estimar a função valor ótima V^{π} é apresentado no algoritmo 2.3.

Inicialize $V(s)$ arbitrariamente
Faça para cada episódio
Inicialize s
  Repita // (para cada passo dentro do episódio)
    $a \leftarrow$ ação dada pela política π em s
    Executar ação a, observar a recompensa r e o próximo estado $s + 1$
    $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s + 1) - V(s)]$
    $s \leftarrow s + 1$
  Até que s seja o último estado

Algoritmo 2.3 – $TD(0)$ para estimar V^{π} (SUTTON; BARTO, 1998)

2.4 Q-Learning

Utilizando o método do Aprendizado por Diferença Temporal, Watkins (1989) propôs o algoritmo *Q-Learning*, que é considerado como um dos mais populares de RL, com o propósito de aprender iterativamente uma política ótima π^* em situações em que o modelo do sistema é desconhecido.

O algoritmo *Q-Learning* tem como objetivo maximizar os valores da função de estimação Q para cada estado e, desta forma, também encontrar o valor de custo de um estado $V(s)$, uma vez que o custo de um estado é o valor máximo de $Q(s, a)$, independentemente da política seguida, assim este algoritmo é definido como um método *off-policy*.

A função Q é definida como sendo a soma do reforço recebido pelo agente por ter realizado a ação a_t no estado s_t em um momento t, mais o valor de seguir a política ótima (SUTTON; BARTO, 1998). Desta forma, pode-se concluir que:

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} T(s_t, a_t, s_{s+1}) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \tag{2.8}$$

A regra de atualização do algoritmo *Q-Learning* é:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.9}$$

Onde:

a) s_t é o estado atual;

b) a_t é a ação realizada em s_t;

c) $r(s_t, a_t)$ é o reforço recebido após realizar a_t em s_t;

d) s_{t+1} é o próximo estado;

e) a_{t+1} é a ação realizada em s_{t+1};

f) γ é o fator de desconto ($0 \leq \gamma < 1$);

g) α é a taxa de aprendizagem ($0 < \alpha < 1$).

Como o Q é definido por uma função recursiva, o algoritmo tem a base para iterativamente aproximar o próprio Q (WATKINS, 1989). As ações usadas durante este processo de aproximação são geralmente definidas por meio de uma regra de transição, que será melhor definida na seção 2.6. Normalmente o *Q-Learning* utiliza a estratégia aleatória gulosa 2.6.1.

O *Q-Learning* completo é apresentado no Algoritmo 2.4.

Inicialize $Q(s, a)$ arbitrariamente
  Repita
    Visite o estado s
    Selecione uma ação a de acordo com a regra de transição de estado
    Execute a ação a
    Receba o reforço $r(s, a)$ e observe o próximo estado s'
    Atualize os valores de $Q(s, a)$ de acordo com regra de atualização:
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    Atualize o estado $s \leftarrow s'$
  Até que algum critério de parada seja atingido

  Onde s' é o próximo estado e a' é a ação realizada em s'

Algoritmo 2.4 – *Q-Learning* (WATKINS, 1989)

2.5 SARSA

O algoritmo $SARSA$ (SUTTON, 1996) foi inicialmente chamado de *Q-Learning* Modificado (RUMMERY; NIRANJAN, 1994), pois usa a mesma regra de aprendizado do *Q-Learning* porém com a eliminação da maximização das ações. O $SARSA$ aprende a política durante a execução do algoritmo, estimando o valor de uma política ao mesmo tempo que a usa na interação com o ambiente, por isso é um método chamado de *on-policy*.

A regra de iteração segue a expressão:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.10}$$

O $SARSA$ requer que a escolha das ações seja feita de maneira aleatória, partindo-se de uma determinada distribuição de probabilidades, como, por exemplo, a Exploração Boltzmann, que será tratada próxina ma seção. Segundo Sutton (1996), o algoritmo $SARSA$ tem um comportamento melhor do que o *Q-Learning* em situações na qual penalidades precisam ser evitadas ou onde o conjunto de ações é muito grande, porém ele apresenta mais dificuldades para alcançar estados com recompensas negativas.

O nome $SARSA$ é proveniente de sua regra de atualização que utiliza todos os elementos da quíntupla $< s_t, a_t, r, s_{t+1}, a_{t+1} >$, responsável por definir a transição de um par estado-ação para o próximo e, assim, atualizar Q.

2.6 Estratégias de exploração

As estratégias de exploração mais usuais para o Aprendizado por Reforço são as chamadas indiretas, na qual nenhum conhecimento sobre a exploração é utilizado. Seguindo este conceito, a exploração aleatória gulosa e a exploração de Boltzmann são, habitualmente, os métodos mais usados.

2.6.1 Exploração aleatória ε–$Greedy$

Uma das estratégias mais comuns de exploração pode ser a exploração aleatória, conhecida como ε–$Greedy$. A ideia central desta exploração é fazer o agente executar a ação que tenha o maior valor de Q com probabilidade $1 - \varepsilon$ e escolher uma ação aleatória com probabilidade ε. Desta forma, a estratégia gulosa determina uma política π que tira proveito das iterações anteriores do algoritmo.

A transição de estados é dada pela regra:

$$\pi(s_t) = \begin{cases} a_{random} & \text{se q} \leqslant \varepsilon, \\ \arg\max_{a_t} \ Q_t(s_t, a_t) & \text{caso contrário.} \end{cases} \tag{2.11}$$

Onde:

a) q é um valor aleatório com distribuição uniforme sobre [0,1];

b) ε é o parâmetro que define a taxa de exploração/explotação ($0 \leqslant \varepsilon \leqslant 1$): quanto menor o valor de ε, menor a probabilidade da escolha aleatória;

c) a_{random} é a ação aleatória selecionada dentre as ações possíveis em s_t.

2.6.2 Exploração Boltzmann

A exploração Boltzmann é, normalmente, utilizada com uma frequência menor do que a exploração aleatória gulosa. Essa exploração utiliza a fórmula 2.12 para calcular a probabilidade de executar a ação a no estado s. Assim, quanto maior o valor de $Q(s, a)$, maior será a sua probabilidade.

$$P(s, a) = e^{Q(s,a)/T} / \sum u^{e^{Q(s,a)/T}} \tag{2.12}$$

Onde:

a) $P(s, a)$ é a probabilidade de selecionar a ação a no estado s;

b) u é uma das possíveis ações que podem ser executadas no estado s;

c) T é um parâmetro de *temperatura*, no qual valores pequenos favorecem a explotação e valores grandes favorecem a exploração.

2.7 Considerações finais

Este capítulo descreveu brevemente os conceitos básicos do Aprendizado por Reforço, que é uma importante técnica de aprendizagem de máquina, que permite que o agente aprenda por meio de interações diretas com o ambiente, sem, necessariamente, ter um conhecimento prévio do meio. Foram citados também alguns dos principais algoritmos utilizados neste método de aprendizado, como o *Q-Learning* e o $SARSA$.

Contudo, por ser um método iterativo, o Aprendizado por Reforço é bastante lento, já que é comum a necessidade de milhares de iterações para que o algoritmo encontre a conver-

gência. Portanto, normalmente são necessárias algumas técnicas, que serão melhor abordadas no próximo capítulo, para acelerar o RL.

3 ACELERAÇÃO DO APRENDIZADO POR REFORÇO

Os algoritmos de RL apresentados no capítulo anterior necessitam de um grande número de iterações para convergir. Quanto mais complexo o ambiente, maior o custo computacional para se resolver o problema. Portanto, para melhorar o desempenho dos algoritmos, é necessário o uso de técnicas que acelerem o RL.

Existem diversas maneiras de se acelerar o aprendizado por reforço. Dentre os métodos de aceleração, podemos citar, como sendo os mais comuns: generalização, abstração, reutilização e heurísticas.

A ideia central da aceleração por generalização é espalhar a experiência obtida de uma iteração para estados diferentes do estado da iteração corrente. A generalização pode ser temporal (SUTTON, 1990), onde as experiências são transmitidas para estados visitados a pouco tempo, ou espacial, como proposto por Ribeiro e Szepesvari (1996), quando a experiência é transmitida para vários estados utilizando algum critério de similaridade (conforme será abordado na seção 3.1).

A aceleração por abstração ocorre quando se realiza o aprendizado em um nível mais alto de representação. A abstração pode ser espacial, quando o objetivo é reduzir o tamanho efetivo do problema e a complexidade através da agregação de estados, ou temporal (RIBEIRO, 2000), quando é possível a formação de macroações, ou seja, ações com granularidade menor. A abstração será abordada novamente na seção 3.2.

A aceleração do aprendizado pode ser realizada também por meio da reutilização de partes de soluções anteriores. Este método foi proposto por Drummond (2002) e baseia-se em casos. O método consiste em identificar as subtarefas do problema, pesquisar possíveis soluções na base de casos, resolver e armazenar as modificações propostas nas funções que solucionam as subtarefas novamente na base de casos. Desta forma, depois que as subtarefas de um novo problema são identificadas, o método busca na base de casos as funções que as podem solucionar e as reutiliza, deixando, assim, mais rápido o aprendizado.

Outra maneira de se acelerar o aprendizado é utilizar o conhecimento sobre o domínio para deixar mais rápidos os algoritmos de RL. Isto pode ser realizado através do uso de heurísticas (BIANCHI, 2004), como apresentado na seção 3.3.

3.1 Aceleração por Generalização

3.1.1 Generalização Temporal

Na aceleração por generalização temporal as experiências são transmitidas para estados visitados a pouco tempo.

Alguns dos algoritmos mais comuns que utilizam a aceleração temporal como forma de agilizar o aprendizado por reforço são o $Q(\lambda)$ (WATKINS, 1989), a arquitetura $Dyna$ (SUTTON, 1990) e o $SARSA(\lambda)$ (RUMMERY; NIRANJAN, 1994).

Combinando elegibilidade com o algoritmo *Q-Learning* , o $Q(\lambda)$ inclui as recompensas futuras de um período na atualização do $Q(s_t, a_t)$, através de uma regra de atualização que é aplicada para todos os estados que participaram em um determinado período de tempo. Para tal, o conceito de elegibilidade é aumentado e o par estado-ação é incluído da seguinte maneira:

$$e(u,v) = \begin{cases} \gamma\lambda e(u,v) + 1 & \text{se } u = s_t \text{ e } v = a_t, \\ \gamma\lambda e(u,v) & \text{caso contrário.} \end{cases} \tag{3.1}$$

Onde:

a) $\gamma\lambda$ é o fator para redução da elegibilidade a cada intervalo de tempo, sendo $0 < \lambda < 1$. Quanto maior o λ, maior será a influência das recompensas nos estados futuros.

Outra importante maneira utilizada para aceleração do aprendizado por reforço através do uso da generalização temporal é a arquitetura $Dyna$. Proposta por Sutton (1990), a arquitetura é um método de se encontrar uma política ótima por meio do aprendizado do modelo do ambiente, pois conforme as ações são executadas, o algoritmo aprende iterativamente o modelo da função de transição de estado $T(s_t, a_t, s_{t+1})$ e das recompensas $R(s_t, a_t)$ e usa a experiência adquirida como exemplo para ajustar a política.

O funcionamento do algoritmo $Dyna$ é bastante similar ao do *Q-Learning* , porém, no $Dyna$, além da atualização do $Q(s_t, a_t)$ atual, são atualizados também os $Q's$ adicionais para k estados aleatoriamente escolhidos.

O $SARSA(\lambda)$ é semelhante ao $Q(\lambda)$, porém, nele, a elegibilidade é combinada com o algoritmo $SARSA$ ao invés do *Q-Learning*.

3.1.2 Generalização Espacial

A aceleração por generalização espacial ocorre quando a experiência de uma iteração é transmitida a outros estados considerando algum critério de similaridade.

O algoritmo *Q-Learning* combinado com Espalhamento Espacial (*Q-Learning combined with Spatial Spreading* – QS), proposto por Ribeiro e Szepesvari (1996) realiza esta generalização através do uso do espalhamento espacial como forma de melhorar o desempenho do algoritmo *Q-Learning*. O QS tem como objetivo espalhar a atualização da função $Q(s, a)$ obtida através da ação a realizada no estado s para outros pares estado-ação (x, u) não envolvidos na presente iteração, respeitando alguma medida de similaridade entre estes pares. Este espalhamento é definido através do conhecimento prévio do domínio.

A função de espalhamento é dada por $\sigma(x, u)$, sendo $0 \leqslant \sigma(x, u) \leqslant 1$, e pode ocorrer, por similaridade, no conjunto de estados e no conjunto de ações. Se o espalhamento ocorrer apenas no conjunto de estados, a função é definida conforme equação 3.2. Porém, se ele ocorrer apenas no conjunto de ações a função é definida pela equação 3.3. Há ainda a possibilidade deste mesmo espalhamento ocorrer nos dois conjuntos (estado e ação) ao mesmo tempo. Para esta situação, a função é definida pela equação 3.4.

$$\sigma(s, a, x, u) = g_x(x, s)\delta(u, a) \tag{3.2}$$

$$\sigma(s, a, x, u) = \delta(x, s)g_u(u, a) \tag{3.3}$$

Onde:

a) $g_x(x, s)$ é o grau de similaridade entre os estados, sendo grau 1 para estados x e s muito similares e grau 0 para pouco similares.

b) $g_u(u, a)$ é o grau de similaridade entre as ações, sendo grau 1 para ações a e u muito similares e grau 0 para pouco similares.

c) δ é o Delta de Kronecker, com valor $\delta(i, j) = 1$ se $i = j$ ou $\delta(i, j) = 0$ se $i \neq j$;

$$\sigma(s, a, x, u) = g_{(x,u)}((x, s), (u, a)) \tag{3.4}$$

Onde:

a) $g_{(x,u)}((x, s), (u, a))$ é o grau de similaridade entre o par estado-ação.

O algoritmo QS converge para valores ótimos (RIBEIRO; SZEPESVARI, 1996) desde que a função de espalhamento $\sigma(x, u)$ decaia ao longo do tempo com uma taxa igual ou maior ao do decaimento da taxa de aprendizado α. O QS é apresentado no Algoritmo 3.1.

A partir de somente uma experiência, várias atualizações podem ser realizadas com o uso do algoritmo QS. Porém, a função que expressa a similaridade entre os pares estado-ação precisa ser definida previamente pelo conhecimento de um especialista e, esta definição, pode ser uma tarefa bastante árdua.

Inicialize $Q(s, a)$ arbitrariamente
  Repita
    Visite o estado s
    Selecione uma ação a de acordo com a regra de transição de estados
    Execute a ação a
    Receba o reforço $r(s, a)$ e observe o próximo estado s'
    Para Todos estados x e ações u **faça**
      Atualize os valores de todos os $Q(x, u)$ de acordo com a regra:
      $Q(x, u) \leftarrow Q(x, u) + \sigma(x, u)\alpha[r(s, a) + \gamma \max_{a'} Q(s', a') - Q(x, u)]$
    Atualize o estado $s \leftarrow s'$
  Até que Algum critério de parada seja atingido

  Onde s' é o próximo estado e a' é a ação realizada em s'

Algoritmo 3.1 – QS

Com o intuito de simplificar a inserção de conhecimento prévio sobre o domínio, Pegoraro (2001) propôs o algoritmo QSx , que é, de forma geral, o algoritmo QS com a inserção de informações sobre o domínio no formato de graus de similaridade entre os pares estado-ação e com a inclusão de restrições que podem fazer com que o espalhamento evite, convenientemente, algumas regiões do ambiente. O principal objetivo destas modificações é conseguir modelar o conhecimento existente sobre o ambiente de maneira mais simples e com isso alcançar um espalhamento que seja mais eficiente e contribua positivamente na aceleração do aprendizado.

A relação entre os pares estado-ação (x, u) e (s, a) é dada pela função de espalhamento $\sigma(s, a, x, u)$, sendo $0 \leqslant \sigma(s, a, x, u) \leqslant 1$. A função $\sigma(s, a, x, u)$ tem valores próximos a 1 quando os pares (s, a) e (x, u) são muito similares e tem valores próximos a 0 quando os pares (s, a) e (x, u) são pouco semelhantes.

O algoritmo QSx é apresentado no algoritmo 3.2 e a função de espalhamento que define a similaridade entre os pares estado-ação pode ser determinada conforme equação 3.5.

$$\sigma(s, a, x, u) = \tau^n \tag{3.5}$$

Onde:

a) n é o grau de diversidade entre o par estado-ação;

b) τ é o fator de similaridade entre o par estado-ação, sendo $0 \leqslant \tau \leqslant 1$.

A similaridade entre os pares estado-ação pode ser definida de várias formas: como por conectividade, por vizinhança e por simetria. É possível também que a similaridade não esteja presente por toda a extensão do ambiente. Assim, para não se perder as vantagens obtidas

Inicialize $Q(s, a)$ arbitrariamente
  Repita
    Visite o estado s
    Selecione uma ação a de acordo com a regra de transição de estados
    Execute a ação a
    Receba o reforço $r(s, a)$ e observe o próximo estado s'
    Para Todos estados x e ações u **faça**
      Atualize os valores de todos os $Q(x, u)$, se $\sigma(s, a, x, u) \neq 0$ de acordo com a regra:
      $Q(x, u) \leftarrow Q(x, u) + \sigma(s, a, x, u)\alpha[r(s, a) + \gamma \max_{a'} Q(s', a') - Q(x, u)]$
    Atualize o estado $s \leftarrow s'$
  Até que Algum critério de parada seja atingido

  Onde s' é o próximo estado e a' é a ação realizada em s'

Algoritmo 3.2 – QSx (PEGORARO, 2001)

pelo seu uso, podemos restringir a similaridade a apenas uma região do domínio (PEGORARO, 2001).

A similaridade por conectividade pode ser definida através do número de ações necessárias para levar o agente do estado s para o estado x. Desta forma, quanto menor for o número de ações necessárias para ir de um estado ao outro, maior será a similaridade entre os estados.

É possível implementar a função de espalhamento de duas maneiras: através da consideração de que o grau de diversidade mede a similaridade entre dois estados para uma mesma ação (figura 3.1) e através da consideração de que o grau de diversidade mede a similaridade entre dois pares estado-ação. No primeiro caso, a similaridade é definida como $f_x(x, s) = \tau^{n_a}$, onde na é o número mínimo de ações para levar o agente do estado s ao estado x. Já no segundo caso, a similaridade pode ser encontrada quando o par estado-ação (x, u) é similar ao par (s, a), considerando que o grau de diversidade entre os pares estado-ação é a quantidade mínima de ações para ir do estado supostamente similar x até o estado atual s (PEGORARO, 2001).

O exemplo da figura 3.1 mostra que o grau de diversidade é igual a 1 entre os estados s e s_1, uma vez que o agente robótico R precisa de somente uma ação a_1 para sair do estado s e chegar ao estado s_1, e igual a 2 entre os estados s e s_2, pois o mesmo agente precisa de duas ações a_1 e a_2 para sair de s e chegar a s_2.

Para que esta similaridade funcione como uma maneira de acelerar o aprendizado por reforço, é necessário que um especialista informe as relações de conectividade presentes no domínio. Estas relações podem ser obtidas automaticamente durante o processo de aprendizagem, mas, como as similaridades são mais importantes no início do aprendizado, praticamente não aconteceria uma aceleração do sistema.

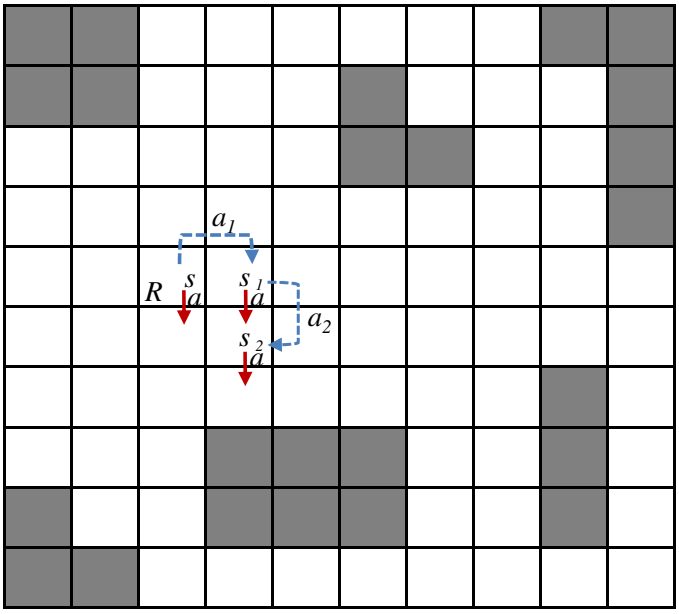


Figura 3.1 – Similaridade por conectividade entre estados. As setas tracejadas indicam as ações que poderiam ser realizadas para se executar a transição entre estados e as setas contínuas indicam as ações realmente executadas.
Fonte: Autor “adaptado de” Pegoraro (2001)

A similaridade por vizinhança acontece, principalmente, no domínio da robótica móvel, pois, normalmente, considera-se que a posição espacial do robô representa o estado. Então, posições topologicamente próximas no ambiente podem ser consideradas similares, mesmo quando não estão diretamente conectadas. A hipótese é válida porque existe uma continuidade espacial quando um robô se movimenta no ambiente.

Nessa situação, o grau de diversidade pode ser atribuído de acordo com a distância entre as posições / estados. Estados próximos podem receber grau de diversidade 1, já estados afastados podem receber graus maiores que 1. Assim, a função de similaridade é definida como: $f_x(x, s) = \tau^{d_{s,x}}$, onde $d_{s,x}$ representa a distância entre o estado s e o similar x (PEGORARO, 2001).

