Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Aprendizado por reforço em modelos probabilísticos de redes imunológicas para robótica autônoma

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Autor: Alisson Gusatti Azzolini
Orientador: Fernando José Von Zuben

Campinas, SP
2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Azzolini, Alisson Gusatti
Az64a Aprendizado por reforço em modelos probabilísticos de redes imunológicas para robótica autônoma / Alisson Gusatti Azzolini. – Campinas, SP: [s.n.], 2011.

Orientador: Fernando José Von Zuben.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Aprendizado do computador. 2. Sistemas inteligentes de controle. 3. Robôs móveis. 4. Sistemas de veículos auto-guiados. I. Von Zuben, Fernando José. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês:	Reinforcement learning in probabilistic models of immune networks for autonomous robotics
Palavras-chave em Inglês:	Machine learning, Intelligent control systems, Mobile robots, Automated guided vehicle systems
Área de concentração:	Engenharia de Computação
Titulação:	Mestre em Engenharia Elétrica
Banca Examinadora:	Maurício Fernandes Figueiredo, Wagner Caradori do Amaral
Data da defesa:	13/05/2011
Programa de Pós Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Alisson Gusatti Azzolini

Data da Defesa: 13 de maio de 2011

Título da Tese: "Aprendizado por reforço em modelos probabilísticos de redes imunológicas para robótica autônoma"

Prof. Dr. Fernando José Von Zuben (Presidente): Fernando José Von Zuben

Prof. Dr. Maurício Fernandes Figueiredo:

Prof. Dr. Wagner Caradori do Amaral:

Resumo

Há uma demanda crescente por soluções avançadas de navegação autônoma em robótica móvel. Apresenta-se então um sistema de síntese e aprendizagem de controladores com tal finalidade. Propõe-se um controlador probabilístico, consistindo no acoplamento de um processo de decisão de Markov parcialmente observável (POMDP) com um classificador logístico multinomial. A parametrização empregada para o POMDP inspira-se numa proposta anterior de controle de robô por meio de redes imunológicas artificiais, que mostrou apresentar flexibilidade e capacidade de representação de conhecimento na execução de tarefas desafiadoras de navegação autônoma. A aprendizagem dos parâmetros do classificador logístico é efetuada através de um algoritmo de aprendizagem por reforço baseado em gradiente de política, e os do POMDP, atráves de um algoritmo de maximização de verossimilhança. Três experimentos computacionais são efetuados, dois deles utilizando somente o classificador logístico, e o terceiro utilizando o acoplamento entre POMDP e classificador logístico. Os resultados permitem a constatação de pontos fortes e algumas deficiências das duas abordagens. O trabalho aponta também para uma potencial reinterpretação do controlador baseado em rede imunológica em termos de um modelo probabilístico similar ao proposto.

Palavras-chave: Aprendizado do computador, Sistemas inteligentes de controle, Robôs móveis, Sistemas de veículos auto-guiados.

Abstract

There is an increasing demand for advanced solutions in autonomous navigation of mobile robots. A system is presented for the synthesis and learning of controllers for such purpose. A probabilistic controller is proposed, consisting of the coupling of a partially observable Markov decision process (POMDP) with a multinomial logistic classifier. The parametrization used for the POMDP draws on an earlier proposal of robot control based on artificial immune networks, that has shown to present flexibility and knowledge representation capability in the execution of challenging autonomous navigation tasks. Learning the logistic classifier parameters is accomplished through a reinforcement learning algorithm based on policy gradient, while the POMDP parameters are learned by a likelihood maximization algorithm. Three computational experiments are performed, two of them using only the logistic classifier, and the third one using the coupling of a POMDP with a logistic classifier. The results show some strong points and drawbacks of both approaches. The work also points torwards a potential reinterpretation of the immune network based controller in terms of a probabilistic model similar to the one proposed.

Keywords: Machine learning, Intelligent control systems, Mobile robots, Automated guided vehicle systems.

Agradecimentos

A meu orientador, Fernando José Von Zuben, por me ajudar a dar os primeiros passos nesse imenso universo acadêmico, no qual é muito fácil se perder.

A meu irmão Anderson, e aos meus pais Iracema e Ortenilo, que, apesar da distância, continuam sendo meu porto seguro.

Ao pessoal do laboratório LBiC, pelo infindável companheirismo e pelas infindáveis e produtivas discussões, que teriam, por si só, feito valer esses dois anos de trajetória.

Aos meus colegas de apartamento, que ajudaram a trazer um ambiente familiar ao lugar onde morávamos.

À FAPESP, pelo importante financiamento fornecido ao longo deste período.

À minha família.

Sumário

Lista de Figuras

Lista de Símbolos

x_i - i-ésimo estado em um conjunto finito de estados. 81
u_i - i-ésima ação em um conjunto finito de ações. 81
A_w - Matriz de probabilidades de transição sob ação u_w 81
a_{ij}^w - Probabilidade de transição para estado x_j, a partir de x_i sob ação u_w 81
μ_i^k - Média da k-ésima dimensão de observação no estado x_i 84
σ_i^k - Desvio padrão da k-ésima dimensão de observação no estado x_i 84
η_i^k - Coeficiente de mistura da k-ésima dimensão de observação no estado x_i 84

ϕ_t - Vetor de entrada do algoritmo de aprendizado por reforço no tempo t 111
d_t^k - Distância a obstáculo detectada pelo k-ésimo sonar no tempo t 109
a_t^g - Área do blob verde no tempo t 109
a_t^r - Área do blob laranja no tempo t 109
x_t^g - Posição horizontal do blob verde no tempo t 109
x_t^r - Posição horizontal do blob laranja no tempo t 109

Capítulo 1

Introdução

O presente estágio da evolução social e tecnológica apresenta novos e grandes desafios à ciência e à engenharia. O ideal de uma máquina independente do homem, capaz de sobreviver por si só ou de resolver um certo número de problemas existe desde a antiguidade[1], assim como a busca pelo entendimento da mente humana e sua ligação com o corpo[2]. Mas foi apenas após a Revolução Industrial (Sharkey & Sharkey, 2009), e principalmente após o surgimento dos computadores digitais, que tais buscas deixaram de ser temas de filosofia e passaram a integrar programas de pesquisa científicos e de engenharia. Inteligência artificial (Russell & Norvig, 2003), cibernética (Wiener, 1948), controle autônomo (Antsaklis & Passino, 1993) são temas da engenharia que surgiram nas últimas seis décadas, tentando, de uma forma ou outra, dotar as máquinas de atributos humanos e, consequentemente, de autonomia. O estudo da mente, apesar de inerentemente filosófico, ganha cada vez mais apoio da psicologia, da neurociência e da computação, disciplinas que juntas compõem o programa de pesquisa da ciência cognitiva (Kellogg, 2007).

Porém, nas últimas duas ou três décadas, apesar de a capacidade de computação ter aumentado exponencialmente em termos de operações matemáticas realizadas por segundo, o computador continua ignorante e pouquíssimo autônomo quando confronta problemas em que humanos são naturalmente bons, por exemplo, em situações nas quais há interação com o ambiente físico. O problema de navegação é uma das tarefas que o homem executa com excelência, mas que um robô tem dificuldade em exercer.

Da mesma forma, as ciências cognitivas ainda estão longe de compreender inteiramente a cognição humana. Apesar de a neurociência mapear com cada vez mais precisão as áreas do cérebro ativas

[1] Na mitologia grega, Hefesto, o deus da siderurgia, teria construído, a pedido de Zeus, um homem mecânico de bronze chamado Talos (Russell & Norvig, 2003, p. 939). Historicamente, há evidências de autômatos programáveis criados pelo engenheiro grego Hero por volta de 60 A.C. (Sharkey, 2007).

[2] Tentativas de entender a mente e sua forma de operar remontam aos antigos gregos, quando filósofos como Platão e Aristóteles tentaram explicar a natureza do conhecimento humano (Thagard, 2010).

em cada atividade realizada, ainda não é possível saber com exatidão como as regiões interagem para executar as funções. Há um abismo entre descrições comportamentais de alto nível da psicologia e as descrições de ativação neuronal da neurociência. Além disso, muitos dos problemas estão ainda em um nível altamente especulativo e filosófico, como o problema da consciência (Gulick, 2004).

1.1 Proposta

Diante do exposto, o presente trabalho tem um duplo objetivo. Por um lado, visa contribuir no campo da inteligência artificial, mais especificamente no desenvolvimento de agentes autônomos, apresentando formas de síntese de controladores para navegação em robôs móveis. Por outro lado, de forma mais especulativa, tenta contribuir junto à ciência cognitiva, propondo modelos que poderiam ser válidos para explicar certos aspectos da inteligência humana.

É proposto, neste trabalho, um sistema de síntese e aprendizagem de controladores voltados à execução de tarefas de navegação em robótica móvel. No desenvolvimento de um tal sistema, há duas escolhas primordiais a serem feitas: 1) arquitetura do controlador, 2) método de síntese e aprendizagem dos controladores.

Primeiramente, quanto ao modelo de controlador, as diferentes arquiteturas presentes na literatura podem ser classificadas em dois extremos: de um lado, as arquiteturas clássicas, que exigem grande conhecimento do ambiente pelo controlador, e, de outro lado, as propostas comportamentais, que defendem controladores completamente desprovidos de modelo do ambiente. Tais propostas levam, respectivamente, a controladores deliberativos e reativos.

Em vista destas duas abordagens antagônicas, toma-se como critério principal deste trabalho o desenvolvimento de um modelo de controlador que seja intermediário entre o paradigma clássico e o comportamental. Como consequência, devem-se conciliar arquiteturas reativas e deliberativas em um só mecanismo híbrido de controle. Mais especificamente, busca-se inspiração na arquitetura de controlador de Cazangi (2008), baseada no paradigma de redes imunológicas artificiais, que mostrou-se capaz de apresentar comportamentos deliberativos devido às interconexões da rede.

Quanto ao método de síntese e aprendizagem do controlador, busca-se inspiração nos sistemas classificadores de Holland (1975) e nos algoritmos de aprendizagem por reforço (Sutton & Barto, 1998). Tais métodos levam a sistemas capazes de aprender através da experiência recebida do ambiente, de forma incremental, associativa, e visando à maximização de uma função-objetivo projetada especificamente para a tarefa em questão.

Finalmente, busca-se enquadrar todo o projeto, incluindo modelo de controlador e algoritmos de aprendizagem, em um único formalismo matemático. Adota-se então a teoria de probabilidades, que se adequa bem às situações de incerteza vivenciadas por um robô móvel autônomo. Assim, o modelo

proposto de controlador consiste no acoplamento de dois modelos probabilísticos: um processo de decisão de Markov parcialmente observável (POMDP, do inglês *Partially Observable Markov Decision Process*) e um classificador logístico multinomial. Tais modelos são genéricos o suficiente para que possam ser parametrizados levando em conta os critérios apresentados no parágrafo anterior.

Da mesma forma, os métodos usados para a aprendizagem do controlador consistem em algoritmos de estimação de modelos probabilísticos. Mais especificamente, emprega-se um algoritmo de gradiente de política, que se enquadra no paradigma de aprendizado por reforço, para o aprendizado do classificador logístico, e um algoritmo de maximização de verossimilhança para o aprendizado do POMDP.