O exemplo apresentado na figura 3.2 mostra que uma ação a executada no espaço s tem efeito similar à mesma ação executada em s_1 e s_2. O grau de diversidade entre s e s_1 é 1 e entre s e s_2 é 2, devido ao aumento da distância.

Na similaridade por simetria, uma experiência realizada em um estado s pode também ter o seu aprendizado refletido no estado similar x, desde que este seja simétrico ao primeiro. Quando os estados são perfeitamente simétricos, atribui-se grau de diversidade igual a 0, pois

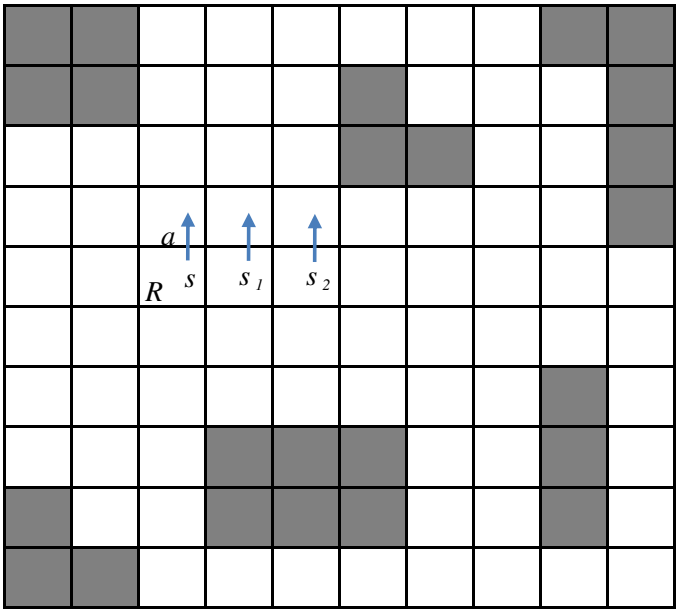


Figura 3.2 – Similaridade por vizinhança entre estados.
Fonte: Autor “adaptado de” Pegoraro (2001)

isto indica que a experiência realizada no estado atual s pode ser totalmente aproveitada pelo estado similar x.

O exemplo mostrado na figura 3.3 indica que as ações a_1 e a_2 geram experiências muito similares, o que permite que o aprendizado seja compartilhado entre os estados s_1 e s_2. Já as ações a_3 e a_4, apesar de criarem experiências similares, necessitam de certo cuidado no compartilhamento do aprendizado, pois as ações são opostas.

A identificação de ações opostas ou semelhantes deve ser obtida através das informações previamente introduzidas na função de espalhamento e depende do domínio. A função de espalhamento deve ser definida por uma função que relaciona dois pares estado-ação simétricos, da seguinte forma: $\sigma(s, a, x, u) = f_{(x,u)}((x, u)(s, a))$ (PEGORARO, 2001).

Em alguns casos a similaridade pode não ser válida para toda a extensão do domínio, portanto torna-se necessária a definição de um método que possa restringir, de maneira conveniente, a similaridade a apenas algumas regiões do ambiente.

Para incluir a restrição é necessário, primeiramente, identificar as possíveis similaridades factíveis dentro do domínio e, então, aplicar as restrições onde for conveniente. A inclusão

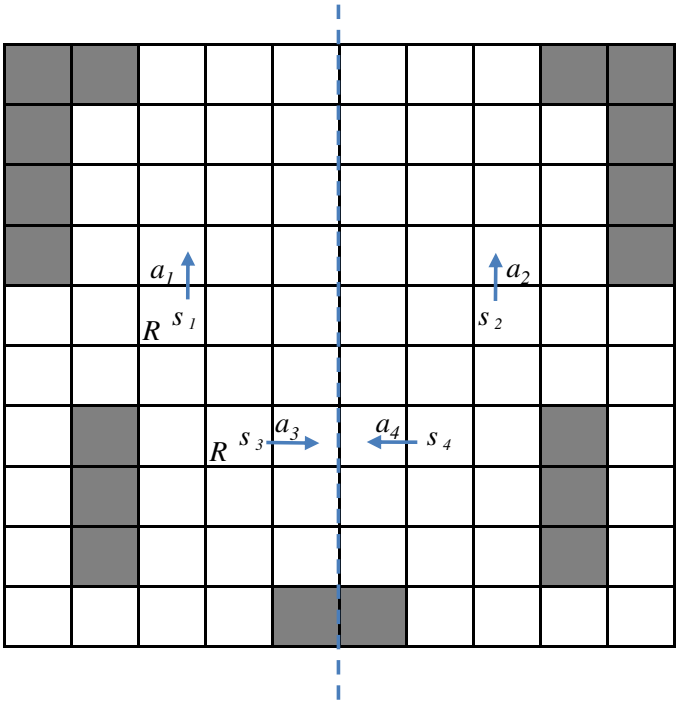


Figura 3.3 – Similaridade por simetria.
Fonte: Autor “adaptado de” Pegoraro (2001)

das restrições deve ser feita na função de espalhamento, que deve retornar zero para as situações que não devem participar do processo de espalhamento.

As restrições aplicadas às similaridades podem ser realizadas através de um conjunto de ações, nas quais as restrições são referentes ao comportamento das ações no ambiente, por um conjunto de estados, em que as similaridades estão restritas a um subconjunto de estados, e pela direção do espaço de estados.

A escolha das restrições aplicadas é muito importante e pode ajudar bastante na criação de uma função de espalhamento eficaz e simples.

3.2 Aceleração por Abstração

3.2.1 Abstração Temporal

Na abstração temporal, o aprendizado utiliza macroações (ações mais abstratas) para diminuir a complexidade de um problema. Isto pode ser feito através da definição de macro-ações, que reúnem as tarefas de muitas ações em uma só. Como exemplo, no domínio da robótica móvel, ao invés de um robô, que pretende sair de um labirinto, tomar ações de baixo

nível, do tipo “vire para direita” ou “vire para esquerda”, ele pode tomar ações mais abstratas, como “saia do labirinto”.

Pode-se notar que ao se aumentar o nível de aprendizado, ou seja, aumentar a abstração, as ações tornam-se, consecutivamente, mais complexas.

3.2.2 Abstração Espacial

Abstração espacial é a capacidade de agregar estados para formar regiões maiores. Ela é utilizada quando é necessário representar a função valor em domínios com o espaço de estados muito grande.

Normalmente utilizam-se aproximadores de funções, como Redes Neurais Artificiais (RNA) e *Cerebellar Model Articulation Controller* (CMAC) (ALBUS, 1975), para realizar a abstração espacial.

A RNA pode ser usada para representar a função valor V através do uso de Perceptrons de Multi-Camadas. A rede pode ser treinada com alguns exemplos e aproximar a função valor V. Esta técnica é bastante usada pelos pesquisadores de Aprendizado por Reforço.

O outro aproximador utilizado é o CMAC, proposto por Albus (1975) como um modelo funcional do cerebelo que pode aproximar funções. O CMAC é constituído por um conjunto de entradas finitas de valores inteiros que se sobrepõem. Estas entradas são chamadas de telhas e a sobreposição destas telhas geram um vetor de entradas. Estas telhas são dispostas, normalmente, em camadas deslocadas em relação às outras, com um intervalo fixo entre elas, e isso permite que o espaço de estados seja particionado em pequenas regiões.

3.3 Uso de heurísticas para aceleração do aprendizado por reforço

O uso de heurísticas pode ser bastante interessante para acelerar o aprendizado por reforço. Segundo Russell e Norvig (2004), a Heurística é uma técnica que melhora a eficiência na solução de problemas, sendo, desta maneira, uma forma de se generalizar o conhecimento do domínio.

Os algoritmos de aceleração do aprendizado através de heurísticas podem ser definidos como uma maneira de se resolver um problema MDP utilizando a função heurística para influenciar, durante o aprendizado, a seleção das ações.

A classe de algoritmos denominada como Aprendizado por Reforço Acelerado por Heurísticas (*Heuristically Accelerated Reinforcement Learning* –HARL) proposta por Bianchi (2004) pode tornar mais eficientes os algoritmos existentes através da limitação do espaço de busca que o agente deve explorar.

Na aceleração do aprendizado por reforço, a função heurística $H_t(s_t, a_t)$ estabelece o quanto a ação a_t é desejada no estado s_t, ou seja, através da função heurística é possível direcionar o agente até o seu objetivo selecionando a ação a_t.

As heurísticas podem ser definidas através de diversas maneiras, como, por exemplo, pela utilização de métodos para extrair o conhecimento sobre o domínio, denominado de "Heurística a partir de X "(BIANCHI, 2004) ou através do conhecimento *a priori (ad hoc)* de um especialista sobre o domínio.

Uma importante característica encontrada no uso de heurísticas é que, se ela for utilizada adequadamente, poderá acelerar o aprendizado, porém se não for, causará um atraso na resolução do sistema, mas não o impedirá de convergir para um estado ótimo (BIANCHI, 2004).

3.3.1 A função heurística

A função heurística pode ser usada como uma maneira de se acelerar o aprendizado através de um conhecimento prévio sobre a política de ações. Ela é inserida na regra de transição de estados para contribuir na escolha da ação a_t que deve ser tomada em s_t.

Uma estratégia que pode ser utilizada para demonstrar a influência da função heurística na escolha das ações é a exploração aleatória ε – Greedy.

Desta maneira, a regra de transição de estados com heurística é dada por:

$$\pi(s_t) = \begin{cases} a_{random} & \text{se q} \leqslant \varepsilon, \\ arg\ max_{a_t}\ [F_t(s_t, a_t) \bowtie \xi H_t(s_t, a_t)^{\beta}] & \text{caso contrário.} \end{cases} \tag{3.6}$$

Onde:

a) $\mathcal{F} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ é uma estimativa de uma função valor que descreve a recompensa esperada acumulada;

b) $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ é a função heurística que influencia a escolha da ação. $H_t(s_t, a_t)$ define a importância de se executar a ação a_t estando no estado s_t;

c) $\bowtie$ é uma função matemática que deve operar sobre números reais e produzir um valor pertencente a um conjunto ordenado;

d) ξ e β são variáveis reais usadas para ponderar a influência da ação heurística;

e) q é um valor escolhido de maneira aleatória com distribuição de probabilidade uniforme sobre [0,1] e ε ($0 \leqslant \varepsilon \leqslant 1$) é o parâmetro que define a taxa de exploração/explotação: quanto menor o valor de ε, menor a probabilidade de se fazer uma escolha aleatória;

f) a_{random} é uma ação selecionada de maneira aleatória entre as ações executáveis no estado s_t.

A convergência desta formulação e a validade das provas pertinentes aos algoritmos de RL são garantidas por Bianchi (2004), desde que os limites máximos e mínimos dos valores da função heurística ($h_{min} \leqslant H(s,a) \leqslant h_{max}$) sejam corretamente estipulados.

Bianchi (2004) determina que $h_{min} = 0$ e que h_{max} deve seguir a seguinte equação:

$$h_{max} \leqslant \left| \frac{r_{min}}{1-\gamma} \right| \tag{3.7}$$

Caso $|r_{min}| > r_{max}$, h_{max} deve também obedecer a equação:

$$h_{max} \leqslant \frac{r_{max}}{1-\gamma} \tag{3.8}$$

Onde r_{min} e r_{max} representam a menor e a maior recompensa estabelecida, respectivamente.

3.3.2 *Q-Learning* Acelerado por Heurísticas

Podem-se criar algoritmos da classe HARL a partir de qualquer algoritmo de aprendizado por reforço. Um algoritmo interessante, apresentado por Bianchi (2004), é o "*Q-Learning* Acelerado por Heurísticas" (*Heuristically Accelerated Q-Learning* – HAQL), que é uma extensão do algoritmo *Q-Learning* que permite o uso de heurísticas para indicar o melhor caminho para um agente aprendiz em cada estado do ambiente.

O HAQL utiliza a função heurística H em conjunto com a função $Q_t(s_t, a_t)$. Logo, o HAQL é aplicado através da modificação da regra de transição ε – Greedy padrão presente no algoritmo *Q-Learning*.

Desta maneira, a transição de estados do HAQL é dada por:

$$\pi(s_t) = \begin{cases} a_{random} & \text{se q} \leqslant \varepsilon, \\ arg\ max_{a_t}\ [Q_t(s_t,a_t) + \xi H_t(s_t,a_t)] & \text{caso contrário.} \end{cases} \tag{3.9}$$

O algoritmo HAQL tem a regra de atualização semelhante à regra do *Q-Learning* convencional, conforme pode ser notado no algoritmo 3.3.

Diversos trabalhos têm sido desenvolvidos seguindo esta linha de estudo, como, por exemplo, o trabalho apresentado por Celiberto Jr. (2007) que propõe o uso das heurísticas como uma forma de se acelerar o aprendizado por reforço no domínio do futebol de robôs simulados, Martins (2007) que também propõe o uso das heurísticas para acelerar o aprendizado, mas no

Inicialize $Q(s, a)$ arbitrariamente
  Repita
    Visite o estado s
    Selecione uma ação a de acordo com a regra de transição de estado
    Execute a ação a
    Receba o reforço $r(s, a)$ e observe o próximo estado s'
    Atualize os valores de $H(s, a)$ utilizando um método "H a partir de X"
    Atualize os valores de $Q(s, a)$ de acordo com regra de atualização:
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma \max a' Q(s', a') - Q(s, a)]$
    Atualize o estado $s \leftarrow s'$
  Até que Algum critério de parada seja atingido

  Onde s' é o próximo estado e a' é a ação realizada em s'

Algoritmo 3.3 – HAQL (BIANCHI, 2004)

domínio do futebol de robôs no mundo real e Bianchi, Ros e Mantaras (2009), com o estudo do uso de uma base de casos como heurística para acelerar o RL.

3.4 Considerações finais

Este capítulo apresentou um resumo sobre a Aceleração do Aprendizado por Reforço. Contudo, é preciso salientar que existem outras maneiras de se acelerar o RL. Os métodos que foram apresentados aqui referem-se aos que, de alguma maneira, influenciaram este trabalho. Maior atenção foi dada à Generalização Espacial e ao Uso de Heurísticas como forma de acelerar o aprendizado, pois estes dois trabalhos foram os que tiveram maior peso na proposta desta dissertação.

O próximo capítulo aborda um diferente tipo de aprendizado, que utiliza a demonstração como base para o desenvolvimento de políticas. Este aprendizado também foi utilizado como forma de inspiração para o desdobramento da proposta deste trabalho.

4 APRENDIZADO POR DEMONSTRAÇÃO

O Aprendizado por Demonstração (*Learning from Demonstration* – LfD) (SCHAAL, 1997), que também é conhecido como "Aprendizado por Observação" (*Learning by Watching*) (KUNIYOSHI et al., 1994), "Programação por Demonstração" (*Programming by Demonstration*), "Aprendizado por Imitação" (*Imitation Learning*) e "Ensino por Exposição" (*Teaching by Showing*), pode ser considerado um método supervisionado de aprendizado que consiste na técnica de se desenvolver políticas através de exemplos de sequências de estado-ação, que são gravados durante a demonstração. Este aprendizado necessita de um professor, ou seja, alguém precisa transmitir conhecimento ao agente para que este possa aprender a partir de alguns comportamentos previamente demonstrados.

Segundo Argall et al. (2009), o LfD tem sido aplicado, principalmente, a uma grande variedade de problemas no domínio da robótica móvel, contudo as discrepâncias entre as pesquisas ainda são grandes, devido ao fato dos detalhes de implementação no mundo real serem muito diferentes de pesquisador para pesquisador. Como aspectos comuns a todos os métodos de implementação do Aprendizado por Demonstração, pode-se citar a necessidade de um professor demonstrando ao agente o comportamento esperado e o fato de o agente usar destas demonstrações para obter uma política capaz de reproduzir este comportamento.

O principal desafio desta técnica é alcançar uma política de ações por meio de exemplos que podem não ser exatamente iguais às tarefas que o agente deverá realizar. Além disso, o aprendizado deve ser realizado com a menor quantidade possível de demonstrações.

O primeiro estágio realizado no Aprendizado por Demonstração é a execução dos exemplos por um professor. Nesta fase, o agente aprende, por meio de dados coletados do ambiente e dos pares estado-ação fornecidos, uma função de aproximação para encontrar a política esperada. Passada a fase de demonstração, o controle do agente robótico será feito pela política obtida. A figura 4.1 mostra o funcionamento básico do LfD, onde π representa a política, s' o estado observado e a' a ação, definida pela política, a ser tomada em s'.

O desenvolvimento de um sistema que utiliza demonstrações para aprender pode ser bastante complexo, devido ao grande número de possíveis variáveis existentes. Normalmente, para diminuir a dificuldade, o LfD requer que diversas escolhas sejam realizadas previamente por um especialista. Algumas destas escolhas são mais evidentes, pois devem ser feitas de acordo com o domínio da aplicação, mas outras, como a definição do professor (4.1) e a definição da técnica de demonstração (4.3) que será utilizada, requerem mais cuidados e devem ser muito bem elaboradas, pois exercem grande influência na maneira como o aprendizado acontecerá.

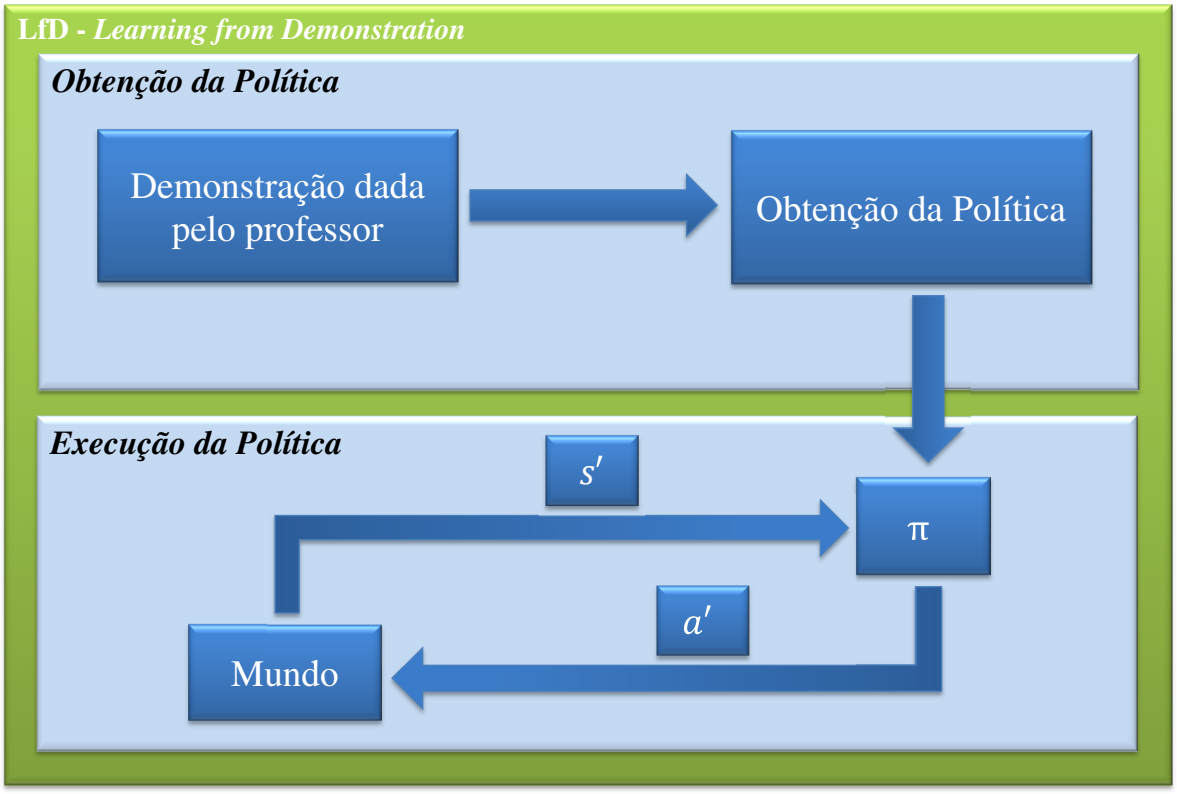


Figura 4.1 – Esquema de funcionamento do Aprendizado por Demonstração.
Fonte: Autor "adaptado de" Argall et al. (2009)

4.1 Definição do professor

No LfD, os seres humanos são, na maior parte do tempo, os professores escolhidos para exemplificar a política ao agente. No entanto, é possível também utilizar robôs, políticas previamente incluídas no algoritmo e planejamento simulado como forma de mostrar exemplos ao agente aprendiz (ARGALL et al., 2009).

A escolha do professor pode mudar a técnica utilizada para obtenção dos dados durante a demonstração, logo, é de extrema importância ter esta definição antes de iniciar o estudo das possíveis técnicas de LfD.

4.2 O problema de correspondência

Um dos desafios presentes no LfD é referente à transformação dos dados coletados durante as demonstrações em dados que realmente sejam úteis ao agente aprendiz, pois o mapeamento dos estados-ações nem sempre é feito de forma direta, uma vez que o professor que

executa as demonstrações pode diferir do aprendiz em seu sistema sensorial ou mecânico. Estas diferenças entre sistemas tornam necessária a identificação de um mapeamento que permita a transferência de informação de um agente para o outro. Por exemplo, a câmera de um robô aprendiz não pode detectar a mudança de estados da mesma maneira que o olho de um professor humano (ARGALL et al., 2009). Este assunto é conhecido como problema de correspondência (NEHANIV; DAUTENHAHN, 2002).

Argall et al. (2009) define a correspondência em relação a dois tipos de mapeamento: registro e personificação. O mapeamento por registro é necessário quando é preciso criar algum tipo de codificação entre os estados-ações vivenciados pelo professor durante as demonstrações e os estados-ações que devem ser gravados no conjunto de dados. Já o mapeamento por personificação é aquele que deve ser realizado quando é necessária a criação de uma codificação entre os pares estado-ação gravados no conjunto de dados e os pares estado-ação que o agente deve observar/executar.