1.2 Organização do texto

A dissertação é constituída de sete capítulos, o primeiro sendo o capítulo introdutório. Nos capítulos 2 ao 5, são apresentados os tópicos que constituirão os fundamentos do trabalho realizado. As contribuições concentram-se nos capítulos 6 e 7.

O **capítulo 2** discorre de maneira geral a respeito do problema de navegação em robótica móvel, grande área em que o trabalho se situa. Primeiramente, são expostos rudimentos da mecânica de robôs móveis. Em seguida, são apresentadas em detalhes as propostas clássica e comportamental, e as estratégias reativa e deliberativa de controle de robôs móveis.

No **capítulo 3**, são apresentados o paradigma de sistemas classificadores de Holland (1975) e a abordagem imuno-inspirada de Cazangi (2008). Esta última, em particular, será posteriormente tomada como inspiração na construção de um modelo probabilístico de controlador.

Os dois capítulos seguintes buscam na matemática e na inteligência artificial um ferramental matemático que possa ser usado para modelar, sintetizar e realizar o aprendizado de controladores para robótica móvel. Assim, no **capítulo 4** é apresentado o conceito de Processos de Decisão de Markov e o modelo de aprendizagem por reforço (Sutton & Barto, 1998), da área de inteligência artificial. No **capítulo 5**, por sua vez, é apresentado o formalismo de POMDPs (Kaelbling et al., 1998; Lovejoy, 1991; Monahan, 1982). Em ambos os capítulos, são expostos, em linhas gerais, algoritmos de aprendizagem, que serão posteriormente ajustados para o problema de navegação.

Em seguida, no **capítulo 6**, os modelos gerais expostos nos capítulos 4 e 5 são parametrizados de forma específica para o problema em questão, inspirando-se nas redes imuno-classificadoras do capítulo 3. Assim, inicialmente, é proposta uma parametrização de POMDP que se assemelha em forma e funcionalidade a uma rede imuno-classificadora. Além disso, propõe-se acoplar tal POMDP a um classificador logístico, responsável por selecionar as ações do robô. Dessa forma, é especificado em detalhes um modelo de controlador de robôs móveis que seja condizente com os critérios

estabelecidos para o projeto.

Ainda no capítulo 6, de posse de uma parametrização do modelo de controlador, procede-se a uma especificação dos algoritmos de síntese e aprendizagem de tal modelo, que serão responsáveis por, a partir da experiência do robô, estimar os parâmetros. Cada parte do controlador é treinada de uma forma: para o POMDP, é utilizado um algoritmo de maximização de verossimilhança, e para o classificador logístico, utiliza-se um algoritmo de gradiente de política, que consiste numa forma de algoritmo de aprendizagem por reforço.

Finalmente, no **capítulo 7**, de posse de um modelo de controlador e de um método de aprendizagem, são efetuados três experimentos simulados para validar a proposta do capítulo 6. Os resultados dos experimentos explicitam pontos fortes e deficiências do sistema proposto. Dessa forma, é possível esboçar um plano de contribuições futuras que deem sequência à proposta, o qual compõe o **capítulo 8**, juntamente com um destaque para as principais contribuições alcançadas pela pesquisa.

Capítulo 2

Robótica Móvel Autônoma

Atualmente, robôs são omnipresentes nas linhas de produção de artigos tecnológicos. São robôs que montam os circuitos integrados e placas de circuito utilizadas em qualquer aparelho eletrônico disponível no mercado. A indústria automotiva é outro exemplo de uso extensivo de robôs em suas linhas de produção. Essa grande penetração da robótica em ambientes de produção industriais é protagonizada em sua maior parte pelas garras móveis. São robôs altamente especializados, geralmente voltados para atividades pré-programadas, e cujo alcance de ação se restringe a alguns metros. Tais restrições permitem que modelos cinemáticos e dinâmicos sejam concebidos a fim de otimizar a performance de cada garra, alcançando capacidades sobre-humanas e agilizando a linha de produção.

O presente trabalho, porém, trata de outro tipo de robô: os robôs móveis. A princípio, é questionável a utilidade de um robô móvel em uma linha de produção. Robôs fixos e especializados são capazes de utilizar toda sua capacidade para executar uma tarefa repetitiva de forma ótima. Há casos, porém, em que a robótica móvel é desejável e mesmo necessária.

O uso mais evidente de um robô móvel é a exploração e a realização de tarefas em ambientes inóspitos, como por exemplo o fundo dos oceanos, desertos ou o solo de Marte. Nesses ambientes inacessíveis a seres humanos, robôs apresentam uma grande vantagem, devido às suas características físicas. Um robô pode ser projetado de forma a aproveitar características do ambiente para uma melhor locomoção. Por exemplo, no fundo do oceano, hélices ou nadadeiras são apropriadas. Já em ambientes desérticos com obstáculo, pode-se optar por múltiplas rodas. Além disso, sistemas de captação e armazenagem de energia podem ser projetados de forma específica, por exemplo paineis solares nos robôs de exploração de Marte.

A princípio, tais robôs permitem a realização de tarefas em ambientes inóspitos através da teleoperação. Em arquiteturas mais simples, um operador remoto pode controlar cada atuador e sensor do robô para realizar a tarefa desejada. Em robôs mais sofisticados, porém, que dispõem de algumas dezenas ou centenas de atuadores e múltiplos sensores, é impossível para um operador humano

controlar cada atuador. Nesses robôs, deve haver um mecanismo de controle automático que seja capaz de mobilizar conjuntamente sensores e atuadores para realizar as ordens do operador. Além disso, deve haver sistemas de informação que processem ao máximo os dados obtidos pela miríade de sensores de forma a dotar o operador de toda informação necessária à tomada de decisões envolvida no comando do robô. Nesses casos, a tarefa básica de controle é automatizada, mas a cognição e a tomada de decisão continuam a cargo de um humano.

Seguindo além dos sistemas automáticos teleoperados descritos acima, é possível vislumbrar robôs móveis cada vez mais autônomos. Por exemplo, no caso do robô explorador de Marte, é necessário que o robô seja autônomo pelo menos em uma escala de tempo de alguns minutos, dado que um comando de um operador em terra só chega em Marte minutos depois. Assim, o robô deve possuir pelo menos algum sistema de controle autônomo que permita atender objetivos de médio prazo estabelecidos pelo operador humano.

Finalmente, pode-se pensar em robôs completamente autônomos, ou seja, que são capazes de realizar suas tarefas sem intervenção humana, desde o instante em que é ligado até o momento em que haja necessidade de manutenção. Tais robôs poderiam ser úteis não apenas em locais inóspitos, como os citados anteriormente, mas também em ambientes ocupados por humanos, como escritórios, ruas e hospitais. Nesse caso, tais robôs podem ser vistos como ajudantes que, apesar de realizar o pedido de humanos, são capazes de se manter em operação sem intervenção humana. Por exemplo, tais robôs devem ser capazes de recarregar suas baterias por si só, quando necessário.

Robôs autônomos devem possuir objetivos de longo prazo e ser dotados de uma capacidade de tomada de decisão que leve à realização de tais objetivos. No caso de robôs móveis autônomos, por exemplo, um importante objetivo é a navegação.

Isto dito, um dos principais mercados de robôs móveis atualmente são laboratórios de estudo de inteligência e cognição. Um robô móvel autônomo enfrenta muitos dos desafios que animais, incluindo seres humanos, enfrentam e para os quais desenvolveram habilidades cognitivas ao longo da evolução. Dessa forma, robôs móveis podem ser uma boa maneira de testar modelos biológicos de inteligência e de levar à descoberta de princípios mais fundamentais da cognição.

O presente capítulo, baseado na exposição de Siegwart & Nourbakhsh (2004), aborda a mobilidade de forma geral. São apresentados os elementos físicos e lógicos necessários para que um robô seja móvel. O problema de navegação é apresentado do ponto de vista da disciplina de robótica móvel, sendo que a metodologia utilizada por tal disciplina é explicitada. Finalmente, as principais arquiteturas de sistemas de navegação são apresentadas e classificadas.

2.1 Elementos de um robô móvel

Antes de discutir sobre aspectos da navegação em robôs móveis, é imprescindível a apresentação de alguns elementos básicos necessários a qualquer robô capaz de se locomover autonomamente em seu ambiente. Tais elementos são locomoção, percepção e localização.

2.1.1 Locomoção e cinemática

Para se locomover, o robô móvel precisa de mecanismos apropriados. A natureza fornece inspiração de várias maneiras: através de quatro, seis ou oito patas ou duas pernas, animais podem correr, saltar ou caminhar. Outros usam o próprio corpo ou centenas de pequenas patas para rastejar ou deslizar pelo solo.

O mecanismo mais utilizado pelo homem na construção de veículos e em robôs móveis, porém, não é inspirada na natureza: trata-se da roda. A roda é altamente eficiente em ambientes de terreno plano e rígido. À medida que o piso se torna mais suave e irregular, as rodas perdem sua eficiência em relação a patas ou pernas, devido ao aumento do atrito de rolagem. Além disso, em ambientes altamente irregulares, com degraus ou depressões, as rodas simplesmente não resolvem o problema, já que ficam facilmente presas em obstáculos.

Ocorre que a construção de robôs com pernas é um grande desafio para a engenharia, ainda não totalmente superado. Em geral, pernas possuem um número elevado de graus de liberdade, o que leva a uma maior complexidade mecânica e de controle. Tal complexidade é possível em sistemas biológicos, que utilizam blocos microscópicos e baseiam-se fortemente na capacidade de replicação. Assim, animais inferiores com centenas de minúsculas patas são encontrados na natureza. Para a engenharia atual, porém, um sistema de locomoção semelhante seria custoso e ineficiente (ou mesmo impossível), tanto pelo número de elementos quanto pelo seu tamanho.

No caso de agentes bípedes, há ainda o problema do equilíbrio, que impõe não só limitações no formato geral do corpo do agente, mas também exige um mecanismo sofisticado de controle. Além disso, agentes bípedes naturais, como o ser humano, são capazes de correr. Numa corrida, há instantes em que nenhum dos dois pés se encontra em contato com o chão. Um robô bípede que pudesse correr necessitaria atuadores capazes de aplicar força suficiente para levantar seu centro de gravidade, removendo os dois pés do chão e, em seguida, deveria aguentar o forte choque de um dos pés contra o chão, dentre outros desafios de projeto.

Devido à maior simplicidade do mecanismo e à grande utilização em robôs já existentes, este trabalho foca na locomoção por rodas.

Locomocão por rodas

Mesmo restringindo-se à locomoção por rodas, há um amplo espaço de *design* que envolve o tipo de rodas e a configuração das rodas no robô. A escolha baseia-se principalmente no ambiente ao qual o robô está inserido. Não há uma escolha que seja a melhor em todos os sentidos.

Há três fatores a se considerar numa escolha de *design*: estabilidade, manobrabilidade e controlabilidade.

A **estabilidade** consiste na capacidade do robô de manter sua posição e sua orientação quando parado (estabilidade estática) e quando em movimento (estabilidade dinâmica), sem tombar nem balançar mais do que o desejável. Surpreendentemente, um veículo com apenas duas rodas pode ser estaticamente estável, se seu centro de gravidade estiver abaixo do eixo das rodas. Com três rodas, o veículo é estaticamente estável se seu centro de gravidade estiver no interior do triângulo formado pelas três rodas. Mais do que três rodas aumentam a estabilidade, mas requerem sistema de suspensão para garantir a correta distribuição do peso do robô em cada roda.

Quanto à **manobrabilidade**, alguns robôs são omni-direcionais, ou seja, permitem mudança de posição em todas as direções e mudança instantânea de orientação. Na comunidade de pesquisa, em geral são utilizadas configurações de roda que não são completamente omni-direcionais, mas apresentam uma boa liberdade de movimento. O tipo mais conhecido é o robô *differential drive* de duas rodas, no qual as rodas giram em torno do centro do robô. Uma ou duas rodas adicionais são necessárias para dar estabilidade. Um exemplo de arquitetura menos manobrável é a Ackman de quatro rodas, encontrada em automóveis. Nesse caso, para fazer curvas, o veículo deve seguir uma circunferência de raio maior que o veículo.

A **controlabilidade** é definida como a capacidade do controlador de levar o robô a uma posição e estado desejados. Em geral, é inversamente proporcional à sua manobrabilidade. Quanto mais manobrável é um robô, mais graus de liberdade ele possui, o que exige um controlador mais sofisticado. Um robô *differential drive*, que pode girar em torno de seu próprio eixo, necessita de um controlador que imponha o mesmo perfil de velocidade às duas rodas para executar um percurso em linha reta. Uma arquitetura menos manobrável, como o Ackman de quatro rodas, efetua um trajeto em linha reta simplesmente travando-se a direção.

Neste trabalho, serão utilizados robôs diferenciais com duas rodas ativas e uma roda livre. A seguir, são introduzidos alguns aspectos da cinemática desse tipo de robô.

Cinemática do robô diferencial

Cinemática consiste no estudo mais básico do comportamento de sistemas mecânicos. A análise cinemática direta consiste na determinação do movimento que o robô efetua em um referencial global,

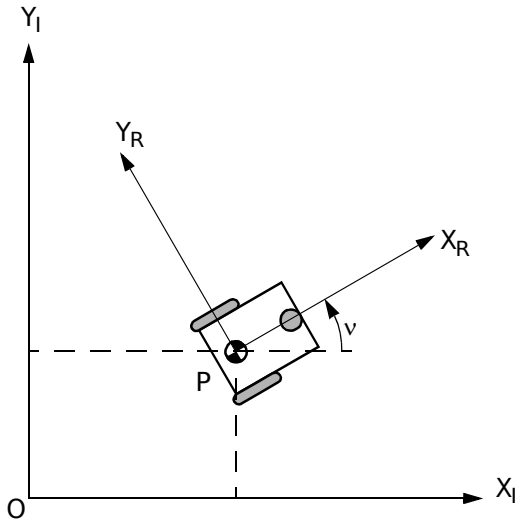


Fig. 2.1: Elementos de cinemática: referencial global e referencial do robô. Extraída de Siegwart & Nourbakhsh (2004, seção 3.2.1).

dada a geometria e o movimento das rodas (ou juntas) do robô. A cinemática reversa, em geral mais complexa, consiste em determinar o movimento de cada roda dado o movimento desejado do robô.

Controladores clássicos requerem o conhecimento da cinemática direta para manter uma estimação da posição do robô, e da cinemática reversa para comandar os atuadores do robô, dado um movimento desejado. Neste trabalho, os controladores desenvolvidos não lidam diretamente com a cinemática do robô. O robô deverá ser capaz de navegar mesmo sem o conhecimento exato de sua posição.

De qualquer forma, é interessante conhecer aspectos de cinemática, particularmente pelo fato de que os experimentos com os robôs serão realizados em ambientes virtuais simulados computacionalmente. A seguir, é apresentada a terminologia e notação em geral utilizada, e a cinemática direta dos robôs diferenciais é especificada.

O robô é modelado como um corpo rígido que se desloca em um plano horizontal. Como mostra a Fig. 2.1, definem-se dois referenciais. O primeiro, fixo no plano, é formado pelos eixos ortogonais $\{X_I, Y_I\}$ e uma origem O e chamado de referencial global. O referencial local do robô, por sua vez, é definido pelo ponto fixo P no chassis do robô, e a base $\{X_R, Y_R\}$. A posição P do robô é especificada pelas suas coordenadas (x, y) em relação ao referencial global, e a orientação relativa entre o referencial do robô e o referencial global é dada pelo ângulo θ. A **pose** do robô em relação ao referencial global I é então definida como:

$$\xi_I = (x, y, \theta). \tag{2.1}$$

A **velocidade** $\dot{\xi}_I$ do robô em relação ao referencial global corresponde à derivada temporal da posição:

$$\dot{\xi}_I = \frac{d\xi_I}{dt} = (\dot{x}, \dot{y}, \dot{\theta}). \tag{2.2}$$

Considere agora um robô diferencial com duas rodas, cada uma com diâmetro r. O ponto P é o ponto médio do eixo que liga as duas rodas. Cada roda se encontra a uma distância l do ponto P. Dados r, l e θ, e as velocidades de rotação $\dot{\phi}_1$ e $\dot{\phi}_2$ de cada uma das rodas, o modelo f de *cinemática direta* consiste na predição da velocidade do robô em relação ao referencial global:

$$\dot{\xi}_I = \mathrm{f}(l, r, \theta, \dot{\phi}_1, \dot{\phi}_2). \tag{2.3}$$

No caso do robô diferencial, o modelo cinemático direto pode ser expresso de forma simples:

$$\dot{\xi}_I = \mathrm{rot}(\theta)^{-1} \begin{bmatrix} \frac{r\dot{\phi}_1}{2} + \frac{r\dot{\phi}_2}{2} \\ 0 \\ \frac{r\dot{\phi}_1}{2} - \frac{r\dot{\phi}_2}{2} \end{bmatrix}, \tag{2.4}$$

onde $\mathrm{rot}(\theta)$ é a matriz de rotação em θ graus, expressa por:

$$\mathrm{rot}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.5}$$

2.1.2 Percepção

O objetivo da percepção é adquirir conhecimento a respeito do ambiente. Tal tarefa é essencial para qualquer sistema autônomo. Para tanto, o robô possui um conjunto de sensores cujas medidas são processadas de modo a proporcionarem informação a respeito do estado do ambiente ou do próprio agente.

Sensores podem ser classificados de diferentes formas. Primeiramente, quanto ao tipo de informação que proporcionam, são classificados em **proprioceptivos** quando medem valores internos ao robô (ex.: carga da bateria, velocidade das rodas, ângulo das juntas), e **exteroceptivos** quando medem valores externos (ex.: luminosidade, intensidade sonora, distância a obstáculos).

Por outro lado, quanto à forma de funcionamento, os sensores **passivos** apenas lêem a quantidade de energia que entra, enquanto que sensores **ativos** emitem energia e lêem a reação do ambiente à energia emitida. Sensores ativos em geral são mais precisos, mas podem alterar o ambiente em relação à variável lida e sofrem interferência de outros agentes que utilizem o mesmo tipo de sensor.

Finalmente, pode-se classificar os sensores pelo tipo de informação que proporcionam: sensores

táteis – detecção de contato físico ou proximidade; sensores de roda ou motor – informação sobre a velocidade ou posição da roda ou de eixos do motor; sensores de direção – detectam a orientação do robô em relação a um referencial fixo; sensores de baliza – detectam a posição do robô em relação a um referencial fixo no solo; sensores de distância ativos – detectam obstáculos por reflexão; sensores de movimento – velocidade relativa a objetos móveis; sensores baseados em visão – processam a imagem (segmentação, reconhecimento de objetos, etc).

Sensores de distância ativos

Por seu preço acessível e pela fácil interpretabilidade das medidas que efetuam, os sensores de distância ativos são ainda os mais populares em robótica móvel. Este será também o tipo de sensor utilizado neste trabalho, em um robô simulado. Apresentam-se aqui dois sensores baseados em *tempo de viagem*: o sensor ultrasônico — sonar — e o sensor *laser*.

Sensores de distância baseados em *tempo de viagem* recorrem ao tempo de propagação de uma onda, seja ela sonora ou eletromagnética. O sensor emite uma onda, que é refletida pela superfície de um obstáculo, e detecta a reflexão. O *tempo de viagem* (*flight-time*) corresponde ao tempo que a onda leva para ir até o obstáculo e retornar ao sensor, após reflexão. Conhecendo-se a velocidade de propagação da onda, é possível então determinar a distância ao obstáculo que causou a reflexão.

A qualidade de um sensor de distância baseado em *tempo de viagem* é limitada pelos seguintes fatores:

- abertura do cone de dispersão da onda emitida;
- interação com o obstáculo (absorção e reflexão especular da onda);
- variação da velocidade de propagação;
- velocidades do robô e do obstáculo;
- falta de precisão na determinação do *tempo de viagem*.

O **sonar** (*sound navigating and ranging*) é um sensor de distância por *tempo de viagem* que funciona emitindo uma onda ultrassônica de pressão em uma determinada direção. Conhecendo-se a velocidade de propagação do som no ar, é possível estimar a distância ao obstáculo a partir do tempo de reflexão. Tem-se:

$$d = \frac{c \cdot t}{2}, \tag{2.6}$$

onde d é a distância estimada ao obstáculo, c é a velocidade do som no ar e t é o tempo de viagem medido pelo sensor.

Entre as maiores limitações do sonar está a baixa resolução angular, ou seja, uma grande abertura do cone de onda impede que as medidas sejam interpretadas como pontos. É mais correto interpretá-las como arcos de circunferência. Isso significa também que, quanto mais longe o obstáculo detectado, menor a certeza quanto à sua localização angular.

Outra limitação importante do sonar é a interferência mútua: mesmo em um só robô, em geral utilizam-se múltiplos sensores do tipo sonar, cada um responsável pela detecção de obstáculos em uma determinada faixa angular. Porém, o sinal emitido por um sonar pode interferir em outro. Para evitar tal fenômeno, em geral faz-se a detecção em sequência, um sonar por vez. Isso, em contrapartida, diminui bastante a velocidade de detecção.

O sensor **laser** de distância, também conhecido como radar óptico ou *lidar — light detection and ranging —*, atinge importantes melhorias em relação ao sonar.

Devido à grande velocidade de propagação das ondas eletromagnéticas, o método direto de leitura do tempo de viagem requer o emprego de temporizadores com resolução de picosegundos, o que pode encarecer muito o componente. Um método muito mais simples corresponde à detecção de deslocamento de fase da onda.

No método de detecção por deslocamento de fase, uma onda de luz modulada em um comprimento de onda λ é emitida. A onda refletida é detectada com um deslocamento de fase θ. A distância ao obstáculo é então estimada como:

$$d = \frac{\lambda}{4\pi}\theta. \tag{2.7}$$

A resolução angular de sensores de distância laser excedem muito a dos sonares. Por exemplo, o laser Sick, comumente usado em robótica, possui resolução angular de 0,5 graus. A resolução de profundidade é de 5 cm, variando de 5 cm até 20 m.

Uma fonte de erro importante de sensores laser, não encontrada em sonares, é a impossibilidade de detectar obstáculos transparentes, que são comumente encontrados em uma série de ambientes, como museus, por exemplo.

2.1.3 Localização e mapeamento

Na resolução de um problem da navegação, em que o objetivo é dirigir-se a um determinado local no espaço, parece natural que o robô necessite ter pelo menos uma noção geral de onde está, ou seja, o robô deve ser capaz de se localizar. A localização está intimamente ligada a outro aspecto, o mapeamento. Criando um mapa do ambiente, o robô adquire conhecimento que pode ser útil para se localizar durante a navegação. Inversamente, para criar um mapa do ambiente, o robô deve ser capaz de se localizar.