4.3 Técnicas para obtenção de dados

Dependendo do modo como o mapeamento do domínio é feito, baseado na presença ou ausência do mapeamento por personificação, o LfD pode ser dividido em Demonstração (seção 4.3.1) e Imitação (seção 4.3.2). Na Demonstração o mapeamento é feito diretamente pelos dados coletados na plataforma do próprio agente aprendiz, logo, não há a necessidade do mapeamento por personificação. Já na Imitação os dados são coletados em uma plataforma diferente da plataforma do aprendiz, portanto, o mapeamento por personificação é necessário (ARGALL et al., 2009).

Tanto a Demonstração quanto a Imitação podem ter, cada uma, duas diferentes abordagens. A Demonstração pode ser realizada por Teleoperação e por Sombreamento. Enquanto a Imitação pode ser feita por meio de sensores colocados diretamente no professor ou no ambiente externo.

4.3.1 Demonstração

A teleoperação é a técnica pelo qual o professor opera diretamente o agente aprendiz e, este, grava esta execução com o uso de seus próprios sensores.

Enquanto a teleoperação acontece, o agente grava as informações pertinentes aos pares estado-ação e, desta forma, cria o mapeamento do domínio. Este método é a maneira mais imediata de se transferir conhecimento ao agente, porém, para aplicá-lo, é necessário que o sistema seja facilmente controlável.

Os seres humanos podem teleoperar um agente robótico por intermédio de diversas maneiras, sendo que a ferramenta mais comum é o uso de um *joystick*. A teleoperação pode também ser executada com o uso da voz e com o uso de políticas previamente incluídas no algoritmo do robô.

Na técnica de sombreamento o agente robótico aprende por intermédio das informações recebidas diretamente em seus sensores durante a execução da tarefa, enquanto repete os movimentos de um professor. Nesta técnica, o mapeamento não é feito diretamente, pois os pares estado-ação da demonstração verdadeira não são os dados gravados. As informações gravadas são obtidas indiretamente por meio dos registros realizados pelos sensores do agente aprendiz as passo que ele reproduz a tarefa que está sendo realizada pelo professor.

O sombreamento requer um algoritmo adicional para realizar a observação dos movimentos do professor.

4.3.2 Imitação

Ao contrário da teleoperação e do sombreamento, a imitação é a técnica pela qual o robô aprendiz não se utiliza de seus próprios sensores para aprender. Ele pode aprender mediante a utilização de dados coletados por sensores presentes no professor ou por sensores presentes no ambiente.

Quando os sensores estão colocados no professor, a coleta de dados acontece diretamente na plataforma de execução. Já com os sensores presentes no ambiente, a coleta de dados é indireta e, normalmente, é baseada em visão, que é um método que apresenta mais incerteza e, portanto, é menos confiável e menos preciso do que o anterior.

Tanto para sensores diretamente incluídos no professor quanto para sensores extracorporais ao agente, os seres humanos são, novamente, os professores mais usuais.

4.4 Técnicas para a obtenção da política

No LfD, existem diversas técnicas para a obtenção da política mediante a análise do conjunto de exemplos. Segundo Argall et al. (2009), os três métodos mais importantes são conhecidos como: função de mapeamento, modelos de sistema e planejamento.

4.4.1 Função de mapeamento

A função de mapeamento tem como objetivo tentar reproduzir a política ensinada pelo professor, de maneira generalizada, a partir dos exemplos obtidos nos treinamentos. Assim, a

função de mapeamento deve ser capaz de encontrar soluções válidas para todos os estados do domínio, mesmo para aqueles que não foram demonstrados durante a fase de demonstração.

Como é possível a produção de saídas discretas ou contínuas pelo algoritmo, normalmente, utilizam-se duas técnicas para alcançar a função de aproximação (ARGALL et al., 2009): a classificação, para saídas discretas, e a regressão, para saídas contínuas. Ambas as técnicas são descritas por Hastie, Tibshirani e Friedman (2001) como formas de aprendizagem inclusas no arcabouço estatístico, porém elas também são bastante difundidas para outras aplicações.

No contexto do aprendizado de uma política, a classificação pode usar, como dados de entrada, os estados s do robô e, como dados de saída, as ações discretas a. Esta técnica pode ser aplicada a qualquer nível de aprendizado de ações em um robô, desde as mais básicas, como andar e virar, até as mais complexas, como o controle do comportamento de um humanoide.

Cada nível de aprendizado requer uma técnica de classificação diferente, dependendo do tipo e complexidade das ações. As técnicas mais usais de classificação para o LfD são as árvores de decisão, os modelos gaussianos misturados, as redes bayesianas e o modelo dos vizinhos próximos (HASTIE; TIBSHIRANI; FRIEDMAN, 2001).

A regressão usa, de forma similar a classificação, os estados s como forma de entrada e as ações contínuas a como dado de saída. Contudo, a regressão não deve ser usada para qualquer nível de aprendizado. Por ser um processo contínuo que precisa coletar dados de diversas demonstrações, a regressão funciona melhor com o aprendizado de movimentos básicos do que com o de comportamentos mais complexos.

4.4.2 Modelos de sistema

Nesse método o LfD usa a transição de estados $T(s'|s, a)$ como modelo de mundo para aprender a política. Este modelo já foi apresentado no capítulo 2, pois esta abordagem é tipicamente usada por métodos de Aprendizado por Reforço.

No LfD, a transição de estados é criada a partir dos exemplos obtidos nas demonstrações e, então, a política é derivada com o auxílio de uma função de recompensas $R(s)$, que pode ser aprendida durante a demonstração ou ser determinada pelo próprio usuário.

A seção 4.6 mostrará um exemplo de como usar o Aprendizado por Reforço em conjunto com o Aprendizado por Demonstração.

4.4.3 Planejamento

Outro método para se obter uma política pode ser o planejamento. Nesta técnica, um plano, contendo a política de ações que conduzem o robô do estado inicial até o final, é diretamente inserido no algoritmo do agente. Contudo, o bom funcionamento deste método requer mais do que simplesmente demonstrações. Normalmente, são necessárias a inclusão de condições, intenções e anotações do professor para fazer com que o agente possa aprender.

A adição destes dados complementares pode incluir informações sobre o objetivo final e sobre o tempo de execução esperado para o comportamento específico e, também, fornecer retornos binários que aumentam ou diminuem a quantidade de ações necessárias para se atingir a meta final.

4.5 Limitações do Aprendizado por Demonstração

O LfD é extremamente limitado pela qualidade dos dados provenientes da fase de demonstração, pois ele é totalmente dependente das informações obtidas neste período. Segundo Argall et al. (2009), são dois os principais motivos que prejudicam o desempenho de um agente aprendiz que utiliza o Aprendizado por Demonstração: a dispersão dos dados, ou seja, existência de áreas com estados não demonstrados, e a baixa qualidade dos exemplos realizados pelo professor.

Dados dispersos podem ser problemáticos para o LfD, pois, num domínio em que a quantidade de estados é muito grande, a probabilidade do professor não ser capaz de exibir a ação correta para cada estado discreto é muito maior do que quando comparado a um ambiente com poucos estados. Duas técnicas podem ser utilizadas para diminuir ou eliminar este problema (ARGALL et al., 2009). Uma delas é a Generalização, que pode ser realizada por intermédio de técnicas como a classificação e a regressão (seção 4.4.1) e visa a obtenção de uma função de aproximação que consiga generalizar os estados não visitados nas demonstrações realizadas. A outra maneira trabalha com a inclusão de novas demonstrações para cobrir os estados que não foram visitados.

A baixa qualidade das demonstrações realizadas pelo professor afeta, de maneira considerável, o Aprendizado por Demonstração. Esta falha pode acontecer devido à falta de habilidade ou de conhecimento do professor para desempenhar a tarefa proposta de maneira ótima. Assim, para solucionar este problema, pode-se excluir da demonstração os exemplos que não são considerados ótimos ou fazer com que o agente aprendiz receba algum retorno que avalie o seu desempenho.

A exclusão de exemplos considerados não ótimos pode ser feita, por exemplo, por meio do uso de repetidas demonstrações, das quais se obtenha alguma média ou generalização que resulte em ações que se aproximem das ações ótimas. Já o uso de uma avaliação de desempenho pode ser possível, por exemplo, por meio do recebimento de recompensas, conforme realizado no Aprendizado por Reforço, como descrito na próxima seção.

4.6 O aprendizado dividido em duas fases

Um método interessante de diminuir as limitações do LfD, proposto por Smart e Kaelbling (2002), possibilita a junção do Aprendizado por Demonstração com o Aprendizado por Reforço. Este método trabalha em duas fases: a primeira utiliza a Demonstração e a segunda a Exploração. Na Demonstração, a função $Q(s, a)$ é atualizada enquanto os exemplos são demonstrados ao agente. Após a realização destas demonstrações, o agente inicia a Exploração por meio do RL utilizando uma função $Q(s, a)$ com valores predefinidos, que representam o conhecimento sobre o domínio obtido na primeira fase.

Além de acelerar o aprendizado, este método apresenta a vantagem de tornar desnecessária a obrigação de se conhecer em detalhes os sensores e os atuadores do robô, pois, desta forma, o agente aprende por meio de dados empíricos e o sistema não é exposto às tendências pessoais do programador (SMART; KAELBLING, 2002).

Contudo, como a Demonstração atualiza diretamente a função $Q(s, a)$, a primeira fase deve conter um número de demonstrações suficientemente grande para que a função fique forte o bastante para influenciar o aprendizado do robô. Como a função Q é iterativa, normalmente são necessárias várias demonstrações para que o desempenho do agente melhore de maneira considerável.

4.7 Considerações finais

O objetivo deste capítulo foi fornecer uma breve revisão bibliográfica do Aprendizado por Demonstração, que é um modelo de aprendizado que trabalha com um método intuitivo de comunicação entre seres humanos e robôs, eliminando, muitas vezes, a necessidade do controlador humano ter que conhecer em detalhes os sensores, atuadores, cinemática inversa ou, até mesmo, programação de máquina.

Por outro lado, o LfD pode ser fortemente limitado pela qualidade dos dados recebidos na fase das demonstrações, como, por exemplo, demontrações com dados dispersos e com baixa qualidade, que podem resultar em casos nas quais a política ótima de atuação não seja encon-

trada. Por isso, é interessante usar o LfD combinado com outras técnicas de aprendizagem, garantindo assim a convergência do aprendizado.

Desta forma, o próximo capítulo visa apresentar a proposta deste trabalho, que utiliza conceitos presentes no LfD combinados com técnicas conhecidas de RL para tornar mais rápida e intuitiva a aprendizagem de robôs móveis autônomos em ambientes fechados.

5 USO DE HEURÍSTICAS OBTIDAS POR MEIO DE DEMONSTRAÇÕES PARA ACELERAÇÃO DO APRENDIZADO POR REFORÇO

Com o intuito de exibir o cenário no qual este trabalho está inserido, foi realizada, nos capítulos anteriores, uma revisão bibliográfica sobre as seguintes técnicas: Aprendizado por Reforço, Aceleração do Aprendizado por Reforço e Aprendizado por Demonstração.

Conforme já descrito no capítulo 2, o Aprendizado por Reforço tem sido apresentado como uma das técnicas muito utilizadas para realizar o aprendizado em robôs, devido a sua capacidade de interação com o ambiente. Porém, conforme abordado nos capítulos iniciais, o RL é muito lento e precisa ser acelerado. Contudo, a implementação pura de uma das técnicas apresentadas no capítulo 3 pode não ser tão eficaz quando o problema é levado para um domínio com agentes dinâmicos e ambientes complexos e extensos, devido ao grande número de variáveis. Um problema similar pode ser encontrado no Aprendizado por Demonstração, uma vez que, depois que a política é derivada das demonstrações, ela não muda mais, a não ser que mais demonstrações sejam realizadas. Isto pode tornar o sistema ineficaz em um ambiente que está sujeito a sofrer alterações e também pode fazer com que uma política ótima de atuação não seja encontrada.

Logo, a proposta desta dissertação tem por objetivo encontrar uma classe de algoritmos que possa manter as propriedades do Aprendizado por Reforço, como a capacidade de exploração, mas que seja mais rápida e eficaz para o domínio proposto. Um dos métodos conhecidos para acelerar o RL é o realizado por intermédio de heurísticas (BIANCHI, 2004)(seção 3.3), que tem como uma das maiores vantagens o aumento na velocidade do aprendizado mantendo todas as propriedades do RL. No entanto, esta técnica tem a desvantagem de não apresentar um estudo completo sobre a aquisição das heurísticas em ambientes mais complexos e dinâmicos, como é o caso da robótica móvel autonôma em ambientes reais.

Outro algoritmo que também utiliza o RL como base de aprendizado no domínio da navegação robótica, é o que trabalha em duas fases, combinando o LfD com o RL (SMART; KAELBLING, 2002), conforme citado no capítulo anterior (seção 4.6). Este método tem a vantagem de eliminar a necessidade do programador conhecer em detalhes o funcionamento dos sensores e atuadores do robô, porém este algoritmo requer um grande número de demonstrações, principalmente em ambientes maiores e mais complexos, para tornar a função $Q(s, a)$ forte o bastante para exercer alguma influência no aprendizado.

Neste contexto, propõe-se a fusão entre o Aprendizado por Reforço Acelerado por Heurísticas (BIANCHI, 2004) e o Aprendizado por Demonstração (SCHAAL, 1997). Inspirado pelo aprendizado em duas fases (SMART; KAELBLING, 2002), esta junção pode ser feita me-

diante o uso da demonstração como método para obtenção das heurísticas ao invés da função Q. Desta forma, a heurística pode ajudar a acelerar o RL e o algoritmo mantém as propriedades do RL e as vantagens do LfD. Esta junção de técnicas é aqui denominada "Aprendizado por Reforço Acelerado por Heurísticas obtidas por meio de Demonstrações" (*Heuristics from Demonstration to Accelerate Reinforcement Learning* – HfDARL).

Com o uso do HfDARL o aprendizado é acelerado por intermédio de um conhecimento prévio inserido pelas demonstrações, basicamente como no aprendizado em duas fases (SMART; KAELBLING, 2002). Porém, como as demonstrações são responsáveis pela obtenção das heurísticas, que, principalmente no início do aprendizado, são fortes o bastante para influenciar a escolha das ações a serem tomadas, poucas demonstrações tornam-se necessárias para acelerar consideravelmente o aprendizado desde os primeiros episódios.

5.1 Obtenção das heurísticas

Conforme exposto anteriormente, o intuito deste trabalho é apresentar uma maneira diferente de obtenção das Heurísticas utilizadas para acelerar o RL. O uso da Demonstração pode ser uma maneira eficaz para realizar esta aquisição, pois o aprendizado pode ser simplificado se o agente puder utilizar, como conhecimento prévio, dados empíricos obtidos diretamente do ambiente e da experiência do professor.

Neste modelo, as Heurísticas devem ser criadas, durante a demonstração, por meio da atualização da função H em cada estado-ação percorrido. Desta maneira, para cada ação tomada pelo professor, um valor máximo predeterminado η deve ser atribuído a $H(s, a)$, sendo que $\eta \in \mathbb{R}$ e $0 < \eta \leqslant \infty$.

Como a intenção deste trabalho é conseguir uma melhora significativa no desempenho do agente, mesmo em ambientes grandes e complexos, com o menor número possível de demonstrações, é necessário o uso de uma técnica de generalização para conseguir construir a maior heurística possível com poucos exemplos de caminho.

Conforme apresentada na seção 3.1, a generalização pode ser temporal ou espacial. Como neste trabalho é necessário realizar o espalhamento da função heurística H para outros pares estado-ação que não foram visitados durante a demonstração, a generalização espacial aparece como uma maneira mais eficaz do que a temporal, uma vez que a generalização espacial trabalha justamente com o espalhamento da função enquanto a temporal transmite a experiência obtida para estados visitados recentemente.

Uma maneira de realizar a generalização espacial é com o uso do algoritmo QSx (PEGORARO, 2001). Conforme citado na seção 3.1.2, este algoritmo é uma modificação do algoritmo QS (RIBEIRO; SZEPESVARI, 1996), que insere os conceitos de similaridade entre os

pares estado-ação e restringe o espalhamento a algumas regiões do ambiente. O QSx trabalha diretamente com a atualização da função Q por meio do uso do algoritmo *Q-Learning*.

Como o QSx pode ser uma eficiente técnica para realizar o espalhamento espacial de uma função, a ideia é utilizar alguns conceitos presentes neste algoritmo e adaptá-los, convenientemente, para que seja possível realizar, por meio dele, o espalhamento da função H.

Duas teorias presentes no QSx são de muita importância para a realização do espalhamento da função Heurística neste trabalho. São elas, a similaridade entre os pares estado-ação e a restrição às regiões do ambiente que não devem participar do espalhamento. Segundo Pegoraro (2001), as similaridades podem ser divididas, como apresentado na seção 3.1.2, em três grupos: conectividade, vizinhança e simetria. Como este trabalho objetiva alcançar o aprendizado de um robô autônomo em qualquer ambiente fechado, somente as similaridades por conectividade e vizinhança são plausíveis, uma vez que nem todo domínio pode ser classificado como simétrico.

A similaridade por conectividade usa a quantidade de ações necessárias para se alcançar um estado esperado x, partindo-se de um estado s, como referência de diversidade. Já no conceito de vizinhança, o fator de diversidade é obtido por meio da distância euclidiana entre o estado inicial s e o estado final x.

A similaridade aplicada às heurísticas será medida por meio da função de espalhamento definida como:

$$\sigma_H(s, a, x, u) = \tau^n \tag{5.1}$$

Desta forma, a regra de espalhamento da função H será:

$$H(x, u) = H(s, a)\sigma_H(s, a, x, u) \tag{5.2}$$

Onde:

a) $H(s, a)$ representa o valor máximo predefinido η aplicado como Heurística no estado s para a ação a.

A função de similaridade utilizada, assim como o grau de diversidade n, serão apresentados durante os experimentos, pois eles podem mudar de acordo com a similaridade adotada, conforme abordado na seção 3.1.

O espalhamento do H é restrito para as regiões do ambiente que ainda não têm valor, ou têm um valor menor do que o novo, estabelecido como Heurística no par estado-ação. Assim, a função H não deve ser atualizada se, no par estado-ação referido, já existir um H maior do que o novo valor proposto. Isto é necessário para manter como Heurística principal os valores mais fortes que, consequentemente, devem se aproximar do caminho ótimo.

5.2 O aprendizado

Uma vez que a Heurística é obtida, o aprendizado deve ocorrer por intermédio de um algoritmo de Aprendizado por Reforço. Conforme visto na seção 3.3.2, o "*Q-Learning* Acelerado por Heurísticas" (*Heuristically Accelerated Q-Learning* – HAQL) (BIANCHI, 2004) é uma adaptação do algoritmo *Q-Learning* que permite o uso de heurísticas para indicar o melhor caminho a um agente aprendiz em cada estado do ambiente.

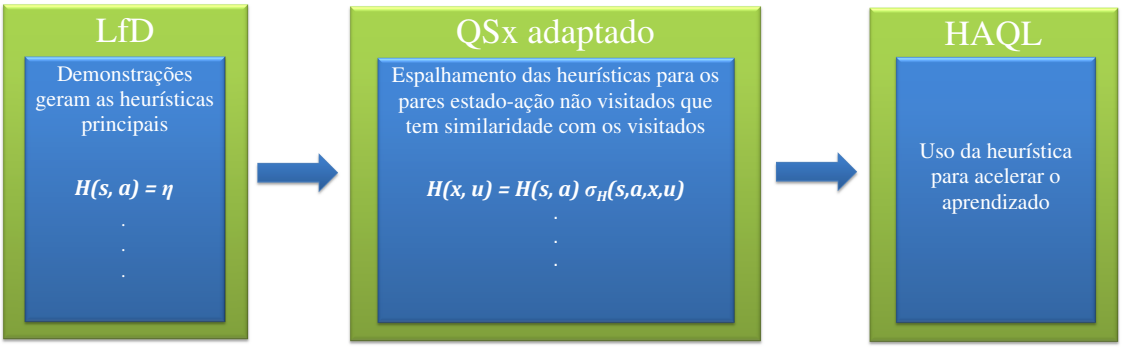


Figura 5.1 – Diagrama de blocos da proposta: Algoritmo HfDAQL.
(s,a) representa o par estado-ação que é diretamente atualizado pela demonstração. (x,u) representa o par estado-ação que não é diretamente atualizado pela demonstração.
Fonte: Autor

Neste trabalho, propõe-se a inclusão de demonstrações no HAQL, criando-se assim o "Q-Learning Acelerado por Heurísticas obtidas por meio de Demonstrações" (*Heuristics from Demonstration to Accelerate Q-Learning* – HfDAQL) que pode ser verificado no algoritmo 5.1. Nota-se que as únicas modificações em comparação com o algoritmo *Q-Learning* proposto por Watkins (1989) são referentes ao uso da função Heurística para escolha da ação a ser tomada e a obtenção da função $H(s, a)$ por meio da demonstração complementada com a Generalização Espacial.

5.3 Considerações finais

Este capítulo descreveu a proposta deste trabalho, apresentando o algoritmo "Q-Learning Acelerado por Heurísticas obtidas por meio de Demonstrações" (*Heuristics from Demonstration to Accelerate Q-Learning* – HfDAQL).

No próximo capítulo são apresentados os experimentos que foram realizados com este algoritmo.

Inicialize $H(s, a) = 0$.
  Repita
    Repita
      Demonstre ao agente aprendiz qual ação a tomar no estado s
      Atualize $H(s, a)$ com o valor máximo η
      Se $H(x, u) < H(x'', u'')$ **Então**
        Espalhe os valores de $H(s, a)$ para os pares estado-ação similares, atualizando $H(x, u)$ de acordo com a regra de espalhamento:
        $H(x, u) \leftarrow H(s, a)\sigma_H(s, a, x, u)$
    Até que que o objetivo seja atingido
  Até que o número total de demonstrações seja atingido
Inicialize $Q(s, a)$ arbitrariamente
  Repita
    Visite o estado s
    Selecione uma ação a de acordo com a regra de transição de estado do HAQL
    Execute a ação a
    Receba o reforço $r(s, a)$ e observe o próximo estado s'
    Atualize os valores de $Q(s, a)$ de acordo com regra de atualização:
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    Atualize o estado $s \leftarrow s'$
  Até que algum critério de parada seja atingido

  Onde s' é o próximo estado, a' é a ação realizada em s', (x, u) representa os pares estado-ação que não estão diretamente envolvidos com a presente demonstração e $H(x'', u'')$ representa os valores de heurística recebidos em demonstrações anteriores para os pares estado-ação (x, u).

Algoritmo 5.1 – HfDAQL - atualização das heurísticas realizada por meio de demonstrações.

6 EXPERIMENTOS E RESULTADOS

Com o objetivo de verificar o comportamento do aprendizado proposto, foram realizados alguns experimentos nos quais o Aprendizado por Reforço Acelerado por Heurísticas obtidas por Demonstrações pôde ser comparado com o RL que utiliza o algoritmo *Q-Learning* de Watkins (1989) e com o RL acelerado por heurísticas definidas com base em um conhecimento *a priori* sobre o domínio (heurística definida *ad hoc*) (BIANCHI, 2004).

Todos os experimentos deste capítulo foram realizados no domínio da navegação robótica, porém dois tipos diferentes de ambientes foram estudados. Inicialmente, fez-se uso do domínio conhecido como *Grid World* - Mundo de Grades, que é um ambiente discretizado em forma de grade por onde o agente deve se mover. Em seguida, o estudo foi realizado em um domínio mais complexo, que simula um robô real se movendo em um ambiente fechado, por intermédio do *software* Player/Stage.

6.1 Grid World - Mundo de Grades

O Mundo de Grades - *Grid World* é um domínio bastante conhecido e constantemente utilizado para os estudos do Aprendizado por Reforço, como pode ser observado nos experimentos de Kaelbling, Cassandra e Kurien (1996), Drummond (2002), Foster e Dayan (2002), Bianchi (2004) e outros.

O ambiente proposto para a realização dos experimentos desta seção é composto por grades com a dimensão total de 10 linhas x 20 colunas, tem obstáculos em seu interior e é totalmente cercado por paredes, exceto no estado meta G. O objetivo do robô é encontrar o estado G, sendo que o estado inicial da busca é escolhido de maneira aleatória. O robô pode executar quatro ações: para cima, para baixo, para esquerda e para direita.

Para a execução dos experimentos foram utilizadas 3 demonstrações de caminhos ótimos. As demonstrações realizadas no experimento foram feitas por teleoperação, sendo que o professor tem a visão global do sistema, e consistem em guiar o robô a partir de uma posição inicial aleatória até o estado meta G. Este procedimento deve ser realizado por um ser humano, mediante o uso das teclas direcionais do teclado. O valor máximo η atribuído para $H(s, a)$ para as experiências realizadas no mundo de grades foi igual a 6,0. Este valor foi escolhido aleatoriamente, respeitando os limites máximos e mínimos da função heurística H, seguindo a equação 3.7.

O espalhamento espacial da função H foi realizado utilizando-se os conceitos de similaridade por conectividade, conforme equação 6.1.

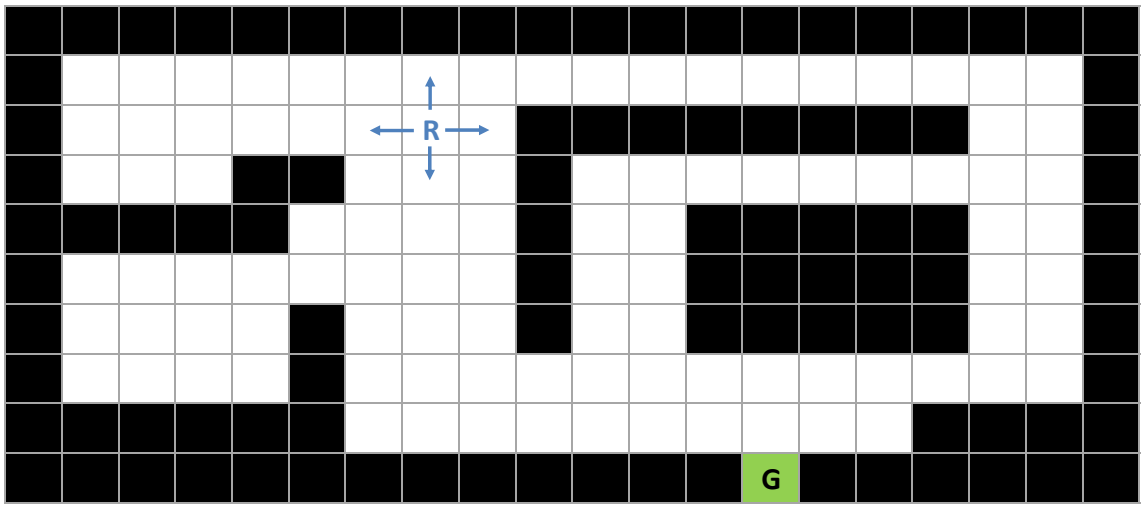


Figura 6.1 – Ambiente por onde o robô R deve aprender a encontrar o objetivo G. Os estados mais escuros representam os obstáculos/paredes.
Fonte: Autor

$$f_x(x, s) = \tau^{n_a} \tag{6.1}$$

Onde:

a) O fator de similaridade τ é igual a $0,5$;

b) O grau de diversidade n_a é igual ao número mínimo de ações para levar o agente do estado s ao x.

Desta forma, a atualização espacial das heurísticas, realizada por intermédio da similaridade, pode ser verificada conforme tabela da figura 6.2.

Após a fase das demonstrações, o aprendizado foi feito com o uso do algoritmo HAQL, considerando os seguintes parâmetros:

a) Taxa de Aprendizado $\alpha : 20\%$;

b) Fator de Desconto $\gamma : 90\%$;

c) Porcentagem de exploração $\varepsilon : 20\%$;

d) Reforço de -1 a cada passo dentro do labirinto que não resultar no encontro da saída;

e) Reforço de -5 se bater em uma parede;

f) Reforço de 1000 quando a saída for encontrada;

H(l-1, c-1 ;a) = η/4 ação: *a*	*H(l-1, c ;a)* = η/2 ação: *a*	*H(l-1, c+1 ;a)* = η/4 ação: *a*
H(l, c-1 ;a) = η/2 ação: *a*	*H(l-1, c ;a)* = η ação: *a*	*H(l, c+1 ;a)* = η/2 ação: *a*
H(l+1, c-1 ;a) = η/4 ação: *a*	*H(l+1, c ;a)* = η/2 ação: *a*	*H(l+1, c+1 ;a)* = η/4 ação: *a*

Figura 6.2 – Grade demonstrando a atualização espacial das heurísticas. O estado s é composto pela linha l e pela coluna c, logo $H(l, c; a)$ indica o valor da heurística no estado l e c para a ação a. O par estado-ação que recebe o valor máximo η é o atual, que, na figura, é representado pela célula central.
Fonte: Autor

g) Quantidade de Episódios: 500.

Para realizar a análise de desempenho do algoritmo proposto, experimentos similares foram conduzidos com o uso do algoritmo *Q-Learning* (WATKINS, 1989) e também com o uso do algoritmo HAQL com heurísticas definidas a partir de um conhecimento *a priori (ad hoc)* (BIANCHI, 2004). No caso deste ensaio, a ação "vá para baixo" constitui a heurística, sendo esta ação a detentora dos valores predefinidos de H, que neste experimento foram iguais a 3,0. O valor de H não pode ser muito alto para a heurística definida *a priori*, pois, do contrário, a função Q encontra dificuldades para se sobrepor a H e o robô pode demorar mais tempo para aprender.

O procedimento total do aprendizado foi então realizado 30 vezes para cada algoritmo, de forma que as comparações pudessem ser feitas com os valores médios obtidos destes treinamentos.

Os resultados encontrados nas figuras 6.3 e 6.4 mostram que o *Q-Learning* Acelerado por Heurísticas obtidas por meio de Demonstrações - HfDAQL apresenta uma melhora significativa quando comparado ao *Q-Learning* acelerado por heurísticas definidas *ad hoc* e, principalmente, quando comparado ao *Q-Learning* . Como, no algoritmo proposto, o agente robótico já tem um conhecimento prévio da política, o desempenho é otimizado e a quantidade de passos necessários para se alcançar a meta fica muito próxima da quantidade ótima.

A figura 6.5 mostra as ações a serem tomadas em cada estado s, ou seja, a política de ações. Conforme pode ser verificado, o agente aprendeu a não bater nas paredes, a desviar dos obstáculos e, a partir de qualquer estado do ambiente, seguir o caminho até o objetivo G.

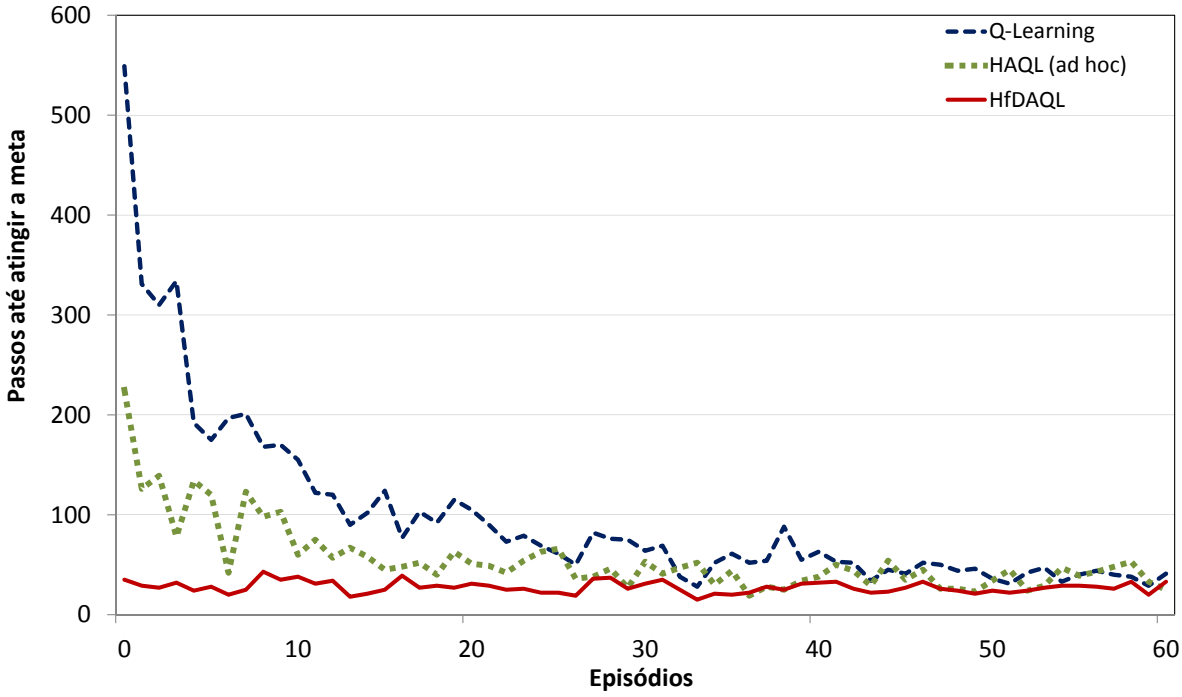


Figura 6.3 – Gráfico da quantidade de passos necessários para se atingir a meta por Episódios. Foco no início do aprendizado, onde a diferença entre as quantidades de passos necessários para cada algoritmo é maior.
Fonte: Autor

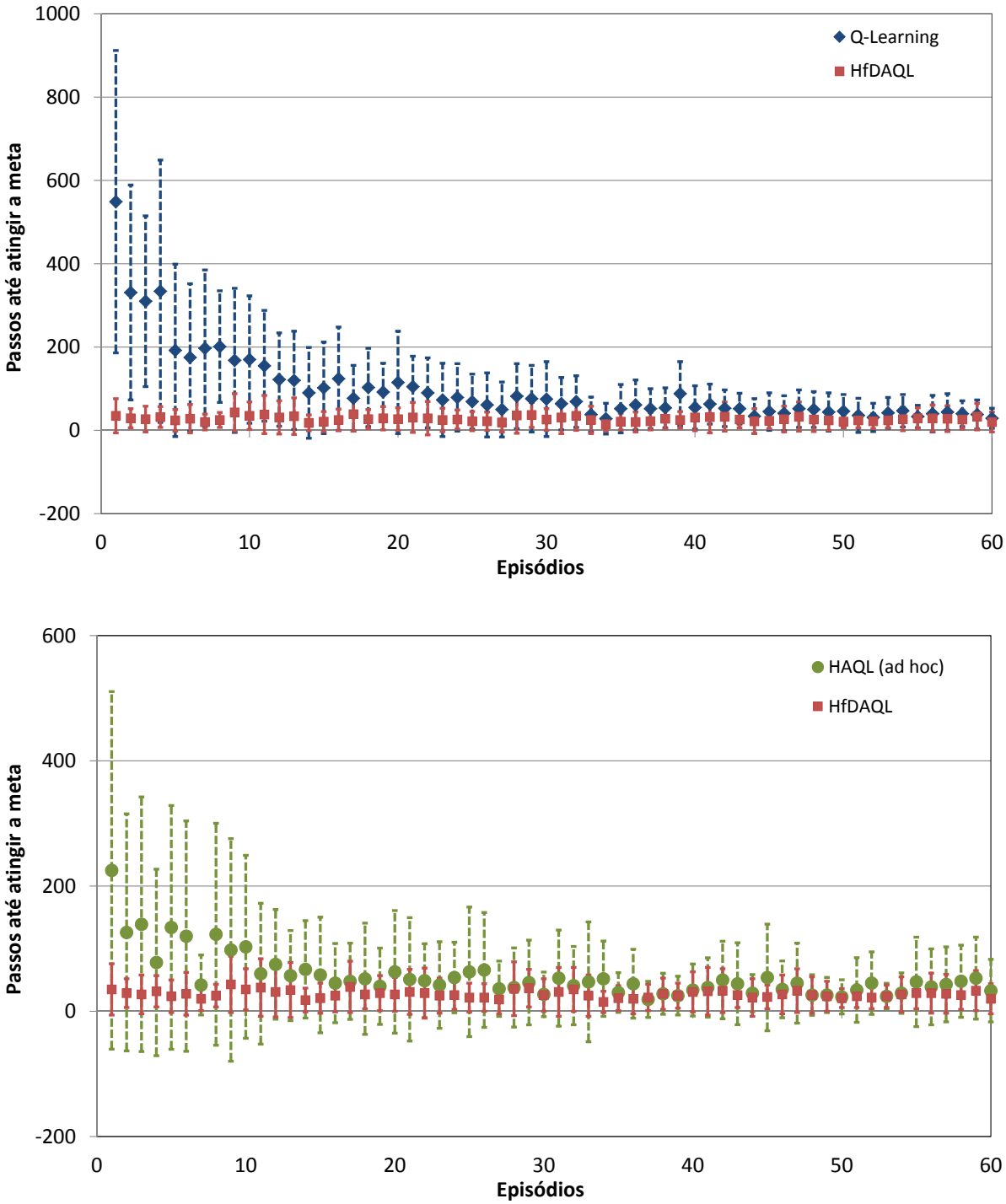


Figura 6.4 – Gráficos das quantidades de passos necessários para se atingir a meta por Episódios, com barras de erro. O superior usa o algoritmo *Q-Learning* como base de comparação e o inferior o algoritmo HAQL (com heurísticas definidas *ad hoc*).
Fonte: Autor

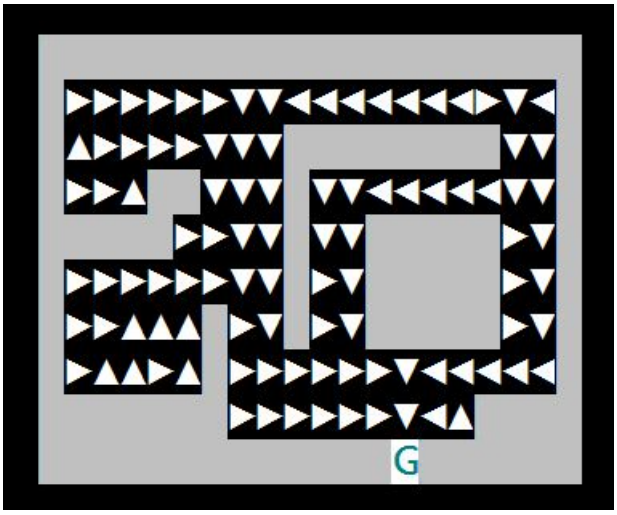


Figura 6.5 – Política aprendida com o uso do aprendizado por reforço acelerado com heurísticas obtidas por meio de demonstrações – HfDAQL.
Fonte: Autor

Foi realizado também o teste t de Student (SPIEGEL, 1984) para verificar a hipótese de que o uso da heurística obtida por meio de demonstrações realmente acelera o aprendizado. O teste t pode ser utilizado para comparar se dois experimentos com médias μ_x e μ_y e com desvios padrão aproximadamente iguais possuem resultados estatisticamente diferentes (NEHMZOW, 2006). Este teste se mostra viável quando as amostras seguem uma distribuição normal de probabilidade e quando a comparação é feita entre dois grupos de dados independentes.

Sendo assim, o módulo de T foi calculado para cada episódio a partir dos mesmos dados que foram utilizados para a criação das figuras 6.3 e 6.4, o que permitiu a comparação entre o *Q-Learning*, o HAQL (com heurísticas definidas *a priori*) e o HfDAQL proposto. O gráfico superior da figura 6.6 compara os resultados obtidos no *Q-Learning* e no HfDAQL, mostrando que até o 17^o episódio os algoritmos são significativamente diferentes, com nível de confiança maior que $99,99\%$. Até o 40^o episódio a diferença entre os algoritmos se revela com nível de confiança maior que 95%. Já o gráfico inferior da figura 6.6 exibe a comparação dos resultados obtidos nos algoritmos HAQL e HfDAQL, no qual pode-se verificar um nível de confiança superior a 95% basicamente até o 10^o episódio.

Todos os experimentos apresentados nesta seção foram codificados em linguagem C, com o uso da IDE DEV C++ e executados em um notebook com processador Intel® Core™ i3, 2 GB de memória RAM e sistema operacional Windows® 7.

Os resultados dos experimentos realizados nesta seção com os algoritmos HfDAQL e *Q-Learning*, foram também publicados por Perico e Bianchi (2011) nos anais do X Simpósio Brasileiro de Automação Inteligente – SBAI 2011.

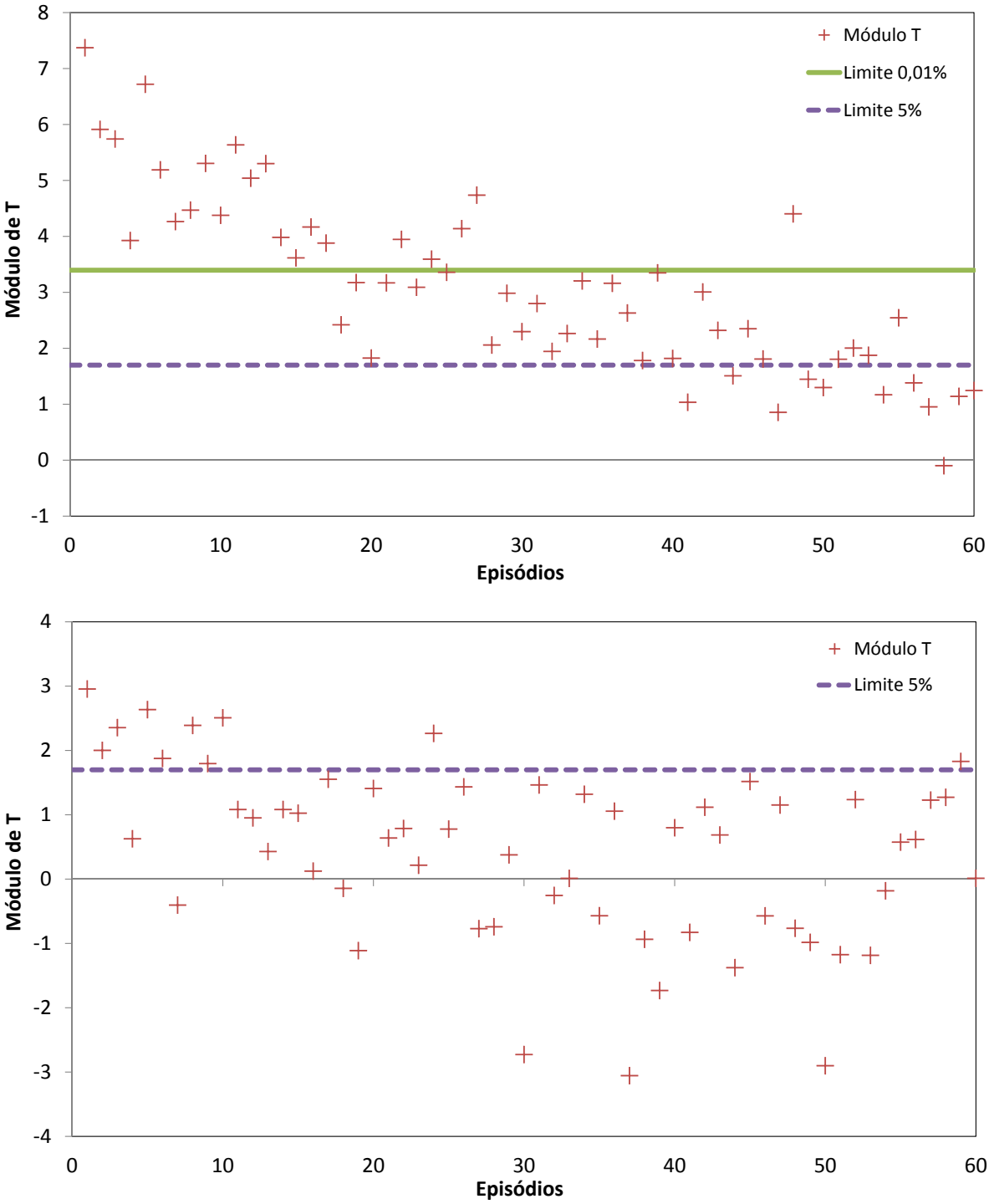


Figura 6.6 – Resultado do teste t de Student para os experimentos realizados no Mundo de Grades. O superior usa o algoritmo *Q-Learning* como base de comparação e o inferior o algoritmo HAQL (com heurísticas definidas *ad hoc*).
Fonte: Autor

6.2 Simulação de um robô real com o uso do Player/Stage

6.2.1 Player/Stage

O Player/Stage é um projeto fundado com o intuito de criar e distribuir *softwares* gratuitos para serem usados nas pesquisas realizadas no domínio da robótica e de sistemas compostos por sensores. Lançado no ano de 2000 por Brian Gerkey, Richard Vaughan e Andrew Howard na Universidade do Sul da Califórnia (*University of Southern California* - USC), o projeto é composto basicamente por dois programas: o Player, que é a camada de abstração de *Hardware* (*Hardware Abstraction Layer* - HAL) e o Stage, que é o ambiente de simulação 2D. Além disso, de 2004 até 2011 o projeto contou também com o simulador 3D Gazebo, mas em 2011 ele se tornou um projeto independente suportado pelo laboratório de pesquisas robóticas Willow Garage.