A necessidade de um conhecimento explícito e acurado da posição, assim como a necessidade

de um mapa ou outra representação interna do ambiente, é tópico de debates na comunidade científica. Tradicionalmente, porém, a disciplina de robótica móvel costuma considerar a localização como um problema fundamental a ser resolvido no contexto de navegação. Além disso, é uma área relativamente bem formalizada e explorada, na qual foram feitos muitos progressos na última década.

Claramente, sensores e atuadores têm um papel fundamental na tarefa de localização. Os sensores recolhem características do ambiente que podem ser usadas como *landmarks* na localização. A cinemática dos atuadores, por sua vez, é usada no cálculo da odometria. Pode-se considerar então que a dificuldade de se obter a localização exata do robô vem de imperfeições em tais mecanismos. A seguir, são apresentadas três importantes fontes de incerteza quanto à localização.

- **imprecisão do sensor**: como apresentado na seção anterior, sensores estão sujeitos a diferentes tipos de imperfeições. Assim, para um dado estado do ambiente, deve-se esperar diferentes medidas. Num sonar, por exemplo, se o obstáculo estiver em um ângulo desfavorável em relação ao robô, apenas uma porcentagem das ondas refletidas será detectada. Uma forma de reduzir este problema é efetuar várias leituras em série (fusão temporal);

- ***sensor aliasing***: também chamada de *perceptual aliasing*, vem do fato que os sensores não são capazes de capturar, de uma só vez, o estado completo do ambiente. De certa forma, é o problema inverso ao problema da imprecisão: o sensor pode retornar a mesma medida para estados diferentes do ambiente. Este problema é vivenciado por humanos em labirintos, por exemplo, onde todas as paredes são iguais e a visão não dá pistas suficientes para revelar a localização;

- **imprecisão dos atuadores**: tome-se por exemplo um robô diferencial, como o da seção precedente. Conhecendo-se a velocidade de rotação de cada uma das duas rodas (odometria), é possível utilizar a eq. 2.4 para obter a velocidade do robô no referencial global, e em seguida integrá-la no tempo para obter o deslocamento do robô. Infelizmente, há várias fontes de erro que tornam o uso exclusivo da odometria inviável. Entre os erros sistemáticos, que podem ser corrigidos, pode-se citar o erro de calibração quanto ao diâmetro e alinhamento das rodas. Porém, há erros aleatórios que não podem ser ignorados, advindos de contato desigual com o solo, variação do ponto de contato da roda com o solo, resolução limitada durante a integração, deslizamento, etc. Em geral, o erro na variação angular do robô ultrapassa rapidamente o erro de distância percorrida. A Fig. 2.2 mostra o aumento da incerteza ao longo do tempo, usando apenas a odometria do robô.

O método usado pela robótica clássica para resolver os problemas acima é a localização por mapa. Assume-se que o robô possui uma representação geométrica interna do ambiente, que torna possível

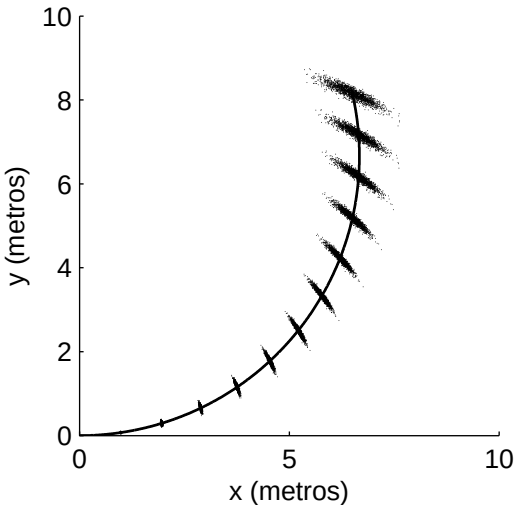


Fig. 2.2: Incerteza quanto à posição de um robô diferencial, usando apenas dados de odometria. O robô parte da origem e se desloca. Se os atuadores fossem perfeitos, o robô seguiria o trajeto da linha. Porém, flutuações na velocidade de cada roda levam a um aumento da incerteza sobre a posição ao longo do tempo.

simular observações e compará-las com as observações reais, e assim estimar em que posição do mapa o robô se encontra. Os métodos probabilísticos de localização permitem, além disso, obter a probabilidade de o robô se encontrar em cada posição. Tal probabilidade é chamada de crença do robô.

Tal estratégia de localização leva a uma gama de escolhas em dois quesitos: na representação do mapa e na representação da crença do robô. A representação da crença é classificada em:

- **hipótese única**: a cada instante, o robô atribui probabilidade máxima para um determinado ponto e probabilidades menores para pontos próximos a ele. A principal vantagem da hipótese única é a facilidade de representação, que geralmente é dada por um valor médio e sua disperção. A desvantagem é a impossibilidade de considerar posições distantes simultaneamente;

- **múltiplas hipóteses**: a cada instante, o robô possui um ou mais máximos locais de crença. Assim, é possível que o robô leve em conta a possibilidade de se encontrar em lugares espacialmente distantes um do outro. Por exemplo, se o robô é *sequestrado* e posicionado diante de uma porta numa sala com duas portas, o robô poderá atribuir crenças igualmente fortes para posições próximas a cada uma das duas portas. Isso não é possível no caso de hipótese única. A principal desvantagem das múltiplas hipóteses é a complexidade da representação.

Quanto à representação do mapa, há uma grande gama de escolhas:

- **representação contínua**: representa os obstáculos por polígonos no espaço 2D contínuo. Nesses mapas, a crença do robô é um domínio contínuo, ou seja, o robô possui uma probabilidade $p(\mathbf{x})$ de se encontrar em cada posição $\mathbf{x} \in \mathbb{R}^2$;
- **representação por decomposição**: nesse esquema, o espaço 2D é decomposto em um número finito de conjuntos, cada um representando uma região. Assim, a crença do robô limita-se à probabilidade de o robô se encontrar em cada uma das regiões. Há diferentes formas de decomposição. Entre elas, está a decomposição por grade uniforme e a representação topológica.

Uma vez escolhida a forma de representação da crença e do mapa, é possível aplicar algoritmos de estimação recursiva para obter, a cada momento, um valor atualizado para a crença do robô. Assim, para crenças de hipótese única em mapas contínuos, utiliza-se o filtro de Kalman. Para múltiplas hipóteses em mapas contínuos, há o filtro de partículas. Para mapas discretos, utiliza-se a localização por modelos de Markov. Todos esses métodos são implementações do filtro bayesiano.

2.1.4 A necessidade da localização

Enquanto que locomoção e percepção são, sem dúvida, elementos fundamentais para a navegação em robótica móvel, há uma discussão quanto à necessidade de localização. A comunidade de robótica comportamental (Arkin, 1998), por exemplo, defende que, para muitas tarefas de navegação, a simples interação imediata com o ambiente é capaz de levar o robô ao seu objetivo, sem necessidade de construção de mapas de identificação da região do espaço onde o robô se encontra.

A localização da forma apresentada acima permite que a crença do robô seja expressa de modo humanamente interpretável. Porém, dependendo da tarefa que o robô tem a executar, pode ser um custo desnecessário para o robô construir tal representação.

Um dos objetivos do presente trabalho é encontrar algoritmos que sejam capazes de criar uma representação intermediária entre as apresentadas acima e a simples ausência de representação proposta pelos roboticistas comportamentais. O resultado desejado é a criação automática de mapas altamente abstratos, cuja decomposição seja adequada para a tarefa que o robô tem a executar, e não necessariamente interpretável por humanos.

2.2 O problema da navegação

Segundo Siegwart & Nourbakhsh (2004, seção 6.1), resolver o problema de navegação em robótica móvel consiste em dotar o robô de cognição, que "geralmente representa a tomada de decisão e execução que um sistema utiliza para realizar seus objetivos de mais alta ordem". O problema da navegação é assim definido: "dado conhecimento parcial de seu ambiente e uma posição final [*goal*

position ou uma série de posições, a navegação consiste na habilidade do robô de agir baseado em seu conhecimento e leituras sensoriais visando chegar a seu destino de forma tão eficiente e confiável quanto possível".

Note que este objetivo pode ser alcançado mesmo por robôs que não sejam dotados de estratégias explícitas de localização. Como mostrado na seção precedente, é possível que robôs naveguem sem conhecimento exato de sua posição física. Ao longo dessa seção, serão apresentadas diferentes estratégias de navegação, desde aquelas que dependem fortemente da localização até aquelas completamente reativas.

Nesse sentido, é possível separar as estratégias de navegação robótica em duas classes principais: de um lado, estão as estratégias deliberativas, que fazem uso de planejamento de longo prazo e utilizam modelos do ambiente, como mapas. Tais estratégias são especialmente apropriadas à resolução de problemas complexos, como o planejamento de trajetória e outras tarefas de alto nível do robô. De outro lado, as estratégias reativas, que utilizam principalmente as leituras imediatas dos sensores e prescindem de um modelo específico do ambiente. Estas, por sua vez, adequam-se a tarefas mais críticas e de curto prazo, como desvio de obstáculos.

As estratégias deliberativas e reativas, apresentadas acima, não devem ser vistas como opostos irreconciliáveis. Na verdade, as duas são altamente complementares: em robótica, é muito difícil obter sucesso em uma delas sem a outra. Em geral, as duas estratégias são implementadas de forma independente, e organizadas em uma arquitetura, como será visto na seção seguinte. Neste trabalho, porém, procura-se conciliar as duas em um só mecanismo.

2.3 Arquiteturas de navegação

Na robótica tradicional, cada técnica utilizada é responsável por um aspecto particular da navegação. Dessa forma, existem técnicas independentes para tratamento da informação sensorial, localização, desvio de obstáculos e planejamento de trajetória. É necessário então definir uma arquitetura que integre todas as partes de forma sistemática.

Roboticistas tradicionais argumentam que a modularidade é importante para que a manutenção de um sistema de controle robótico possa efetuada de forma independente em cada um dos componentes. Apesar dessa visão não ser compartilhada por toda a comunidade de robótica, não é comum falar de um sistema completo de navegação sem decompô-lo, mesmo que de forma mais implícita.

Siegwart & Nourbakhsh (2004) identifica duas maneiras de proceder à decomposição de uma arquitetura: a *decomposição temporal* separa a arquitetura verticalmente em camadas que lidam com demandas em diferentes escalas de tempo, enquanto que a *decomposição funcional* separa a arquitetura horizontalmente em sub-tarefas executadas, desde a entrada sensorial até o atuador.

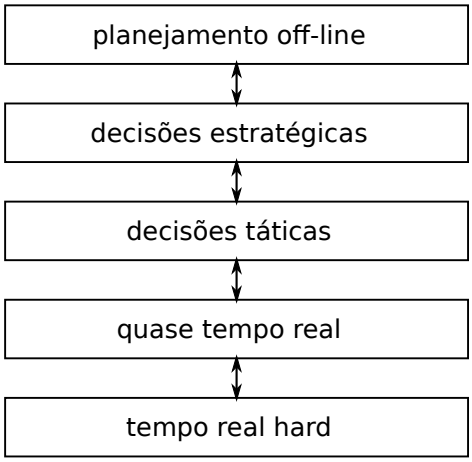


Fig. 2.3: Decomposição temporal genérica de uma arquitetura de navegação. Traduzida de Siegwart & Nourbakhsh (2004, seção 6.3.3.1).