Desenvolvido como uma evolução do *software* Golem, projetado por Kasper Stoy e por Richard Vaughan em 1999, o Player é responsável pela interface entre o programa de controle e o robô, ou seja, ele permite que o programador controle os *hardwares* do robô sem ter que conhecer em detalhes como cada *hardware*, sensor ou parte do robô funciona (OWEN, 2010). O Player é baseado no modelo cliente-servidor, sendo o servidor responsável por gerenciar as tarefas de baixo nível do robô e o cliente responsável pela comunicação entre o código de controle e o servidor.

O Player possibilita que o código de controle seja escrito em qualquer linguagem de programação e que este código seja executado a partir de qualquer computador que tenha uma conexão com o robô. O Player também suporta uma grande variedade de robôs e dispositivos e permite que novos acessórios sempre sejam criados.

O *software* Stage é um simulador bidimensional multi-robô que pode simular uma população de robôs móveis em um ambiente virtual, trabalhando como um *plugin* do Player. De fato, o Stage interage com o Player da mesma forma que um robô real faria, pois ele recebe as instruções do Player e as transforma em uma simulação, usando para isso robôs e ambientes virtuais que são capazes de alimentar o Player com informações fornecidas pelos sensores e *hardwares* do robô simulado. O Stage foi desenvolvido, basicamente, como uma evolução do simulador Arena, criado, também, por Richard Vaughan, em 1998.

Juntos, Player e Stage podem realizar simulações de robôs móveis em ambientes virtuais (OWEN, 2010). O Player/Stage apresenta como vantagem a fácil transição entre a simulação e o *hardware*, pois praticamente nenhuma alteração é necessária na programação para aplicar o que foi simulado em um robô real, sendo que somente o Player é usado no robô verdadeiro.

O projeto Player/Stage teve seu nome baseado na citação "*All the world's a stage; and all the men and women are merely players*" (Todo o mundo é um palco; e todos os homens e mulheres são meros autores) de Shakespeare (1623). Os programas do projeto podem ser executados em sistemas operacionais baseados em Unix, como o Linux, o Solaris e o BSD e tem licença gratuita, portanto, todo o código do programa é livre e pode ser copiado, modificado e distribuído segundo os termos da GNU *General Public License*.

6.2.2 Robô escolhido para as simulações

O arcabouço proposto nesta seção considera um robô móvel terrestre que se move sobre um solo plano em um ambiente fechado. Além disso, seus sensores devem permitir que ele possa se localizar e detectar a presença de objetos ao seu redor.

O Pioneer 2-DX (figura 6.7), da ActivMedia Robotics® , nome anterior da atual Adept MobileRobots® , foi o robô utilizado em todas as simulações desta seção. Este robô é composto por um corpo de alumínio de 44 cm de comprimento por 38 cm de largura e 21,5 cm de altura. Ele tem direção diferencial, sendo que as rodas principais medem 18,5 cm de diâmetro e 5 cm de largura e, para estabilizar a base, ele conta com uma roda castor localizada na parte traseira (figura 6.8).

O robô conta com dois motores de corrente contínua reversíveis de alta velocidade e alto torque, um para cada roda principal. Isto permite que ele possa chegar a uma velocidade de translação máxima de 1,6 m/s e 230^o/s de velocidade de rotação. Além disso, ele pode ter diversos tipos de sensores, como *encoders*, sonares, lasers, câmeras e *bumpers*, porém, neste trabalho somente os *encoders* e os sonares serão utilizados.

Figura 6.7 – Robô Pioneer 2-DX da ActivMedia Robotics® .
Fonte: Laboratótio de Automação e Robótica da Universidade de Bolonha (*Università di Bologna*)

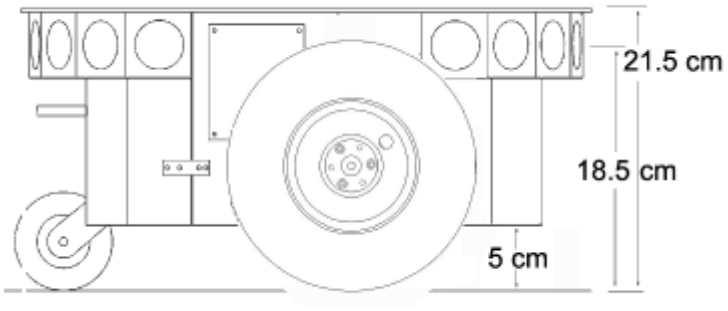


Figura 6.8 – Dimensões básicas do robô Pioneer 2-DX.
Fonte: ActivMedia Robotics®

Os *encoders* deste robô são usados para medir o deslocamento das rodas e fornecer a localização do robô. Eles possuem 500 marcas, podendo assim detectar deslocamentos nas rodas de até $0{,}72^o$. Contudo, definir a localização do robô por meio de dados recebidos dos *encoders*, ou seja, por odometria, pode ser um método arriscado, pois este processo apresenta erro acumulativo.

Uma outra possível maneira de se determinar a postura (posição e orientação) relativa do robô no ambiente, que pode ser capaz de diminuir o problema do erro acumulativo encontrado na odometria, é a Localização de Monte Carlo (*Monte Carlo Localization* – MCL). A MCL é um método probabilístico baseado no filtro de Bayes, no qual partículas representam a possibilidade do robô estar em uma determinada localização (DELLAERT et al., 1999).

Para a implementação da MCL, o Player/Stage possui o *driver* $amcl$, acrônimo de *Adaptative Monte Carlo Localization*, contudo este driver requer que o laser esteja disponível para funcionar corretamente. Com o intuito de testar o método, e ignorando o fato de que este trabalho considera somente o uso de *encoders* e sonares, o laser LMS200 foi configurado e o $amcl$ foi implementado na simulação, porém a localização do robô não funcionou bem, mesmo após diversos tipos de configurações e testes.

Assim, para simplificar a localização do robô neste trabalho, uma odometria ideal que ignora os erros gerados durante a movimentação do robô foi adotada. Este artifício só é possível no ambiente de simulação, com o uso de um recurso disponível no Player/Stage.

O Pioneer 2-DX tem dois conjuntos de 8 sonares cada, localizados a 18,5 cm do chão. A primeira série fica na parte frontal do robô (figura 6.9) e a segunda na parte traseira, somando um total de 16 sonares. Todos eles trabalham com uma frequência de 25 Hz e tem sensibilidade para detectar distâncias que variam de 10 cm a 5 m.

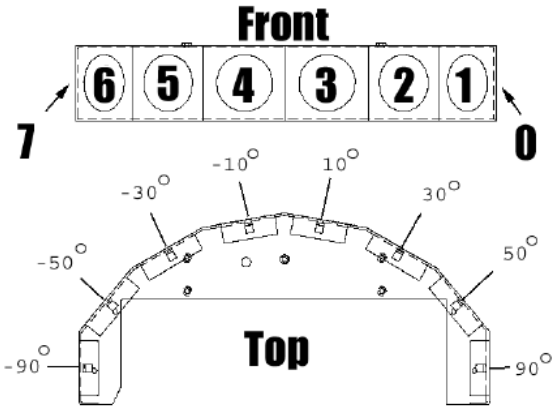


Figura 6.9 – Disposição dos sonares frontais do Pioneer 2-DX.
Fonte: ActivMedia Robotics®

6.2.3 Ambiente

Para a realização das simulações foi criado um ambiente virtual baseado em uma sala real, que fica localizada no 5^o andar do prédio K do Centro Universitário da FEI (sala K-506). Esta sala tem dimensões de 10 x 7,25 metros e tem paredes, mesas e armários em seu interior, conforme figura 6.10. Com o intuito de simplificar os experimentos e considerando que os sonares do robô estão a 18,5 cm do chão, as mesas utilizadas neste ambiente virtual são tidas como um único bloco de madeira sem qualquer vão para as pernas, pois, de outra forma, o robô não poderia detectá-las. Porém, para resolver este problema só é necessário colocar outra série de sonares na altura das mesas ou utilizar um laser para reconhecer os obstáculos.

O objetivo básico do robô é aprender a encontrar a saída da sala a partir de qualquer ponto inicial, sem colidir com os obstáculos.

6.2.4 Simulação

A figura 6.11 exibe o ambiente de simulação Player/Stage com a integração da sala K-506 e do robô Pioneer 2-DX virtuais. É possível visualizar também pela imagem o alcance dos sonares, que são representados em verde na imagem. Neste domínio foram realizadas as comparações entre os algoritmos *Q-Learning* , HAQL com heurísticas definidas a partir de um conhecimento *a priori* e o HfDAQL proposto.

Para realizar a simulação, o ambiente foi discretizado em células quadradas, com dimensões de 25 x 25 centímetros. Desta forma, foi criada uma grade de 40 x 29 células. A orientação

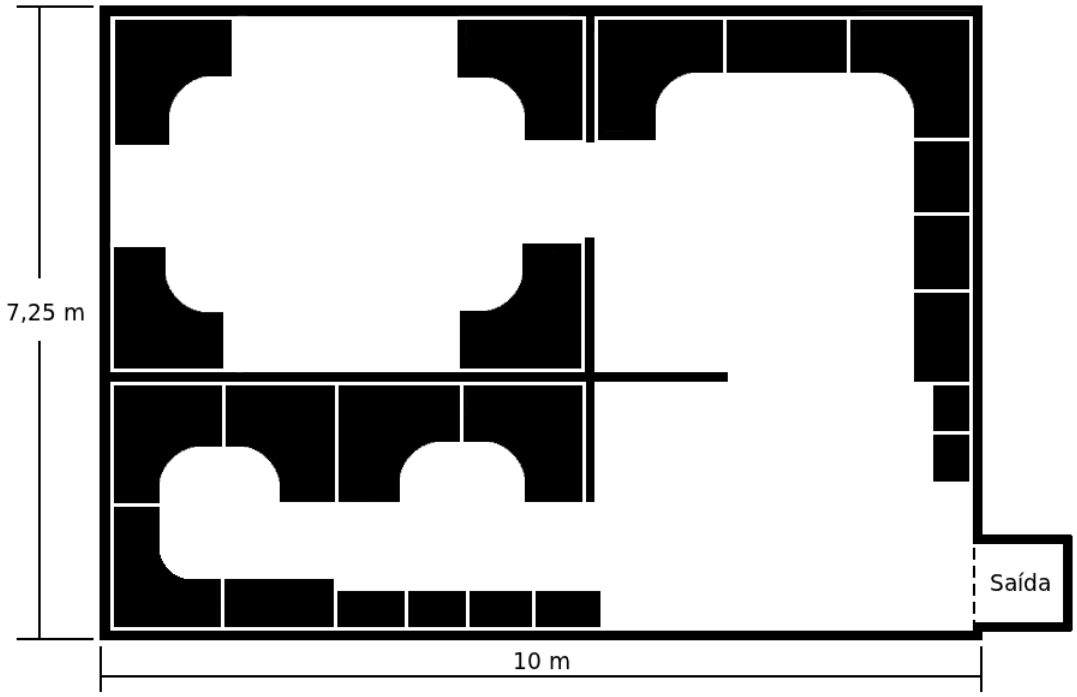


Figura 6.10 – Sala virtual utilizada nas simulações. Os objetos escuros representam os obstáculos (paredes, mesas e armários).
Fonte: Autor

do robô foi particionada em 16 valores discretos, múltiplos de 22,5^o. Assim, as variáveis x, y e θ, utilizadas para designar o estado do robô durante o aprendizado, podem ser consideradas como uma aproximação do estado real do robô (BIANCHI, 2004).

Foram adotadas três possíveis ações para o robô: mover para frente, girar no próprio eixo no sentido horário e girar no próprio eixo no sentido anti-horário. As ações são executadas até que aconteça uma mudança no estado s, representado pela tripla (x, y, θ), do robô. Adicionalmente a estas ações, existe a possibilidade de o robô se mover para trás, mas esta é uma ação passiva, executada automaticamente quando o robô tenta se mover para o lado e está a menos de 25 cm de um obstáculo lateral ou quando tenta se mover para a frente e está a menos de 30 cm de um obstáculo frontal. Esta ação resulta sempre em um reforço negativo para o robô.

Os reforços definidos para os experimentos desta seção são os seguintes:

a) -50 quando o robô chega muito próximo a um obstáculo e precisa se mover para trás;

b) -1 quando o robô executa uma ação que não resulta em encontrar o estado meta;

c) 1000 para quando o robô encontra o estado meta.

O robô inicia um episódio de treinamento em uma postura aleatória e tem como objetivo encontrar a saída da sala, definida pela região $x >$ 10,00 m; 0,50 m $> y \geqslant$ 0,75 m. O robô

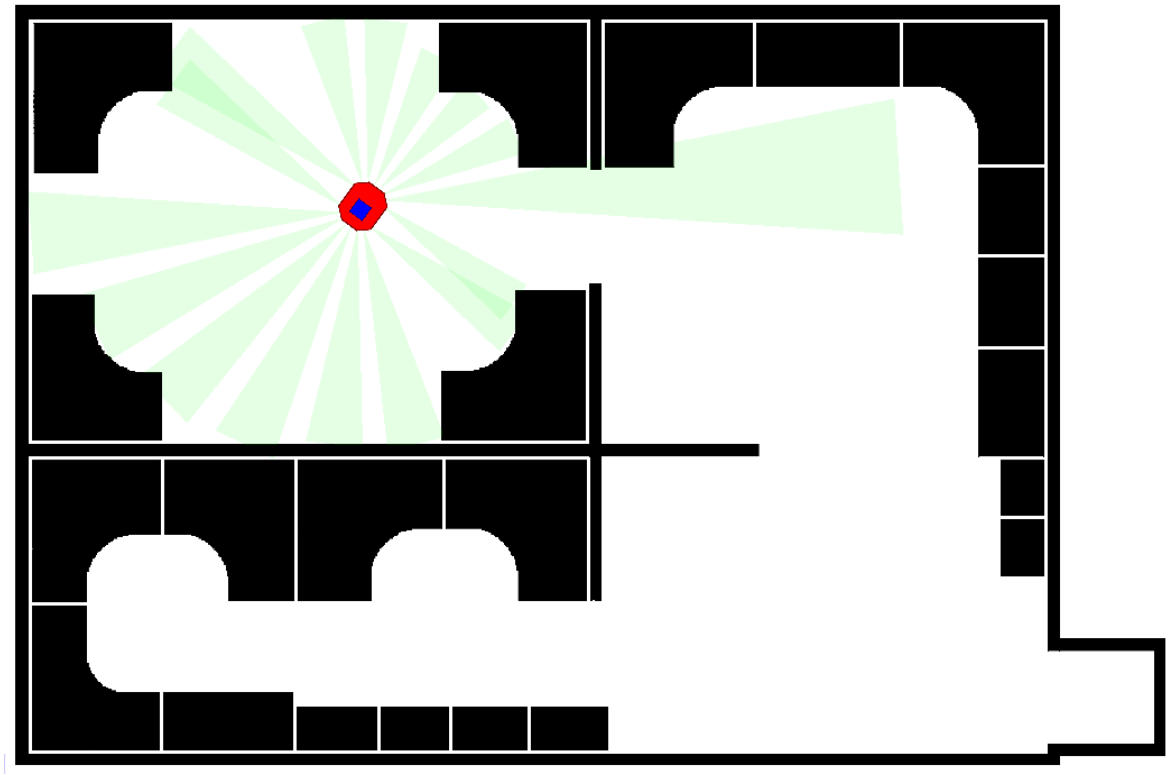


Figura 6.11 – Simulação no Player/Stage com o uso da sala K-506 e do robô Pioneer 2-DX virtuais.
Fonte: Autor

sempre recebe os valores iniciais de x, y e θ para poder começar a se localizar com o uso da odometria, conforme descrito na seção 6.2.2.

Os parâmetros utilizados em todos os métodos de aprendizado foram os mesmos:

a) Taxa de Aprendizado $\alpha : 20\%$;

b) Fator de Desconto $\gamma : 90\%$;

c) Porcentagem de exploração $\varepsilon : 20\%$;

Neste domínio, foram realizadas as comparações entre os algoritmos *Q-Learning*, HAQL com heurísticas definidas *a priori* e o HfDAQL proposto.

Para os experimentos realizados com o algoritmo HAQL com heurística *ad hoc*, o valor concedido para $H(s, a)$ foi igual a 5,0 para a ação "mover para frente"em todos os estados referentes às orientações 0^o e 320^0 e valor igual a 0,0 para todos os outros estados e ações, conforme representado na figura 6.12. Desta forma, a heurística deve ajudar o agente a encontrar o seu objetivo, dando indícios de qual caminho ele deve seguir. Conforme já citado no experimento do Mundo de Grades, o valor de H não pode ser muito alto quando a heurística é definida com

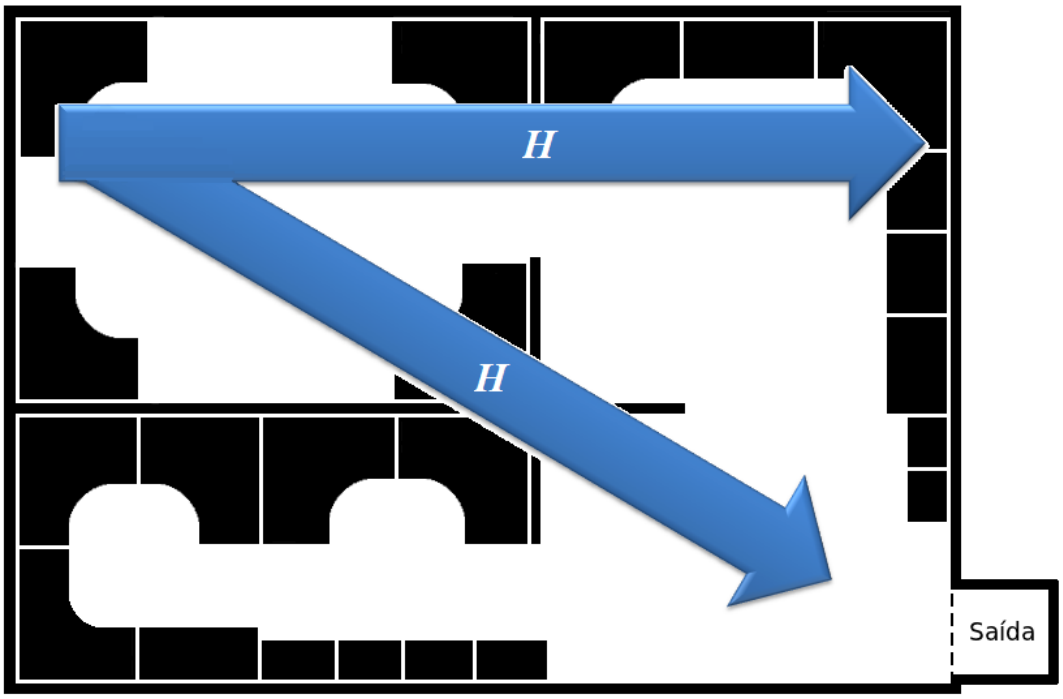


Figura 6.12 – As setas azuis representam a heurística definida *a priori (ad hoc)*.
Fonte: Autor

conhecimento *a priori*, pois, do contrário, existe uma certa dificuldade para que a função Q se sobreponha a H e o robô pode demorar mais tempo para aprender.

Por sua vez, no algoritmo HfDAQL as demonstrações são as responsáveis pela obtenção das heurísticas. Então, um valor máximo η foi atribuído a $H(s, a)$ para os pares estado-ação utilizados durante as demonstrações de caminho e este H foi espalhado para o maior número possível de pares estado-ação que não participaram diretamente da demonstração, por intermédio dos conceitos de espalhamento espacial realizado com similaridade por conectividade e por vizinhança.

Para se definir η, a premissa estabelecida na seção 3.3.1 foi seguida, levando em consideração $r_{min} = -1$, que é a recompensa dada quando o robô executa uma ação que não resulta em encontrar o estado meta. Esta não é a menor recompensa concedida ao agente, pois a menor seria -50 para quando o robô se aproxima muito de um obstáculo, todavia a recompensa de -1 é a menor com maior frequência e pode garantir um H máximo que permita a convergência ao aprendizado para qualquer uma das recompensas preestabelecidas.

Assim, o valor de η foi definido como 10, conforme pode ser verificado na equação 6.2.

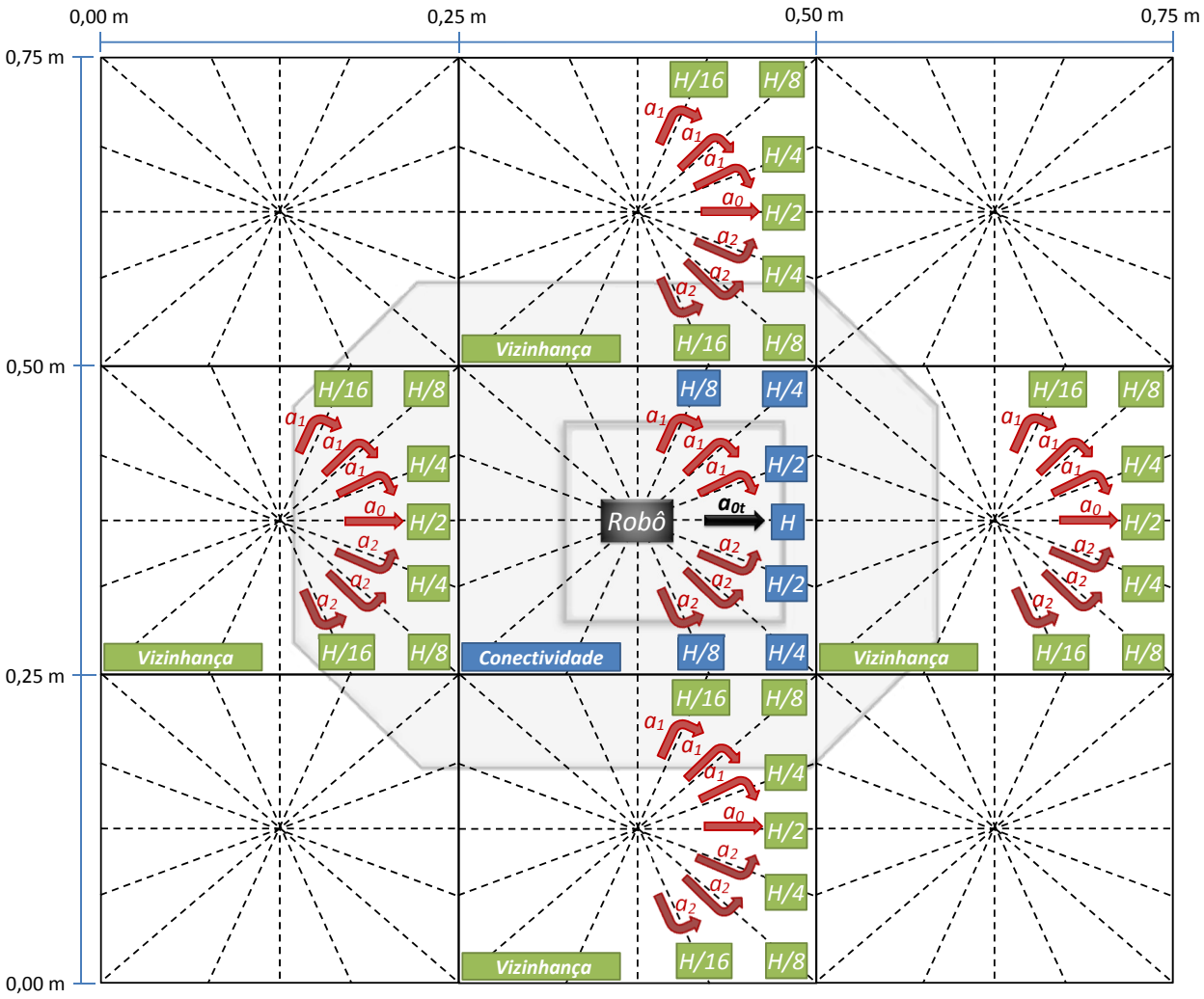


Figura 6.13 – Espalhamento espacial da heurística por meio do QSx adaptado, utilizando a similaridade por conectividade e vizinhança. H representa o valor máximo η atribuído para a heurística.
Fonte: Autor

$$h_{max} \leqslant \left| \frac{-1}{1-0,9} \right| \therefore h_{max} \leqslant 10 \tag{6.2}$$

O espalhamento foi realizado conforme exibido na figura 6.13, onde as células da grade criada são representadas pelas linhas contínuas e a discretização das orientações pelas linhas tracejadas. Ainda pela figura, é possível verificar que o espalhamento foi realizado com o uso da similaridade por conectividade (equação 6.1 apresentada na seção 6.1) no estado presente do robô, considerando a ação tomada a_{0t}, e com o uso da similaridade por vizinhança (equação 6.3) nas células vizinhas.

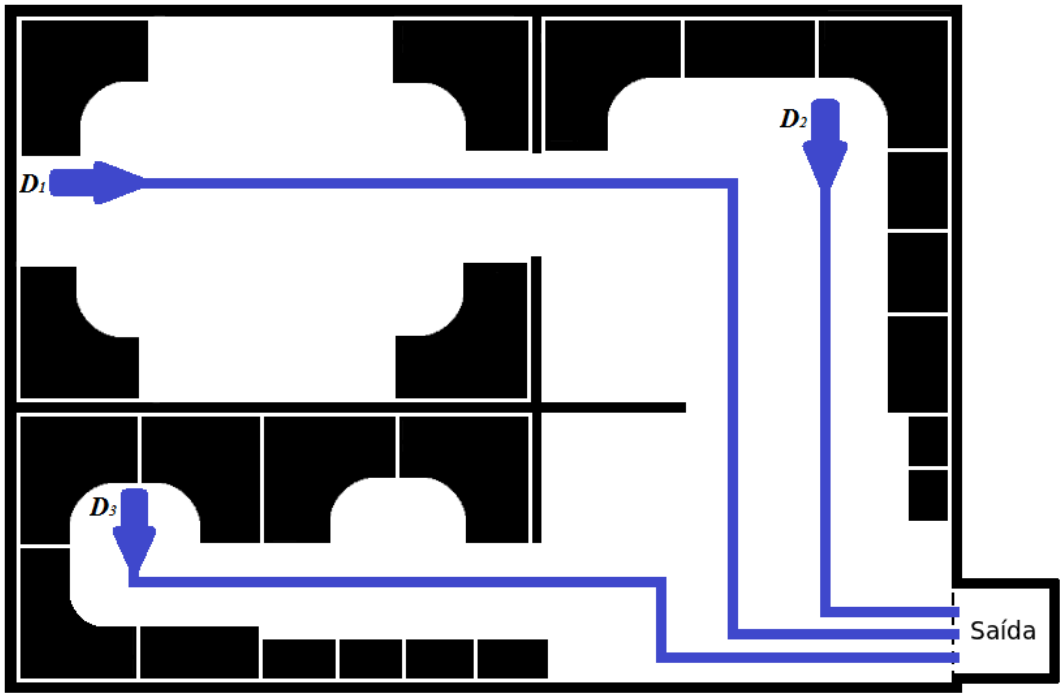


Figura 6.14 – Exemplo com 3 demonstrações (D_1, D_2, D_3) consideradas boas ou subótimas.
Fonte: Autor

$$f_x(x, s) = \tau^{d_{s,x}} \tag{6.3}$$

Onde:

a) O fator de similaridade τ é igual a $0,5$;

b) O grau de diversidade $d_{s,x}$ representa a distância entre o estado s ao x.

Neste experimento, as demonstrações consistem em guiar o robô a partir de uma posição inicial aleatória até a saída da sala, por teleoperação, sendo que o professor tem visão global do sistema. As demonstrações serão consideradas subótimas quando existir a tentativa, por parte do professor, de conduzir o robô diretamente até o objetivo proposto, considerando o menor número de passos possível (figura 6.14). Estas demonstrações não serão aqui definidas como ótimas, pois, como os exemplos de caminhos serão realizados por um professor, caberá a ele decidir qual caminho percorrer e nem sempre este caminho será a melhor escolha. No entanto, a variação entre o subótimo e o ótimo é bastante baixa, sendo que às vezes esta variação pode não existir.

Desta forma, um estudo foi realizado para se verificar como a quantidade destas demonstrações subótimas influencia no aprendizado do agente por intermédio do algoritmo HfDAQL.

O resultado exibido na figura 6.15 é a média de 30 treinamentos, com as suas respectivas barras de erro.

É possível concluir pelo estudo da figura 6.15 que quanto mais demonstrações são realizadas mais eficiente é o aprendizado, uma vez que o número de passos necessários nos primeiros episódios diminui quando o número de demonstrações aumenta, tendo como limite mínimo o número ótimo de passos. Porém, esta não é a única conclusão possível, pois, na realidade, existe mais uma variável que precisa ser analisada neste ensaio. Esta variável será chamada de "qualidade das demonstrações".

Mesmo considerando que todas as demonstrações, até o presente momento, foram realizadas com caminhos subótimos, é preciso se atentar que as demonstrações têm início aleatório e portanto existe a possibilidade delas se sobreporem. Esta sobreposição pode fazer com que a tabela dos valores H não receba nenhum, ou quase nenhum, aperfeiçoamento durante a fase das demonstrações. Assim, independentemente da quantidade de demonstrações realizadas, a heurística criada pode ser sempre a mesma se o início das demonstrações for no mesmo, ou muito próximo, do mesmo estado.

Logo, a "qualidade das demonstrações" faz referência à área do ambiente percorrida durante as demonstrações, tendo-se como base que os caminhos demonstrados sempre visam conduzir o robô diretamente até o estado objetivo, a partir de qualquer estado do domínio. Desta forma e considerando a condição de caminhos subótimos, quanto maior a área coberta na fase das demonstrações, melhor será a eficácia desta fase e, consequentemente, o conhecimento prévio transmitido para a fase do aprendizado será mais eficiente no seu intuito de acelerar o RL.

Para executar o experimento pertinente à qualidade das demonstrações, o ambiente foi discretizado em células de $0,56m$. A figura 6.16 exibe a área livre coberta durante as demonstrações, por meio da identificação cinza das células. Os resultados observados nos gráficos da figura 6.17 refletem a média de 30 treinamentos e indicam a influência da área coberta durante as demonstrações na aceleração do aprendizado. Todas as demonstrações foram realizadas com caminhos subótimos.

Assim, conforme análise, a quantidade das demonstrações influencia na aceleração do aprendizado, mas não é o único fator, uma vez que a quantidade depende diretamente da qualidade desejada e vice-versa, pois n demonstrações que percorrem basicamente o mesmo caminho não são tão eficientes quanto as mesmas n demonstrações percorrendo caminhos diferentes que resultam na cobertura de uma área maior do domínio.

Todos os estudos realizados até agora tiveram como escopo analisar as boas demonstrações e a sua influência na aceleração do RL. Contudo, uma demonstração que não é boa, ou seja,

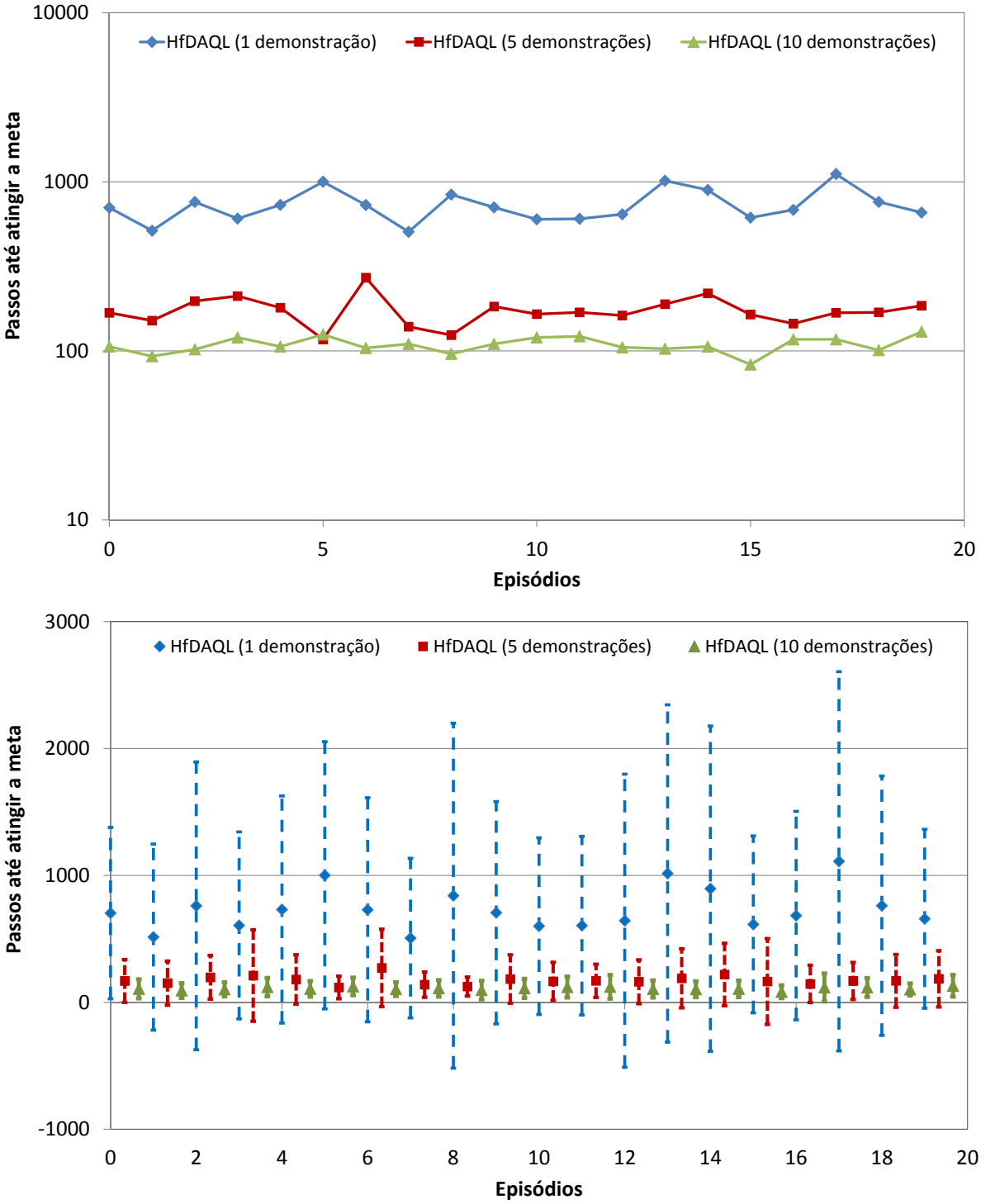
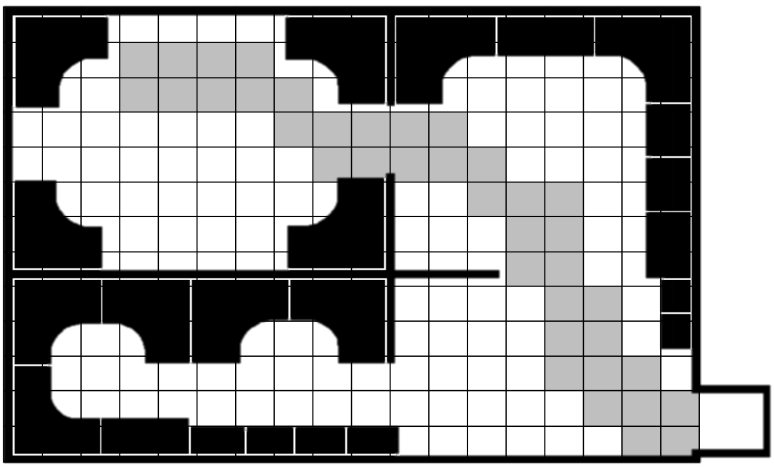
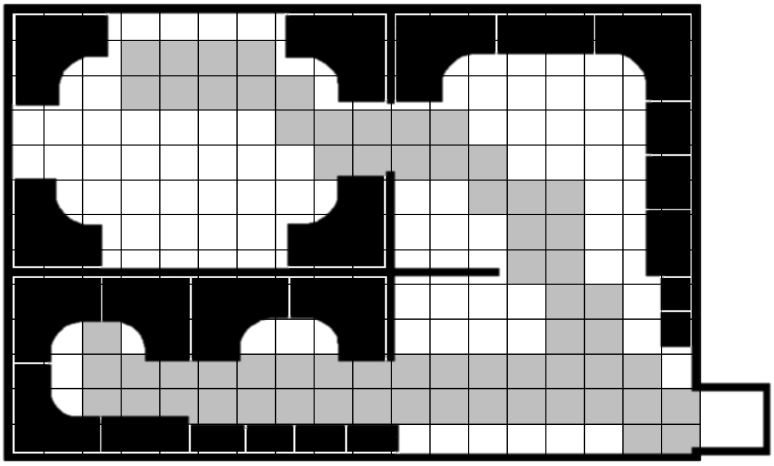


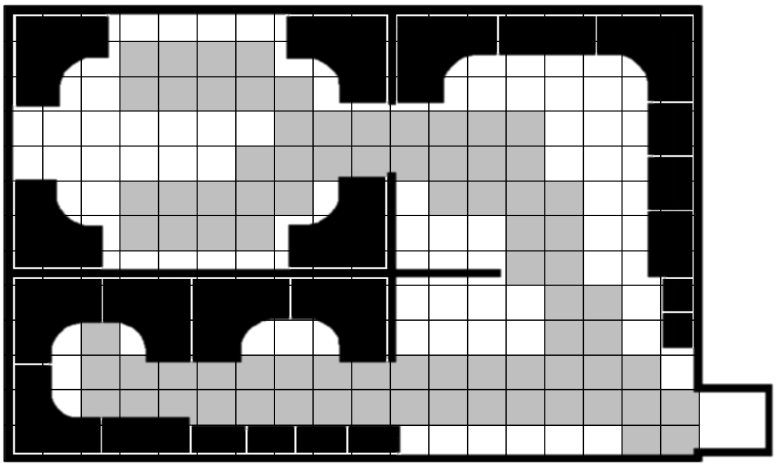
Figura 6.15 – Análise da influência da quantidade de demonstrações com início aleatório no aprendizado. Superior exibe a média (em monolog) e o inferior as barras de erro.
Fonte: Autor



(a) Aproximadamente 30% da área livre coberta



(b) Aproximadamente 50% da área livre coberta



(c) Aproximadamente 60% da área livre coberta

Figura 6.16 – Porcentagem da área livre coberta por demonstrações de caminhos subótimos (representada em cinza).
Fonte: Autor

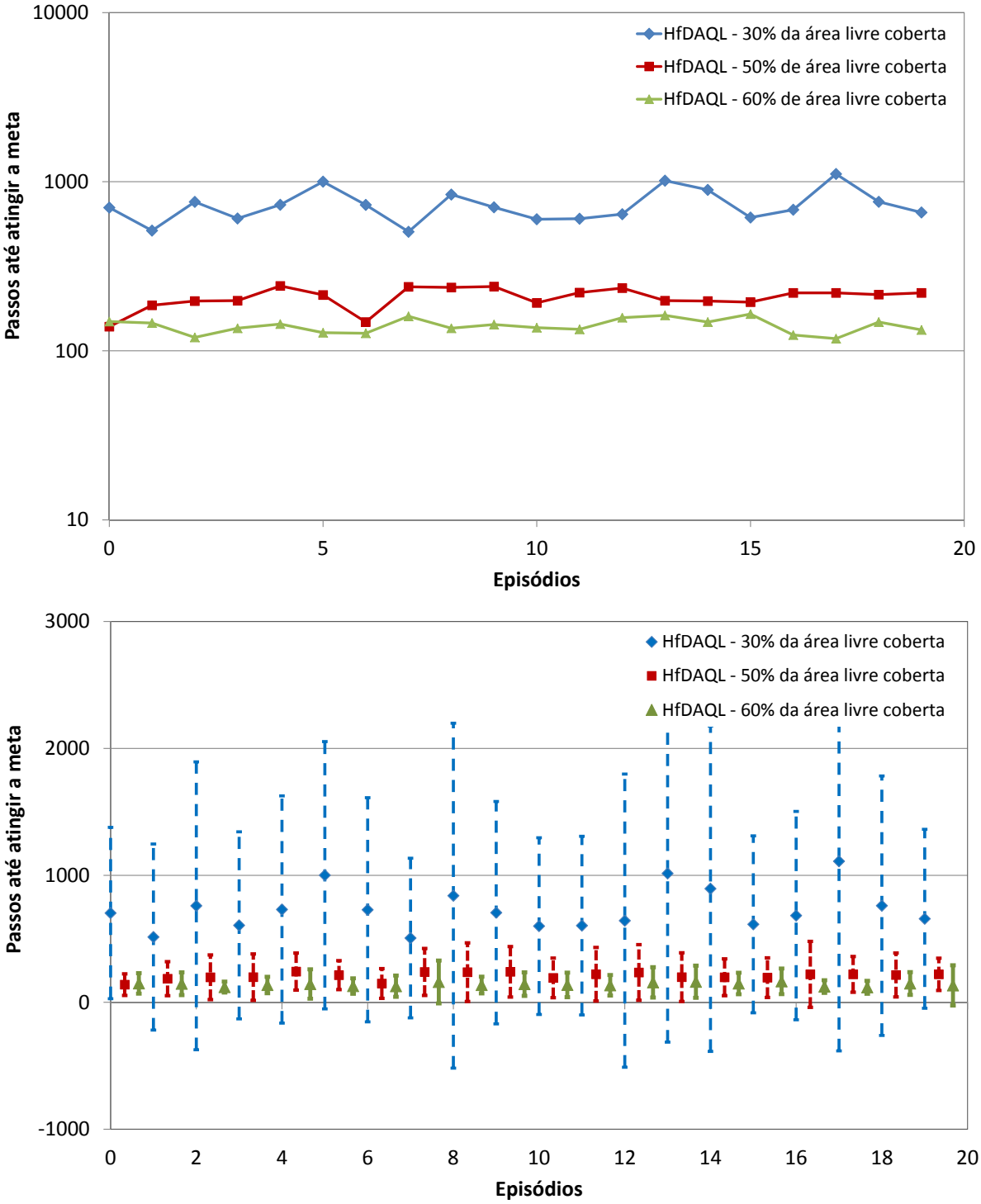


Figura 6.17 – Análise da influência da qualidade das demonstrações no aprendizado com o uso do algorimo HfDAQL. Superior exibe a média (em monolog) e o inferior as barras de erro.
Fonte: Autor