2.3.1 Decomposição temporal

A decomposição temporal distingue processos com demandas de tempo real de processos menos temporalmente críticos. Tal separação leva a uma estrutura vertical em camadas, como mostrado na Fig. 2.3. Os componentes da arquitetura mais abaixo da pilha representam componentes cuja resposta temporal é mais crítica. Assim, por exemplo, o tempo real *hard* corresponde a componentes que devem gerar resposta numa frequência fixa de dezenas de *Hertz*, como é o caso do controlador PID de velocidade das rodas. De forma complementar, no topo da pilha estão os processos que não demandam tempo de resposta, como planejamento *off-line*.

Pode-se dizer que há quatro fatores que determinam em qual nível da escala temporal um componente se situa:

- **tempo de resposta ao sensor**: é definido como o tempo que um determinado módulo leva para que sua saída seja influenciada por um evento sensorial. Em níveis mais baixos da hierarquia temporal, o limitante é a própria velocidade do sensor e do processador. Em níveis mais elevados, o tempo de resposta pode ser retardado deliberadamente através de planejamento;

- **profundidade temporal**: corresponde à janela temporal, cobrindo desde o passado até o futuro, que é utilizada no funcionamento de um determinado módulo. De um lado, a janela de tempo passado de leituras sensoriais e outras informações utilizadas pelo módulo consiste na *memória temporal* do módulo. De outro lado, o *horizonte temporal* consiste na quantidade de tempo à frente que será considerado pelo módulo para determinar sua saída. Módulos mais deliberativos, localizados no topo da hierarquia temporal, possuem horizonte e memória temporal grandes, enquanto que o nível mais baixo deve se preocupar apenas com a leitura imediata do

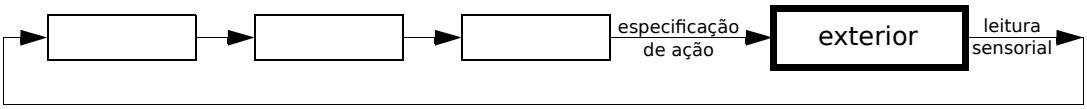


Fig. 2.4: Arquitetura de controle puramente serial. Adaptada de Siegwart & Nourbakhsh (2004, seção 6.3.3.2).

sensor ou com o comando imediato a impor ao atuador;

- **proximidade espacial**: já que o objetivo da navegação é fazer o robô se deslocar no espaço ao longo do tempo, o espaço coberto por um módulo está relacionado diretamente com a sua profundidade temporal. Assim, módulos com baixa profundidade temporal tomam decisões de velocidade e orientação, altamente localizadas. Níveis com profundidade temporal maior devem lidar com planejamento cobrindo regiões espaciais mais extensas, como por exemplo determinar a posição do robô num futuro mais distante;

- **especificidade do contexto**: em geral, módulos tomam decisões não apenas como função da entrada imediata do sensor, mas utilizando também outras variáveis que informam o estado do robô. Módulos na base da hierarquia temporal, reativos, são independentes de contexto. Já módulos deliberativos dependem fortemente do contexto em que está inserido o robô para tomarem decisões.

2.3.2 Decomposição funcional

A decomposição funcional (*control decomposition*) diz respeito à decomposição do controlador ao longo do caminho que a informação percorre desde a entrada no controlador (após ser emitida pelos sensores) até a saída do controlador (e consequente entrada nos atuadores). Há dois extremos que classificam arquiteturas quanto a tal decomposição: de um lado, as arquiteturas *puramente seriais*, nas quais o sinal percorre um caminho linear e perfeitamente sequencial, passando por um módulo de cada vez, desde os sensores até a saída. No outro extremo, encontram-se as arquiteturas *puramente paralelas*. Nesse caso, a leitura dos sensores é prontamente distribuída a todos os módulos do sistema. A saída de todos os módulos é combinada por um módulo de combinação e entregue aos atuadores.

As Figs. 2.4 e 2.5 exemplificam, respectivamente, uma arquitetura puramente serial e uma arquitetura puramente paralela. Nessas figuras, cada quadro é um módulo e as linhas representam a comunicação de informações entre módulos. O módulo **externo** representa o ambiente e a parte física do robô, ou seja, seus atuadores e sensores. Os módulos restantes, portanto, correspondem ao controlador.

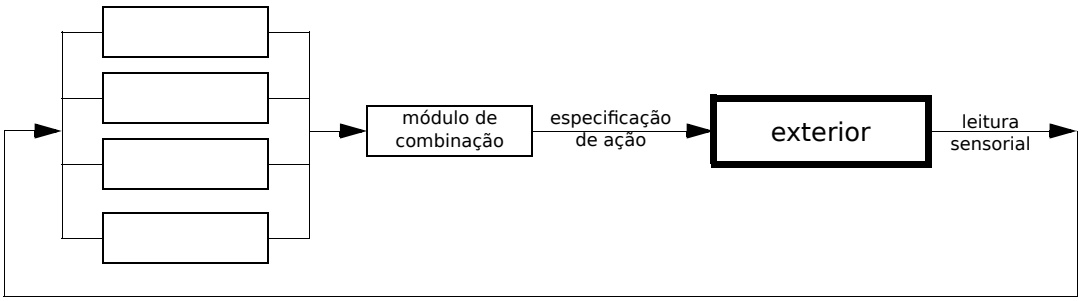


Fig. 2.5: Arquitetura de controle puramente paralela. Adaptada de Siegwart & Nourbakhsh (2004, seção 6.3.3.2).

Arquiteturas seriais

A principal vantagem de uma arquitetura serial como a da Fig. 2.4 é o fato de ser previsível e verificável. Como a informação segue um único caminho da entrada até a saída, é possível simular cada componente separadamente. Além disso, é fácil prever como a saída de um único componente afetará o componente seguinte.

Arquiteturas paralelas

Como pode ser notado a partir da Fig. 2.5, numa arquitetura puramente paralela cada um dos módulos de controle recebe diretamente uma parte ou mesmo toda a leitura sensorial. Cada módulo é então capaz de processar tais leituras, utilizando ou não um estado interno, e produzir comportamento, ou seja, é capaz de comandar diretamente os atuadores do robô.

Cada um desses módulos pode ser responsável por um comportamento distinto. Por exemplo, um deles pode ser responsável pelo desvio de obstáculos, enquanto que o outro é responsável pelo planejamento de trajetória. Fica claro, então, que uma arquitetura paralela necessita de um componente adicional para coordenar a atividade de cada um dos módulos. Há duas maneiras de se fazer tal coordenação. A primeira delas consiste no chaveamento temporal, em que apenas um módulo tem sua saída executada a cada instante de tempo. A segunda é chamada de controle misto, em que as saídas de mais de um módulo são combinadas a cada instante de tempo para compor o sinal de saída do controlador.

O chaveamento temporal é também chamado de sistema competitivo, pois cada módulo compete para ter sua saída executada. O controle misto, por sua vez, é um sistema colaborativo, pois cada módulo colabora com uma parcela do sinal de controle a cada instante de tempo.

A grande vantagem de um sistema baseado em **chaveamento temporal** é que, se a frequência de chaveamento for suficientemente baixa, é possível prever o comportamento do sistema como um todo a partir do comportamento de cada um de seus módulos. Assim, por exemplo, um robô com

os módulos de planejamento de trajetória e de desvio de obstáculos pode comutar do primeiro para o segundo quando o robô se encontrar na proximidade de um obstáculo. Será possível, durante a navegação do robô, identificar e prever seu comportamento.

Porém, à medida que a frequência de comutação se torna alta em relação ao horizonte de controle de cada um dos módulos, o comportamento geral do robô pode se tornar imprevisível e pobre. Por exemplo, se o módulo de desvio de obstáculos for ativado e desativado antes de concluir sua tarefa, o robô pode não ter sucesso em contornar uma parede. Se o chaveamento for rápido demais para todos os módulos, o robô ficará oscilando entre um comportamento e outro e apresentará um comportamento errante. Portanto, não se deve pensar que bons módulos individuais levem necessariamente a um bom controlador. É necessário que o chaveamento entre comportamentos seja elaborado com cuidado.

Uma limitação de sistemas de chaveamento temporal é que um comportamento não é capaz de influenciar o outro. Por exemplo, num caso em que o desvio de obstáculos pode ser efetuado tanto à esquerda quanto à direita, o módulo de planejamento de trajetória poderia influenciar na escolha. Porém, na arquietura puramente chaveada, não há essa possibilidade.

Esta é, por sua vez, uma vantagem dos sistemas de **controle misto**. Nesses sistemas, as saídas de dois ou mais módulos são combinadas matematicamente para produzir a ação a ser executada pelos atuadores. Dessa forma, um módulo pode complementar o outro simultaneamente. Os modelos de controle misto são mais genéricos que os modelos de chaveamento temporal, mas apresentam desafios muito maiores. Enquanto que no chaveamento temporal é possível controlar a velocidade de comutação para garantir os comportamentos individuais, no modelo misto todos os comportamentos estão parcialmente ativos a cada instante.

Por exemplo, suponha que os módulos de trajetória e de desvio de obstáculos fornecessem em suas saídas vetores de direção e que a saída final fosse a soma vetorial ponderada dos dois módulos. Se um módulo aponta para curva à esquerda e o outro à direita, o resultado poderia ser "seguir em frente", que tem fortes chances de ser um comportamento de baixa qualidade. O módulo de combinação se torna um fator determinante no comportamento do robô, e deve ser especificado cuidadosamente.

Arquiteturas paralelas são populares em robótica comportamental. Como exemplo de sistemas chaveados (competitivos), cita-se a rede de comportamentos de Maes (1990), na qual o comportamento escolhido é aquele com o maior grau de ativação. Os comportamentos estimulam e inibem uns aos outros, através das conexões da rede. Outro exemplo é a arquitetura de subsunção de Brooks (1986). Nesta arquitetura, cada componente, além de ser ligado aos sensores e aos atuadores, recebe e envia sinais de supressão. Assim, os componentes, cada um deles capaz de gerar comportamento, são também capazes de suprimir a ação de outros componentes.

Como exemplo de sistemas de controle misto (colaborativo), cita-se Arkin (1998). Sua arqui-

tetura, chamada de *motor-schema*, é composta de comportamentos, ou módulos, cujas saídas são linearmente combinadas e então ligadas aos atuadores.

Finalmente, pode-se dizer que as arquiteturas de controle paralelas apresentam a importante desvantagem de serem difíceis de avaliar, porque num mesmo instante de tempo há vários processos sendo executados em velocidades diferentes, o que se soma às restrições de tempo dos sensores, dos atuadores e da própria tarefa a executar. Por outro lado, uma característica interessante de sistemas paralelos é a maior proximidade a sistemas biológicos. Animais apresentam comportamentos complexos e altamente paralelos. O desenvolvimento de arquiteturas paralelas pode ajudar na compreensão de sistemas biológicos e usar tal conhecimento para trazer melhoria aos sistemas artificiais.

2.4 Resumo

Ao longo do capítulo, foram apresentados alguns tópicos relacionados à robótica móvel, com ênfase no problema de navegação em robôs móveis autônomos.