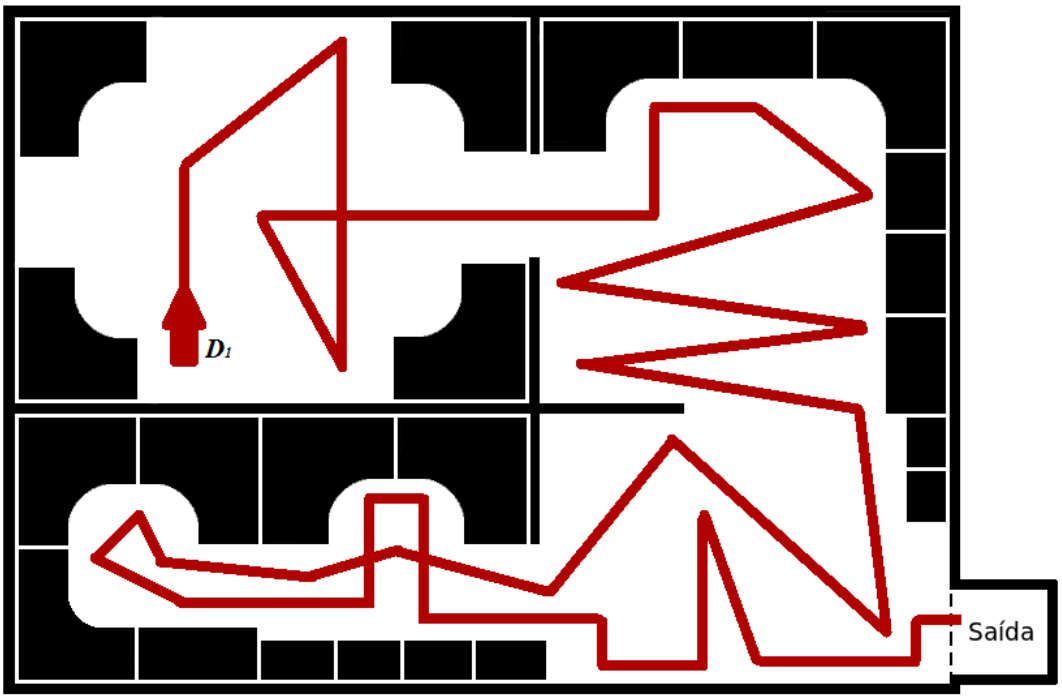


Figura 6.18 – Exemplo com 1 demonstração (D_1) que não é considerada boa.
Fonte: Autor

um exemplo de caminho que também conduz o robô até o seu objetivo, mas que ignora a condição de realizar esta tarefa com o menor número de passos possível (figura 6.18), pode também acelerar o RL quando comparado ao algoritmo *Q-Learning*. Na verdade, uma demonstração "não-boa" acelera o aprendizado de maneira similar a encontrada nos algoritmos HAQL com heurística definida *a priori* e HfDAQL com somente uma demonstração subótima, conforme pode ser analisado na figura 6.19, que exibe a média de 30 treinamentos de cada algoritmo.

Uma vez concluído o estudo sobre as demonstrações, voltamos para os experimentos que permitem a análise do algoritmo HfDAQL proposto. Assim, para a fase de demonstrações do HfDAQL, foram utilizados 5 exemplos subótimos de caminhos ao robô, sendo que os estados iniciais destas demonstrações foram aleatórios e cerca de 64% da área livre foi coberta. Inicialmente, cada experimento foi realizado com 3000 episódios, com exceção do algoritmo *Q-Learning*, que devido ao elevado tempo de convergência, foi limitado a 1000 episódios. Como os resultados foram obtidos a partir de somente 1 treinamento, o recurso de média móvel foi utilizado para deixar mais clara a tendência de cada algoritmo. Logo, o que pode ser melhor identificado no gráfico da figura 6.20 são os pontos obtidos pela média dos últimos 30 períodos alcançados por meio do treinamento realizado.

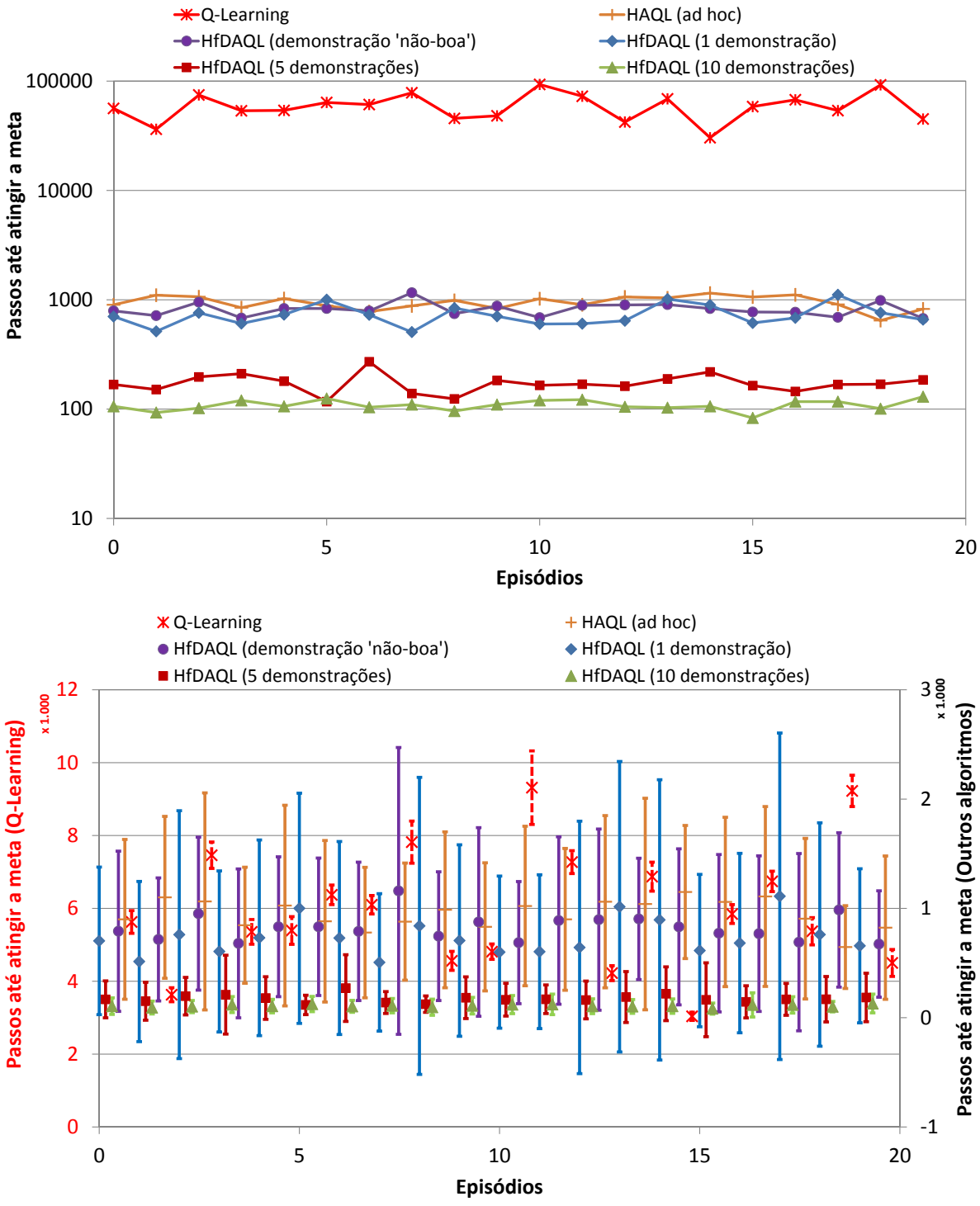


Figura 6.19 – Análise da influência de 1 demonstração considerada "não boa" no aprendizado com o uso do algoritmo HfDAQL. Superior exibe a média (em monolog) e o inferior as barras de erro.
Fonte: Autor

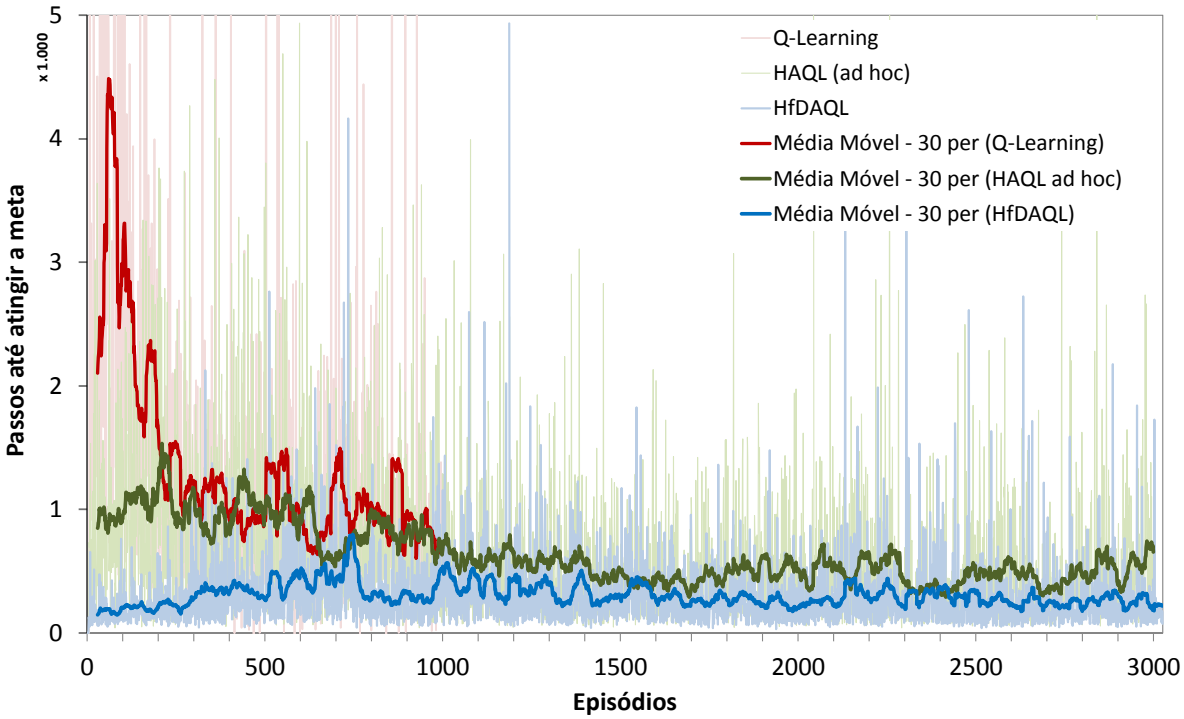


Figura 6.20 – Gráfico da primeira simulação realizada no Player/Stage para análise do HfDAQL. HAQL (ad hoc) e HfDAQL com 3000 episódios, *Q-Learning* com 1000.
Fonte: Autor

É possível notar pela avaliação da figura 6.20, que o robô consegue aprender com todos os algoritmos utilizados, porém, algumas peculiaridades podem ser notadas, como, por exemplo, o fato do *Q-Learning* precisar de muitos passos no início do treinamento para levar o agente robótico até o seu objetivo, o que já era esperado, e o fato de que, com o uso do algoritmo HfDAQL, o aprendizado passa por um momento, entre os episódios 250^o e 750^o aproximadamente, no qual a quantidade de passos para se atingir o estado meta aumenta, porém isto pode ser explicado por alguns fatores, que serão tratados no próximo parágrafo.

O aumento notado no número de passos, quando o HfDAQL foi usado, pode ser explicado, basicamente, por um principal motivo: no início do aprendizado o H tem valor muito alto quando comparado ao Q, o que faz com que a escolha das ações seja fortemente influenciada pela heurística. Porém, com o passar do tempo e considerando que o robô recebe reforço de -1 para ações que não resultam em encontrar o estado meta, Q começa a diminuir e ficar com o valor absoluto igual a H. Como a escolha das ações é feita por meio da somatória de H com Q, a heurística vai gradualmente perdendo a força e o agente aprendiz começa a visitar estados que ele não tinha visitado anteriormente.

Para uma análise mais precisa dos algoritmos, os mesmos experimentos foram realizados novamente, porém com menos episódios, 500, e com mais treinamentos, 10, para cada algoritmo. Os resultados obtidos pela média destes treinamentos são exibidos na figura 6.21.

O teste t de Student (SPIEGEL, 1984) também foi realizado para verificar a hipótese de que o uso da heurística obtida por meio de demonstrações realmente acelera o aprendizado. O módulo de T foi calculado para cada episódio a partir dos mesmos dados que foram utilizados para a criação da figura 6.21, o que permitiu a comparação entre todos os algoritmos testados. A figura 6.22 compara os resultados obtidos no *Q-Learning* e no HfDAQL, mostrando que os algoritmos são significativamente diferentes até aproximadamente o episódio 285, com nível de confiança maior que 95%. Essa conclusão pôde ser obtida pela observação da linha de tendência dos módulos de T.

Já a figura 6.23 exibe a comparação dos resultados obtidos nos algoritmos HAQL e HfDAQL, na qual pode-se verificar, por meio da análise da linha de tendência do módulo de T, um nível de confiança superior a 95% ultrapassando o episódio 300. Em ambos os testes, poucos pontos tiveram nível de confiança menor que 95%, mas, curiosamente, a maior parte deles foi encontrado na comparação que envolveu o *Q-Learning*. Contudo, esta comparação pode ter sido ligeiramente comprometida devido ao fato do *Q-Learning* apresentar grande oscilação nos dados coletados, o que fez com que o desvio padrão ficasse muito alto, conforme pode ser verificado no gráfico com barras de erro da figura 6.21.

Concluindo, pode-se ver na figura 6.24 os caminhos percorridos pelo robô com o uso dos algoritmos *Q-Learning*, HAQL com a heurística definida *a priori* e HfDAQL. As três figuras foram geradas como resultado do aprendizado no 1^o episódio de treinamento. Nota-se, pela análise das figuras, que o robô anda aleatoriamente até encontrar o seu objetivo quando o *Q-Learning* é usado. Já com o HAQL, o robô vagueia menos e tende a encontrar o estado meta de forma mais direta, enquanto que no HfDAQL o robô segue praticamente o caminho subótimo demonstrado, desviando somente para pequenas explorações. Para completar o episódio analisado o *Q-Learning* levou cerca de 3 horas e 47 minutos, o HAQL levou cerca de 33 minutos e o HfDAQL levou pouco menos de 5 minutos.

Todos os experimentos apresentados nesta seção foram simulados no Player/Stage, versões 3.1 e 3.2.2 respectivamente, e codificados em linguagem C++ com o auxílio da IDE Code::Blocks. As execuções foram realizadas em um notebook com processador Intel® Core™ i3, 2 GB de memória RAM e sistema operacional Ubuntu 11.10.

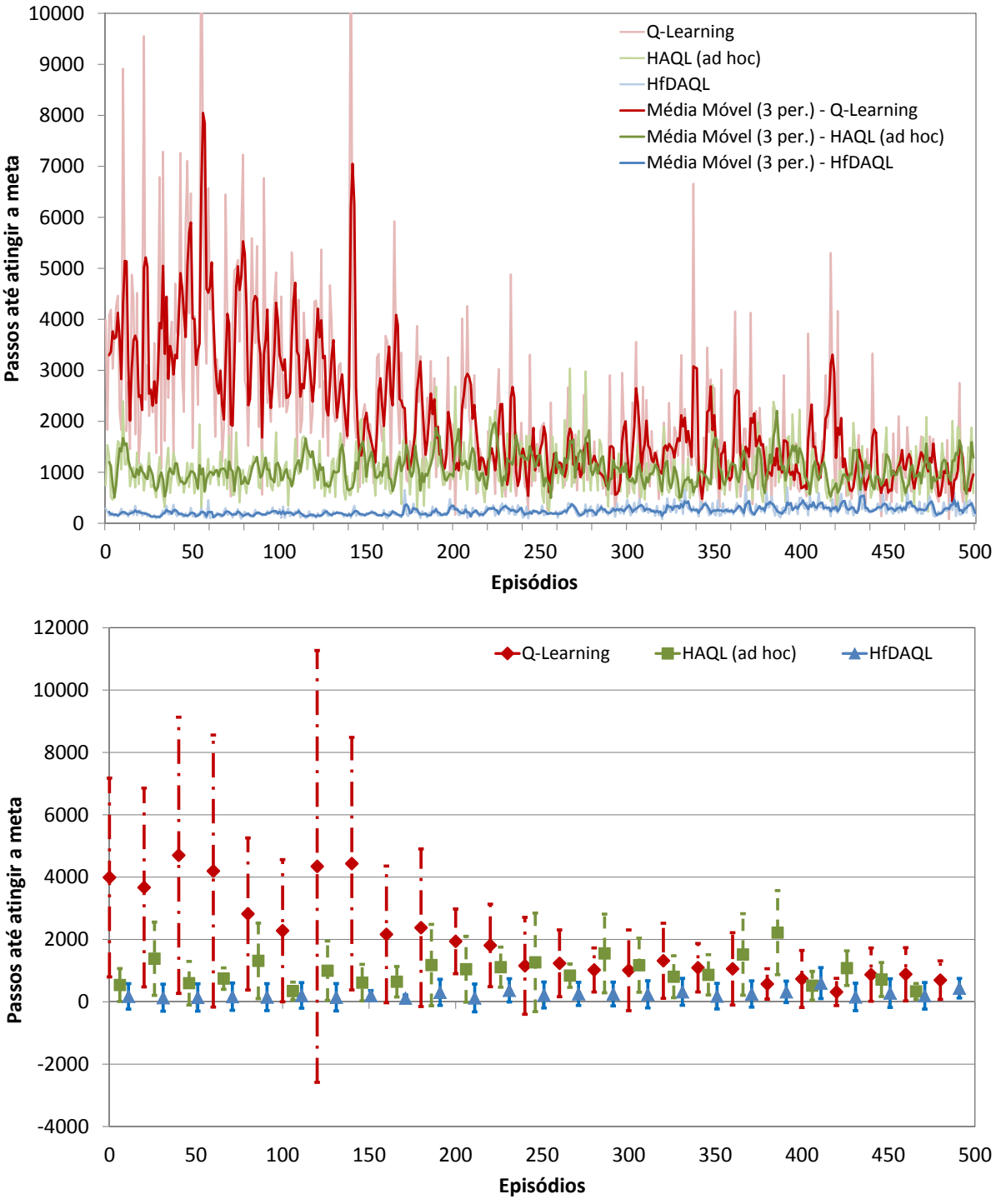


Figura 6.21 – Gráfico com o resultado da média de 10 treinamentos com 500 episódios para cada algoritmo.
Fonte: Autor

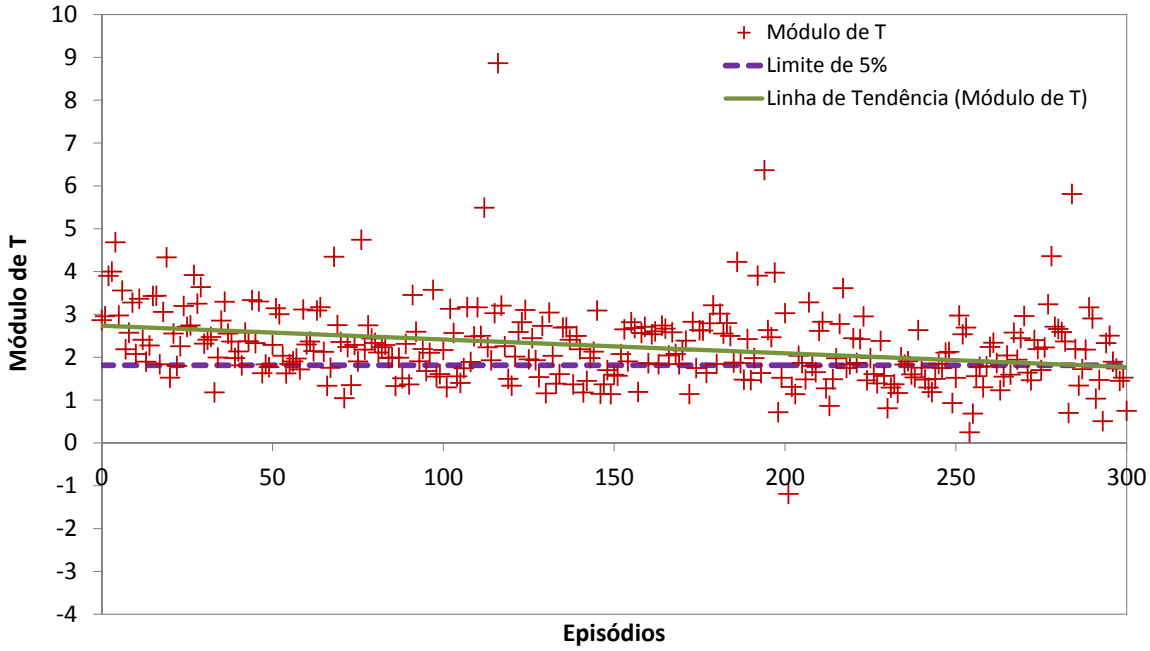


Figura 6.22 – Resultado do teste t de Student. Comparação entre o *Q-Learning* e o HfDAQL.
Fonte: Autor