- Tratou-se da locomoção de um robô móvel, e foi apresentado um modelo simples de cinemática para o caso de robô diferencial com duas rodas e dois atuadores independentes.

- Foram apresentados alguns tipos de sensores utilizados em robótica móvel. Foi explicado, em específico, o caso de sensores de distância, que serão utilizados neste trabalho.

- Discorreu-se sobre o problema de localização, que consiste em, a partir da leitura dos sensores e de um mapa do ambiente, estimar a posição e orientação do robô. A solução para tal problema, no contexto de navegação autônoma, é considerada imprescindível pelos roboticistas clássicos, e desnecessária para alguns roboticistas comportamentais.

- Foram apresentadas as duas abordagens antagônicas para a solução do problema de navegação em robótica móvel: as estratégias completamente reativas, propostas por alguns roboticistas comportamentais, e as estratégias deliberativas, consideradas necessárias por roboticistas clássicos.

- As estratégias deliberativas utilizam memória temporal e realizam localização. O resultado é um controlador cuja estrutura interna é mais compreensível aos humanos. Porém, roboticistas comportamentais argumentam que tal complexidade é desnecessária e mesmo prejudicial ao controle autônomo. A maioria dos projetos de sistemas de navegação utiliza sistemas híbridos, que integram subsistemas reativos e deliberativos.

- Finalmente, foram apresentadas as decomposições temporal e funcional de um controlador de navegação em robótica autônoma. A decomposição temporal diz respeito a como os diferentes subsistemas de um controlador se organizam quanto ao tempo de resposta, à quantidade de informações passadas registradas na memória e à abrangência temporal e espacial de planejamento. A decomposição funcional, por sua vez, diz respeito a como a informação flui ao longo do caminho entre sensores e atuadores.

- Quanto à decomposição funcional, controladores podem ser classificados em seriais ou paralelos. Em arquiteturas seriais, a informação sensorial flui ao longo de uma cadeia de subsistemas, até chegar aos atuadores. Em arquiteturas paralelas, todos os subsistemas recebem paralelamente a leitura sensorial, e competem ou colaboram para alimentar os atuadores.

Capítulo 3

Sistemas baseados em regras

O capítulo anterior apresentou o problema de navegação em robótica móvel de um ponto de vista tradicional, da forma como é geralmente apresentado por roboticistas. Neste capítulo, toma-se um caminho conceitualmente diferente. São apresentadas meta-heurísticas desenvolvidas pela comunidade de inteligência artificial, mais especificamente pela comunidade de algoritmos evolutivos, para solucionar o problema de navegação.

A ideia de sistemas baseados em regras surgiu ainda nas primeiras décadas do desenvolvimento da inteligência artificial e adquiriu novas roupagens ao longo do tempo, reaparecendo com diferentes nomes. Este capítulo apresenta inicialmente a abordagem de Holland (1975), os sistemas classificadores.

O capítulo introduz também o sistema de regras desenvolvido por Cazangi (2008), que estende a ideia de sistemas classificadores. Criam-se conexões entre regras, o que resulta numa rede classificadora. Em capítulos posteriores, esta abordagem será tomada como inspiração na especificação de um modelo probabilístico de controlador autônomo para navegação em robótica móvel.

3.1 Sistemas classificadores

Em seu livro, Holland (1975) elabora uma teoria geral para sistemas adaptativos. Segundo tal teoria, um sistema adaptativo possui uma **estrutura interna**, que é modificada ao longo do tempo por meio de um conjunto de **operadores**. O **ambiente** no qual o sistema se encontra fornece uma **resposta** para cada estrutura interna apresentada pelo sistema.

Em geral, como parte da resposta do ambiente encontra-se um valor numérico chamado *fitness*, que mede a qualidade da estrutura testada pelo sistema, em termos de seu grau de adaptação ao meio. Assim, o objetivo de um sistema adaptativo é encontrar uma sequência de operadores que modifique a sua estrutura interna de modo a maximizar o *fitness* retornado pelo ambiente.

Rótulo	SE	ENTÃO	Energia
A	`1#1##`	`11`	8,5
B	`1110#`	`01`	15,2
C	`11111`	`11`	5,9
D	`##0##`	`10`	19,0

Tab. 3.1: Exemplos de classificadores.

Esta teoria é geral o suficiente para modelar processos adaptativos de diferentes áreas do conhecimento: evolução das espécies na biologia, aprendizagem na psicologia, planejamento ótimo na economia, controle de mecanismos e inferência estatística. Abordando especificamente o problema da aprendizagem, Holland (1986) formula os sistemas classificadores, cujo objetivo é responder à questão: como um sistema melhora sua performance em um ambiente em que avaliações explícitas de performance são apenas raramente disponíveis?

Um sistema classificador é um sistema adaptativo que interage com o ambiente por meio de detectores e executores. Os detectores capturam uma característica específica do ambiente e a codificam na forma de uma mensagem de entrada, que é entregue ao sistema classificador. O sistema, por sua vez, entrega uma mensagem de saída aos executores, que atuam sobre o ambiente. Além disso o ambiente fornece ao sistema, de tempos em tempos, uma recompensa, que representa uma medida de sua performance.

A base computacional de um sistema classificador é fornecida por um conjunto de regras *condição / ação*, chamadas de *classificadores*. Uma regra é formada por duas partes, o antecedente e o consequente, e expressa como "SE antecedente ENTÃO consequente". Um classificador é uma regra do seguinte tipo:

SE <`mensagem de entrada`> for fornecida pelos detectores
ENTÃO envie <`mensagem de saída`> aos executores,

onde <`mensagem de entrada`> e <`mensagem de saída`> são específicas a cada um dos classificadores do sistema. No sistema classificador padrão definido por Holland, as mensagens consistem em sequências de bits de tamanho fixo k. Assim, uma mensagem pertence ao conjunto $\{0, 1\}^k$. Além disso, as mensagens de entrada podem conter caracteres-curinga $\#$, que casam com ambos os bits 0 ou 1. A *especificidade* de uma regra é definida como o recíproco do número de condições que casam com o seu antecedente, que por sua vez é função do número de caracteres-curinga encontrados no antecedente. A tabela 3.1 apresenta exemplos de classificadores.

Vários classificadores podem estar ativos ao mesmo tempo. Para determinar qual deles terá sua mensagem de saída executada, é efetuada uma competição entre regras. Para tanto, a cada classificador, ou regra, é atribuída uma *energia*, que representa o quão útil a regra foi para o sistema no

passado. A cada regra é atribuído um valor de aposta, que depende diretamente de sua energia e de sua especificidade. A regra que oferecer a maior aposta tem sua mensagem de saída executada.

Além do processo acima, que realiza o processamento das mensagens, há dois outros processos executados pelo sistema de regras, relacionados ao aprendizado: a *atribuição de crédito* e a *descoberta de novas regras*.

O processo de *atribuição de crédito* consiste na definição de *quais regras* foram responsáveis pela recompensa retornada pelo ambiente em um determinado instante, e pela atualização da energia de tais regras. Na versão original do sistema classificador, a regra que tiver a aposta vencedora no instante do recebimento da recompensa tem a recompensa somada à sua energia.

Em versões mais sofisticadas, é utilizado o sistema de *bucket brigade*, em que, além de emitirem mensagens aos atuadores, as regras emitem também mensagens a outras regras, facilitando sua ativação. Nesse sistema, o antecedente de cada regra é formado por duas mensagens, uma delas comparada com a mensagem produzida pelos detectores, e a outra comparada com a mensagem de saída emitida pela regra executada anteriormente. O formato da regra é, então:

SE `<antecedente 1>` for fornecido pelos detectores
E `<antecedente 2>` for igual à saída da última regra
ENTÃO envie `<mensagem de saída>` aos executores.

A mensagem que ganha a aposta é executada, e o valor da aposta é transferido de sua energia para a energia da regra executada anteriormente. Assim, é como se a regra atual pagasse à regra anterior para ser executada. Finalmente, a recompensa recebida do ambiente é somada à última regra ativa. O processo inteiro funciona como uma cadeia, que transfere gradualmente a recompensa do ambiente às regras mais distantes. Regras pertencentes a sequências que levam a recompensas positivas acabam sendo fortalecidas.

O processo de *descoberta de regras* é executado ao fim de uma sequência de iterações dos processos de atribuição de crédito. Tal processo consiste num algoritmo genético em que cada regra do sistema corresponde a um indivíduo da população. O *fitness* de cada indivíduo é dado pela energia da regra correspondente, calculada pelo processo de atribuição de crédito. Uma regra dá origem a seu cromossomo através da concatenação do seu antecedente com seu consequente, por exemplo.

O algoritmo genético primeiramente seleciona os indivíduos com maior *fitness* através de um método probabilístico, como o da roleta. Em seguida, os indivíduos selecionados sofrem mutações e recombinações, e substituem a população anterior.

3.2 Sistemas classificadores de Cazangi

Cazangi (2004) apresenta um sistema de navegação autônomo (SNA) baseado nos sistemas classi-

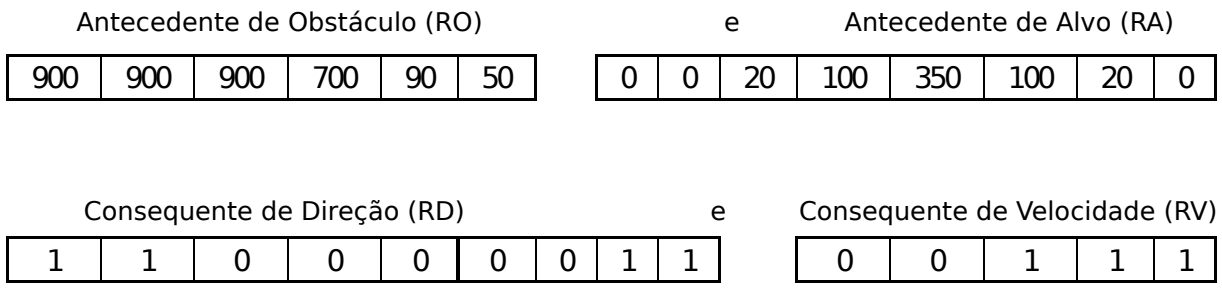


Fig. 3.1: Exemplo de regra cujos consequentes determinam 3 graus de giro à esquerda e aumento de velocidade. Extraído de Cazangi (2008, seção 3.3.1).

ficadores com aprendizagem, introduzidos na seção anterior. Esse sistema é formado por duas partes: sistema de controle e processo de evolução ou aprendizagem. O sistema de controle corresponde à parte de processamento das mensagens num sistema classificador, enquanto que o processo de evolução engloba simultaneamente a atribuição de crédito e a descoberta de regras.

Diferentemente dos sistemas classificadores de Holland, o ciclo de evolução ou aprendizagem não ocorre a cada intervalo de tempo fixo, mas sim quando ocorrem determinados eventos: colisão, captura ou monotonia. Isso significa que, cada vez que o sistema recebe uma recompensa, seja ela positiva ou negativa, em vez de ocorrer reatribuição de crédito às regras, o algoritmo genético é executado.

Cada regra no SNA de Cazangi (2004) é formada por um antecedente e um consequente. Tanto o antecedente quanto o consequente são, cada um deles, compostos de dois vetores. O antecedente é formado pelos vetores **antecedente de obstáculo** ($RO \in \mathbb{R}^6$) e **antecedente de alvo** ($RA \in \mathbb{R}^8$). Os consequentes, por sua vez, são formados pelo **consequente de direção** ($RD \in \{0,1\}^9$) e pelo **consequente de velocidade** ($RV \in \{0,1\}^4$). Os vetores antecedentes (RA e RO) são formados por elementos pertencentes aos números reais, enquanto que os consequentes são formados por elementos binários. A Fig. 3.1 mostra um exemplo de regra.

Cada elemento nos vetores de antecedentes da regra corresponde ao valor de leitura de um dos sensores do robô *Khepera* II, no qual o sistema foi aplicado. Assim, cada um dos 8 valores do antecedente de obstáculo corresponde a um sensor de distância do robô, e cada um dos 6 consequentes de alvo corresponde a um sensor de luminosidade.

O consequente de direção RD é formado por 5 bits de sentido e 4 bits de valor absoluto da direção. Se a maioria dos 5 primeiros bits for 0, o robô gira à esquerda. Caso contrário, o robô gira à direita. Os 4 últimos bits permitem realizar uma curva de 0 a 15 graus. A redundância na codificação do sentido de rotação evita que mutações esporádicas modifiquem bruscamente o comportamento do robô. O consequente de velocidade, RD, por sua vez, leva a um aumento de velocidade se a maioria de seus 5 bits for 1, e diminuição da velocidade caso contrário. O incremento ou diminuição de velocidade é

sempre de um valor constante.

A competição de classificadores no SNA é, de certa forma, mais simples do que nos sistemas classificadores originais, pois não há noção de energia de um classificador. O único critério utilizado na competição é a semelhança $S(r)$ do antecedente da regra r com o estímulo instantâneo recebido do ambiente pelos sensores. Assim, a cada instante de tempo, o classificador r mais semelhante ao estímulo tem seu consequente executado, como especificado acima.

Porém, como os antecedentes não são binários, a definição de semelhança não é tão trivial quanto para o caso dos sistemas classificadores de Holland. Cazangi define a semelhança $S(r)$ da regra r com a leitura dos sensores como:

$$S(r) \equiv \frac{\|RO(r) - EO\|}{MaxO} + \frac{\|RA(r) - EA\|}{MaxA}, \tag{3.1}$$

onde EO e EA são as leituras instantâneas dos sensores de distância e de luminosidade, respectivamente, $RO(r)$ e $RA(r)$ são os antecedentes da regra r, $\|\cdot\|$ é a norma euclidiana e $MaxO$ e $MaxA$ são constantes de normalização. Note que, apesar da definição de Cazangi, $S(r)$ na verdade não é uma semelhança, mas sim uma *distância*. A regra executada a cada instante de tempo é então aquela com o menor $S(r)$.

Como no SNA não há o conceito de energia das regras, não existe o processo de atribuição de créditos. Assim, toda modificação no comportamento do sistema é gerada por processos evolutivos. Outra diferença é que, no SNA, não há o conceito de recompensa. Isso torna necessária a criação de processos evolutivos especializados para cada evento que ocorrer com o robô. Os eventos considerados por Cazangi são: colisão, captura de alvo e monotonia.

O evento de **colisão** ocorre toda vez que pelo menos um dos sensores de distância apresenta uma leitura menor que um limiar pré-definido. No processo evolutivo disparado por colisão, as regras com maior probabilidade de serem selecionadas para reprodução são aquelas cujos antecedentes apresentam maior semelhança com o estímulo dos sensores ($S(r)$ pequeno), cujos consequentes de direção apresentem tendência de girar no sentido oposto ao do obstáculo, e cujos consequentes de velocidade apresentem tendência de reduzir a velocidade do robô. Após a seleção das melhores regras, ocorre a reprodução. Antecedentes e consequentes sofrem, separadamente, cruzamento e mutação, e em seguida são recombinados aleatoriamente para formar novas regras.

O evento de **captura de alvo** também leva a um processo evolutivo. Nesse caso, as regras selecionadas são aquelas com maior similaridade ao estímulo (como no caso da colisão), e aquelas cujo ângulo de máxima luminosidade seja próximo ao ângulo onde o alvo luminoso foi capturado. Além disso, são priorizadas regras que levem a uma redução de velocidade. A reprodução das regras selecionadas ocorre da mesma forma que na colisão.

Finalmente, um evento de **monotonia** é disparado quando uma sequência de regras de compri-

mento fixo não produz uma tendência de redução da distância em relação ao alvo. Nesse caso, todas as regras causadoras desta estagnação ou piora são removidas e substituídas por novas regras aleatórias.

É importante notar que o processo evolutivo descrito acima é, de certa forma, supervisionado, porque guia as regras no sentido da solução considerada melhor. Por exemplo, no caso de colisão, o processo evolutivo está guiando o conjunto de regras rumo a uma solução que gire o robô no sentido oposto ao obstáculo e que reduza a velocidade na proximidade de um obstáculo. Num sistema em que o ambiente é completamente desconhecido, esse tipo de treinamento não seria possível. Assim, um sistema classificador completo deveria ser capaz de aprender a solução através de tentativa e erro, sem ser guiado de forma supervisionada a uma solução específica.

3.3 Redes classificadoras de Cazangi

Nesta abordagem, Cazangi (2008) implementa uma rede híbrida baseada em sistemas classificadores e em redes imunológicas, visando dotar o agente de comportamentos não-reativos.

A teoria de rede imunológica sugere que o sistema imune seja composto por uma rede de antígenos e anticorpos capazes de reagir uns aos outros, ativando-se ou inibindo-se mutuamente (Jerne, 1974). Assim, numa rede imunológica artificial, cada nó representa um anticorpo, e as conexões entre eles correspondem às ações ativadoras ou inibidoras entre diferentes anticorpos. Cada anticorpo i possui uma concentração instantânea $c_i(t)$, cuja variação é função, por um lado, da concentração de antígenos que têm afinidade com este anticorpo e, por outro, das ações inibidoras ou ativadoras dos anticorpos vizinhos.

As redes imunológicas artificiais possuem, então, ao contrário dos sistemas classificadores, um estado interno, que pode ser comparado a uma memória de curta duração. Esta memória pode gerar comportamentos não reativos, necessários em alguns problemas. Um sistema baseado exclusivamente em regras isoladas, sem interações e sem considerar uma concentração dinâmica na atuação das regras, poderia apenas dar origem a um sistema puramente reativo, sem memória e com dificuldade de contornar mínimos locais durante a navegação. O sistema de redes imunológicas vem, portanto, aumentar a capacidade cognitiva do sistema (de Castro & Timmis, 2002).

A dinâmica da rede imunológica que Cazangi usa é modelada por um sistema de equações diferenciais ordinárias acopladas, cujas variáveis são as concentrações instantâneas $c_i(t)$ de cada anticorpo:

$$\frac{d}{dt}c_i(t) = c_i(t)\left[\sum_{j=1}^{N}\left(\alpha m_{ij}^{+} - \beta m_{ij}^{-}\right)c_j(t) + \gamma M_i - h\right], \quad \forall i \in \{1,\ldots,N\} \tag{3.2}$$

onde α, β e γ e h são constantes, m_{ij}^{+} e m_{ij}^{-} representam, respectivamente, os efeitos de estimulação

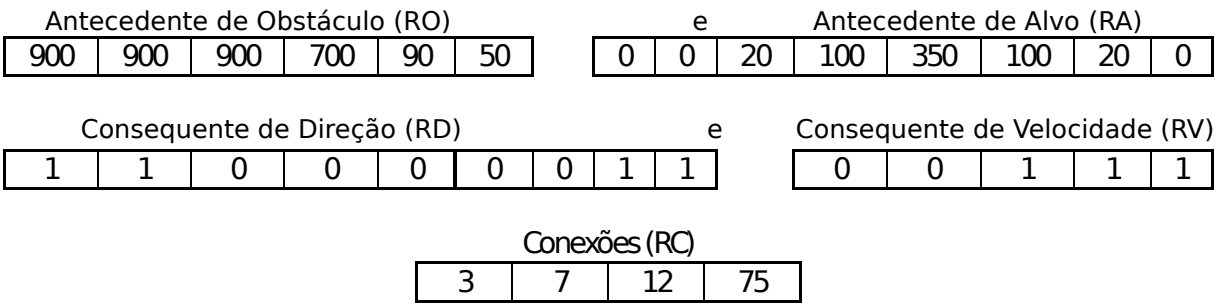


Fig. 3.2: Formato de uma regra da rede imunológica de Cazangi.
Formato de uma regra (anticorpo) da rede imunológica de Cazangi. Extraída de Cazangi (2008, seção 5.5.1).

e supressão do anticorpo j sobre o anticorpo i, e M_i é uma medida de similaridade entre o anticorpo i e o estímulo instantâneo do ambiente.

Na rede imunológica classificadora de Cazangi, cada anticorpo corresponde a um classificador. A medida de similaridade M_i entre anticorpo e antígeno é então obtida a partir da similaridade do classificador com a leitura sensorial, dada pela eq. 3.1. Porém, como aquela medida é na verdade uma medida de disparidade, deve-se fazer a seguinte transformação:

$$M_i \equiv \frac{1}{S(r_i)}, \tag{3.3}$$

onde r_i é o classificador correspondente ao i-ésimo anticorpo. Nota-se que agora cada regra r_i possui, além de um antecedente e um consequente, uma lista de conexões m_{ij} com as outras regras, como mostra a Fig. 3.2.

Finalmente, a ativação da regra para efeito de competição não é mais simplesmente a sua similaridade $S(r)$ com o estímulo. A competição entre regras, a cada instante de tempo, é feita em relação à concentração $c_i(t)$ de cada regra. A regra que possuir a maior concentração no instante t tem o seu consequente executado.

Quanto à evolução dessa rede híbrida, há dois aspectos a tratar: a evolução dos nós (classificadores), e a de suas conexões. Adota-se, para a primeira, um algoritmo imunológico, com clonagem, mutação e exclusão dinâmica de nós inativos. Para as conexões, por sua vez, adota-se um algoritmo genético com mutação e recombinação. Para efetuar a evolução, o robô é inserido em um ambiente virtual de navegação. Inicializam-se os nós e as conexões de forma aleatória. Cada vez que há uma colisão do robô com um obstáculo, quando o robô atinge um alvo ou quando o mesmo tem um comportamento monótono, há uma evolução dos nós, como explicado na seção precedente para o SNA. A evolução das conexões ocorre menos frequentemente, a cada período de tempo pré-determinado.