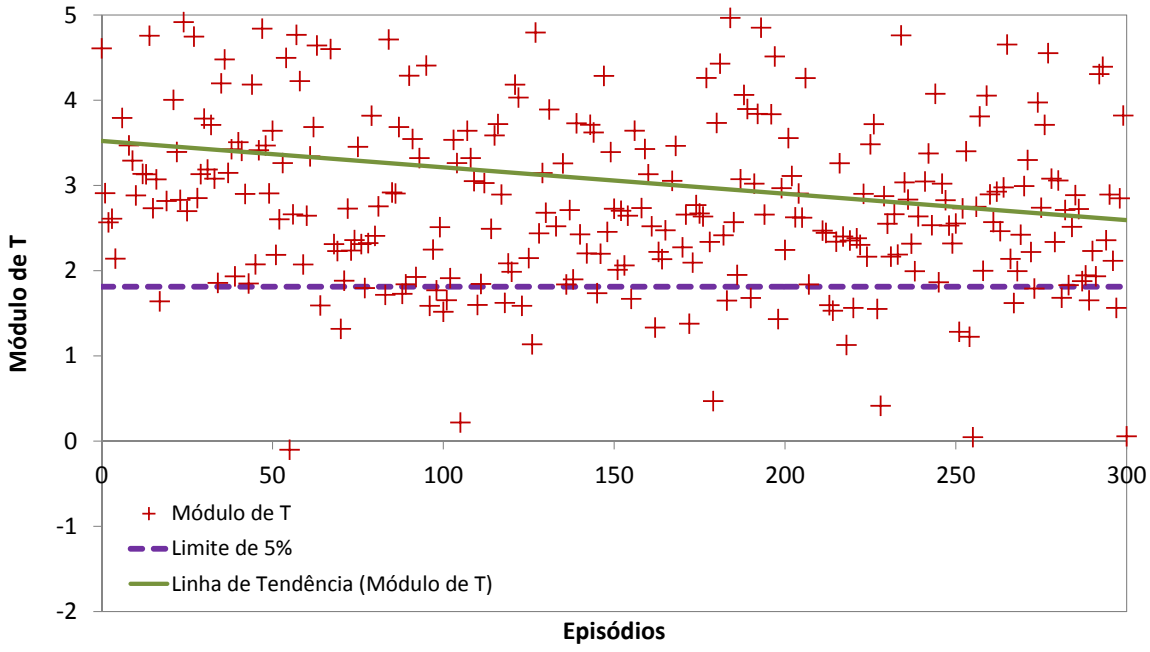
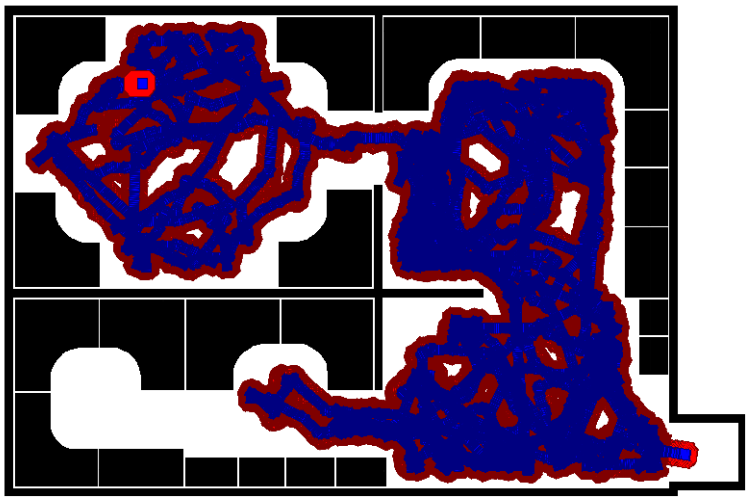
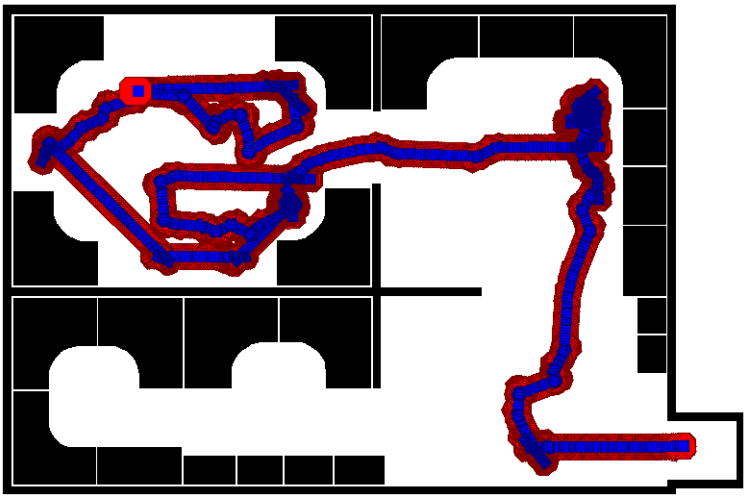


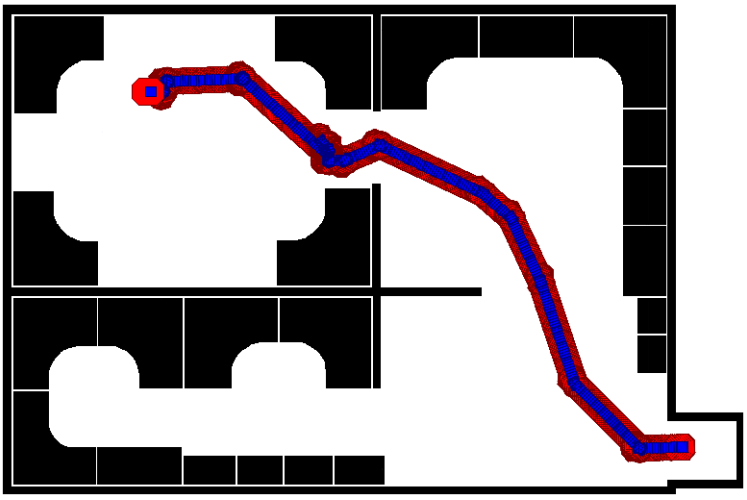
Figura 6.23 – Resultado do teste t de Student. Comparação entre o HAQL com heurísticas definidas *ad hoc* e o HfDAQL.
Fonte: Autor



(a) Caminho percorrido pelo robô utilizando o *Q-Learning*



(b) Caminho percorrido pelo robô utilizando o HAQL (ad hoc)



(c) Caminho percorrido pelo robô utilizando o HfDAQL

Figura 6.24 – Caminhos percorridos pelo robô no 1^o episódio de treinamento com o uso dos algoritmos estudados.
Fonte: Autor

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho propôs o uso de demonstrações para a obtenção das heurísticas utilizadas na aceleração do Aprendizado por Reforço. Os resultados obtidos indicam que o algoritmo proposto, denominado "Q-Learning Acelerado por Heurísticas obtidas por meio de Demonstrações" (*Heuristics from Demonstration to Accelerate Q-Learning* – HfDAQL), foi mais eficiente do que os algoritmos *Q-Learning* e HAQL com heurísticas definidas *a priori*.

O HfDAQL demonstrou melhor desempenho nos dois domínios propostos, ou seja, no Mundo de Grades e na simulação realizada no *software* Player/Stage, principalmente no início do aprendizado, onde a quantidade de passos necessários para se encontrar o objetivo foi menor. Isto se deve ao fato do HfDAQL limitar o espaço de busca explorado pelo agente robótico aprendiz, pois a heurística indica o melhor caminho a seguir.

A experiência conduzida no Mundo de Grades teve como principal objetivo avaliar o funcionamento do algoritmo proposto em um ambiente menos complexo e, relativamente, pequeno. Os resultados apresentados neste domínio foram satisfatórios e apresentaram o comportamento esperado para todos os algoritmos utilizados: *Q-Learning*, HAQL com heurísticas definidas *a priori (ad hoc)* e o HfDAQL.

Por sua vez, os experimentos realizados no ambiente simulado pelo *software* Player/Stage tinham como meta a análise dos mesmos algoritmos citados no parágrafo anterior em um domínio mais complexo e extenso, com agente não-determinístico e dinâmico.

Para se implementar o algoritmo HfDAQL neste ambiente simulado, foi necessária, primeiramente, a realização de uma análise sobre a influência das demonstrações na aceleração do aprendizado. Assim, foi realizada uma avaliação sobre como a quantidade e a qualidade das demonstrações, conduzidas durante a obtenção das heurísticas, afetam o aprendizado.

Este estudo indicou que as demonstrações são mais eficazes quando os caminhos demonstrados ao agente aprendiz o levam diretamente até o estado objetivo e que a quantidade destas demonstrações deve ser a necessária para cobrir a maior área possível do meio no qual o robô vai se movimentar. Quanto maior a área livre coberta por demonstrações de caminhos ótimos ou subótimos, mais eficaz o algoritmo será na aceleração do aprendizado.

Após a análise das demonstrações e da implementação dos três algoritmos estudados no ambiente simulado, pode-se concluir, pela avaliação dos experimentos realizados, que:

a) O *Q-Learning* levou o robô ao aprendizado, porém de forma muito lenta e com bastante oscilação nos resultados obtidos. Como o *Q-Learning* demora muito para aprender, as experiências realizadas no ambiente simulado pelo Player/Stage consumiram sempre muito tempo.

b) O HAQL com heurística definida de maneira *ad hoc* apresentou uma melhora considerável no tempo de convergência para o aprendizado quando comparado ao *Q-Learning*, principalmente nos primeiros episódios.

c) O HfDAQL fez com que o agente aprendiz já iniciasse o aprendizado muito próximo da condição ótima, sendo ainda mais rápido do que o HAQL na aprendizagem.

Porém, é importante citar que, no HfDAQL, o robô pode passar por um processo na qual o número de passos necessários para se atingir o objetivo pode aumentar durante o aprendizado. Este fato acontece quando o robô começa a perder a influência da heurística, o que o leva a visitar e aprender outros pares estado-ação desconhecidos, conforme explicado na seção 6.2.4.

Algumas estratégias foram experimentadas de maneira superficial para tentar diminuir esta piora na curva de aprendizado, como por exemplo o aumento no valor de H, o uso de um fator de decaimento para a heurística, a combinação das duas técnicas anteriores e a diminuição no valor de H, porém todas estas abordagens somente deslocaram o evento do aumento do número de passos mais para o final ou mais para o começo do aprendizado. Assim, este fato mostrou-se necessário para que o aprendizado completo sobre o ambiente acontecesse.

Os resultados obtidos nos experimentos permitem concluir que o algoritmo proposto, HfDAQL, acelerou o aprendizado quando comparado ao *Q-Learning* e ao HAQL com heurística definida *a priori*. Além disso, o uso das demonstrações para a obtenção das heurísticas podem tornar mais intuitva a inserção de conhecimento sobre o domínio no algoritmo de aprendizado do robô aprendiz.

7.1 Contribuições

A proposta realizada neste trabalho é um método de aprendizado eficiente para o domínio da robótica móvel, pois, por intermédio da junção do Aprendizado por Reforço Acelerado por Heurísticas (BIANCHI, 2004) com o Aprendizado por Demonstração (SCHAAL, 1997), que resultou no HfDARL, pôde-se alcançar as principais vantagens provenientes dos dois métodos de aprendizado.

O HfDARL tem como principais vantagens:

a) A possibilidade de eliminar a necessidade do programador conhecer em detalhes o funcionamento dos sensores do robô, já que o conhecimento prévio do domínio é inserido na aprendizagem por meio de demonstrações.

b) A baixa quantidade de demonstrações necessárias para acelerar consideravelmente o aprendizado.

c) O número reduzido de passos para se atingir a meta, desde os primeiros episódios do aprendizado.

As principais contribuições deste trabalho foram:

a) A proposta da classe HfDARL, implementada por meio do algoritmo HfDAQL (capítulo 5).

b) A proposta de se realizar o espalhamento das heurísticas utilizadas no HfDARL, como forma de diminuir a quantidade de demonstrações necessárias (seção 5.1).

c) O estudo dos algoritmos *Q-Learning*, HAQL com heurísticas definidas *a priori (ad hoc)* e HfDAQL no ambiente simulado pelo *software* Player/Stage (seção 6.2).

d) O estudo da influência da quantidade e qualidade das demonstrações no aprendizado realizado com o algoritmo HfDAQL (seção 6.2.4).

7.2 Trabalhos Futuros

Alguns trabalhos futuros derivados deste trabalho se mostram de grande interesse, como por exemplo:

a) Estudar o comportamento do algoritmo HfDAQL em um ambiente ainda mais complexo, por meio da implementação do algoritmo em um robô real.

b) Ainda no domínio da navegação robótica pode-se analisar o HfDAQL em ambientes mais dinâmicos, com objetos móveis, por exemplo.

c) Estudar outras formas e funções de espalhamento para a função $\mathcal{H}$, além da função QSx adaptada, utilizada neste trabalho.

d) Verificar a influência de demonstrações extremamente ruins, do tipo que não levam o robô até o estado meta, no aprendizado.

REFERÊNCIAS

ALBUS, J. S. A new approach to manipulator control: The cerebellar model articulation controller (cmac). **Journal of Dynamic Systems, Measurement, and Control**, USA, v. 97, n. 3, p. 220–227, 1975.

ARGALL, B. D. et al. A survey of robot learning from demonstration. **Robot. Auton. Syst.**, Amsterdam, The Netherlands, v. 57, p. 469–483, May 2009.

BELLMAN, R. **Applied Dynamic rogramming**. Princeton, NJ: Princeton University, 1957.

BERTSEKAS, D. P. **Dynamic Programming: Deterministic and Stochastic Models**. Upper Saddle River, NJ: Prentice-Hall, 1987.

BIANCHI, R. A. C. **Uso de heurísticas para a aceleração do aprendizado por reforço**. 2004. 174 f. Tese (Doutorado em Engenharia Elétrica) — Universidade de São Paulo, São Paulo.

BIANCHI, R. A. C.; ROS, R.; MANTARAS, R. L. D. Improving reinforcement learning by using case based heuristics. In: INTERNATIONAL CONFERENCE ON CASE-BASED REASONING. **Proceedings of...** Berlin, 2009. p. 75–89.

CELIBERTO JR., L. A. **Aprendizado por Reforço Acelerado por Heurísticas Aplicado ao Domínio do Futebol de Robôs Simulados**. 2007. 122 f. Dissertação (Mestrado em Engenharia Elétrica) — Centro Universitário da FEI, São Bernardo do Campo.

DELLAERT, F. et al. Monte carlo localization for mobile robots. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA99). **Proceedings of...** Detroit, Michigan, USA, 1999.

DRUMMOND, C. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. **Journal of Artificial Intelligence Research**, USA, v. 16, p. 59–104, 2002.

FOSTER, D.; DAYAN, P. Structure in the space of value functions. **Machine Learning**, USA, v. 49, n. 2/3, p. 325–346, 2002.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**: Data mining, inference, and prediction. 2nd. ed. USA: Springer, 2001.

KAELBLING, L.; CASSANDRA, A.; KURIEN, J. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In: IEEE/RSJ INTERNATIONAL

CONFERENCE ON INTELLIGENCE ROBOTS AND SYSTEMS. **Proceedings of...** Los Alamitos, CA, USA, 1996.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of Artificial Intelligence Research**, USA, v. 4, p. 237–285, 1996.

KUNIYOSHI, Y. et al. Cooperation by observation - the framework and basic task patterns. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION - ICRA. **Proceedings of...** San Diego, California, USA, 1994. p. 767–774.

LITTMAN, M. L. Markov games as a framework for multi-agent reinforcement learning. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING. **Proceedings of...** New Jersey, USA, 1994. p. 157–163.

MARTINS, M. F. **Aprendizado por Reforço Acelerado por Heurísticas Aplicado ao Domínio do Futebol de Robôs**. 2007. 102 f. Dissertação (Mestrado em Engenharia Elétrica) — Centro Universitário da FEI, São Bernardo do Campo.

MITCHELL, T. **Machine Learning**. New York: McGraw Hill, 1997.

NEHANIV, C. L.; DAUTENHAHN, K. The correspondence problem. In: _____**. Imitation in animals and artifacts**. Cambridge, MA, USA: MIT, 2002. p. 41–61.

NEHMZOW, U. **Scientific Methods in Mobile Robotics**: quantitative analysis of agent behaviour. New York: Springer-Verlag, 2006.

OWEN, J. **How to use Player/Stage**. York, April 2010. Disponível em: <http://www-users-.cs.york.ac.uk/~jowen/player/playerstage-manual.html>. Acesso em: 15 out. 2011.

PEGORARO, R. **Agilizando aprendizagem por reforço em robótica móvel através do uso de conhecimento sobre o domínio**. 2001. 106 f. Tese (Doutorado em Engenharia Elétrica) — Universidade de São Paulo, São Paulo.

PERICO, D. H.; BIANCHI, R. A. C. Uso de heurísticas obtidas por meio de demonstrações para aceleração do aprendizado por reforço. In: X SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE (SBAI). **Anais...** São João del-Rei, MG: SBA, 2011. p. 851–856.

RIBEIRO, C. H. C. On the use of option policies for autonomous robot navigation. In: INTERNATIONAL JOINT CONFERENCE, 7TH IBERO-AMERICAN CONFERENCE ON AI: ADVANCES IN ARTIFICIAL INTELLIGENCE. **Proceedings of...** London, UK: Springer Verlag, 2000. p. 369–378.

RIBEIRO, C. H. C.; SZEPESVARI, C. Q-learning combined with spreading: Convergence and results. In: ISRF-IEE INTERNATIONAL CONFERENCE ON INTELLIGENT AND COGNITIVE SYSTEMS – NEURAL NETWORKS SYMPOSIUM. **Proceedings of...** Tehran, Iran, 1996. p. 32–36.

RUMMERY, G.; NIRANJAN, M. **On-line Q-learning using connectionist systems**. Cambridge, UK, 1994. University of Cambridge. Disponível em: <http://mi.eng.cam.ac.uk-/reports/svr-ftp/auto-pdf/rummery_tr166.pdf>. Acesso em: 20 fev. 2011.

RUSSELL, S.; NORVIG, P. **Inteligência Artificial**. 2. ed. Rio de Janeiro: Elsevier, 2004.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. **IBM Journal of Research and Development**, v. 3, n. 3, p. 210–229, July 1959.

SCHAAL, S. Learning from demonstration. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. **Proceedings of...** Cambridge, MA, USA: MIT, 1997.

SHAKESPEARE, W. **Mr. William Shakespeares Comedies, Histories & Tragedies (First Folio)**. Londres: [s.n.], 1623.

SMART, W. D.; KAELBLING, L. P. Effective reinforcement learning for mobile robots. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION. **Proceedings of...** Washington, D.C., USA: IEEE, 2002. p. 3404–3410.

SPIEGEL, M. R. **Estatística**. 2. ed. São Paulo: McGraw-Hill, 1984.

SUTTON, R. S. Learning to predict by the methods of temporal differences. **Machine Learning**, Hingham, MA, USA, v. 3, 1988.

SUTTON, R. S. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING. **Proceedings of...** Austin, TX, USA, 1990.

SUTTON, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. **Proceedings of...** Amherst, MA, USA: MIT, 1996. v. 8, p. 1038–1044.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. Cambridge, MA, USA: MIT, 1998.

WATKINS, C. J. C. H. **Learning from Delayed Rewards**. 1989. 234 f. Tese (Doutorado) — King's College, Cambridge, UK.

APÊNDICE - O TESTE T DE STUDENT

O teste t de Student (SPIEGEL, 1984), conhecido também como *T-Test*, fornece a ferramenta para poder verificar se os resultados de duas experiências são significativamente diferentes. Esta ferramenta é muito utilizada em todas as ciências e permite decidir se um algoritmo exibe um desempenho superior em comparação a outro.

Para poder estabelecer se dois valores médios, chamados de μ_x e μ_y, são diferentes, o valor T é calculado segundo a equação A.1.

$$T = \frac{\mu_x - \mu_y}{\sqrt{(n_x - 1){\sigma_x}^2 + (n_y - 1){\sigma_y}^2}} \sqrt{\frac{n_x n_y (n_x + n_y - 2)}{n_x + n_y}} \tag{A.1}$$

Onde:

- n_x e n_y são os números de pontos;
- μ_x e μ_y são as médias;
- σ_x e σ_y são os desvios padrões.

Se o valor do módulo de T for maior que a constante t_α a hipótese H_0 que $\mu_x = \mu_y$ é rejeitada e os experimentos são diferentes significativamente. O valor t_α, constante que pode ser encontrada na maior parte dos livros de estatística, define a probabilidade de um teste estar errado por meio de um nível de confiança, que, na maioria dos casos, é definido como 5%.