Como exemplo de rede imuno-classificadora em operação, toma-se o controlador de um robô

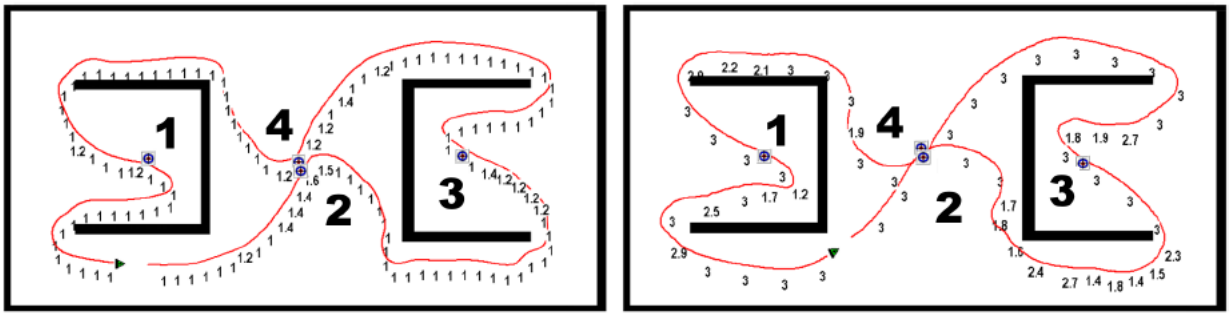


Fig. 3.3: Ambiente em duplo U na qual o robô deve desviar de obstáculos e chegar ao alvo. Extraída de Cazangi (2008, seção 5.6.2).

cujo objetivo é desviar de obstáculos e atingir alvos representados por fontes de luz. O robô detecta obstáculos através de seus sensores infra-vermelhos, e o alvo através de seus sensores de luz. O deslocamento é efetuado por duas rodas movidas por motores independentes.

Para mostrar que o controlador é capaz de tirar o robô de becos, possuindo portanto um comportamento não-reativo, o alvo é posicionado atrás de uma parede em forma de U, como mostrado na Fig. 3.3. O robô deve fugir do interior do U e buscar o alvo luminoso, representado na figura por círculos azuis. A cada instante de tempo, apenas um dos alvos está visível. A figura mostra a ordem de aparecimento dos alvos através de um número posicionado sobre cada um deles.

Os robôs evoluídos utilizando a rede imuno-classificadora mostraram-se capazes de escapar de mínimos locais associados às condições de navegação, exibindo comportamentos não-reativos. Mais especificamente, o robô foi capaz de sair do interior do U e perseguir o alvo que estava em seu exterior.

Um exemplo de rede imuno-classificadora resultante do processo evolutivo é ilustrado na Fig. 3.4. As regras com maior número de conexões são representadas pelos círculos de maior raio. Das 100 regras inicialmente consideradas, a rede resultante do processo evolutivo convergiu para 80 regras. Isso reflete uma das características dos sistemas imunológicos artificiais: a presença de meta-dinâmicas capazes de auto-regular o tamanho da população de anticorpos.

Apesar da natureza altamente distribuída das representações incorporadas na rede, é possível distinguir a função específica de vários de seus nós. A Fig. 3.5 mostra a mesma rede de uma forma diferente, com os nós replicados acima e abaixo para facilitar a visualização. Os 36 nós violeta, à esquerda, são associados ao desvio de obstáculos; os 40 azuis à direita associam-se à captura de alvo. Finalmente, 4 deles não têm tarefas específicas.

No mesmo experimento, foi efetuada uma tentativa de se retirar as conexões entre as regras e deixar o robô navegar. O robô ficou andando em círculos. Isso mostra que, ao menos nas redes que foram obtidas através de evolução, as conexões entre regras realizam papel crucial no controle do robô. Observando-se a dinâmica de tais redes, percebe-se que as conexões induzem uma sequência de

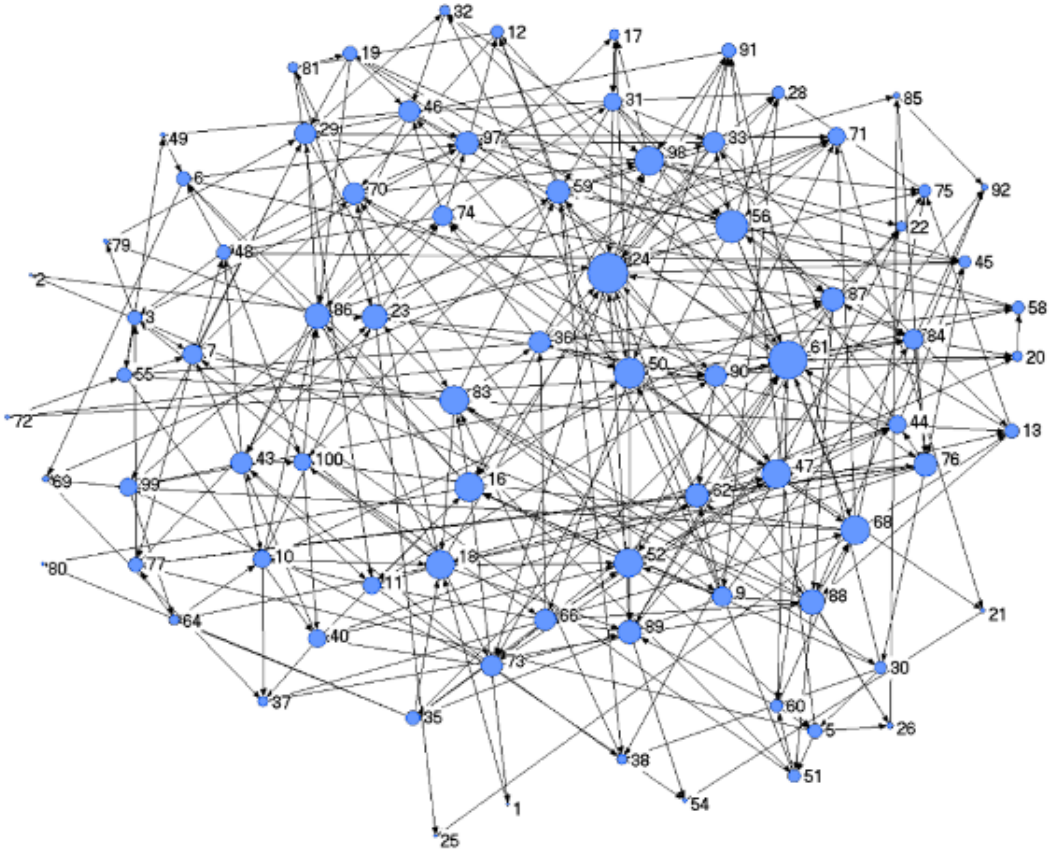


Fig. 3.4: Exemplo de rede imuno-classificadora evoluída. Extraída de Cazangi (2008, seção 5.6.3).

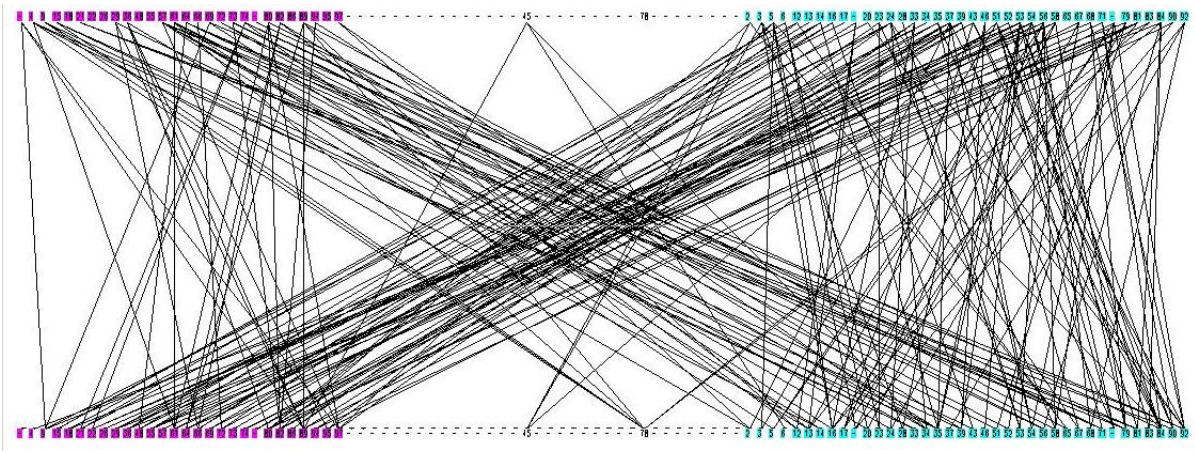


Fig. 3.5: Outra visualização da rede da Fig. 3.4. Os nós acima e abaixo são os mesmos, para facilitar a visualização. Extraída de Cazangi (2008, seção 5.6.3).

ativação de regras ao longo do tempo, o que pode explicar o comportamento não-reativo apresentado.

3.4 Resumo

Neste capítulo, foram apresentados alguns sistemas de regras adaptativos, e a aplicação dos mesmos como controladores de robôs móveis autônomos.

- Os sistemas classificadores de Holland (1986) consistem em sistemas de regras cujas regras são sintetizadas e evoluem de forma adaptativa. Através do uso da leitura sensorial e de uma medida de *fitness*, as regras são modificadas de forma que a performance do sistema melhore ao longo do tempo.

- Cazangi (2004) apresenta um sistema de navegação autônomo cuja inspiração se encontra nos sistemas classificadores de Holland (1986). Uma das principais diferenças em relação aos sistemas de Holland é a presença de números reais, e não apenas dígitos binários, no antecedente das regras. Para tanto, Cazangi introduz o conceito de distância entre o antecende de uma regra e a leitura sensorial.

- Cazangi (2008) desenvolve um controlador para navegação que consiste em uma rede imunológica artificial. O conjunto de nós da rede, ou seja, os anticorpos, formam um sistema de regras semelhante ao do sistema de navegação autônomo de Cazangi (2004). A rede imunológica introduz, além disso, o conceito de concentração de um anticorpo e interconexão entre anticorpos. Dessa forma, é possível obter do robô comportamentos não alcançáveis a agentes puramente reativos, como comprova o experimento efetuado em labirinto com forma de U.

Capítulo 4

Processos de decisão de Markov e aprendizagem por reforço.

No capítulo anterior, foram apresentados os sistemas classificadores de Holland (1975). Tais sistemas podem ser usados para resolver o problema de navegação de um robô através de um conjunto de regras geradas por um algoritmo genético. No presente capítulo, é apresentada uma nova roupagem para o mesmo problema: a aprendizagem por reforço, que possui um bom embasamento matemático e no qual os processos de decisão de Markov (MDPs) assumem papel central.

O conteúdo deste capítulo baseia-se principalmente em Sutton & Barto (1998), um texto voltado para a comunidade de inteligência artificial. Outras duas referências importantes são Bertsekas & Tsitsiklis (1996), voltado para controle, e Powell (2007), da área de pesquisa operacional. Nessas áreas, a aprendizagem por reforço recebe os nomes de *Neuro-Dynamic Programming* e *Approximate Dynamic Programming*, respectivamente.

A aprendizagem por reforço é uma das formas mais intuitivas de se pensar no problema de aprendizagem: os seres humanos, assim como os animais, aprendem através da interação com o ambiente. Mesmo não havendo um professor para nos guiar a respeito das escolhas a tomar, nosso organismo proporciona sinais de reforço dependendo da resposta que o ambiente produz. Por exemplo, se encostamos em uma chapa quente, um sinal negativo é produzido e aprendemos a nos afastar da chapa. Note que este esquema de separação entre agente e ambiente é o mesmo utilizado por Holland na sua definição de sistemas classificadores.

A visão computacional de aprendizagem por reforço se baseia fortemente nessa ideia intuitiva de aprendizagem. Diferentemente da maioria dos algoritmos da área de aprendizado de máquina, num problema de aprendizagem por reforço não há um vetor de erro, ou seja, não há instrução guiando o agente rumo à resposta correta. Um sinal de reforço negativo apenas indica que o resultado da ação tomada foi ruim, e não qual a correção a tomar.

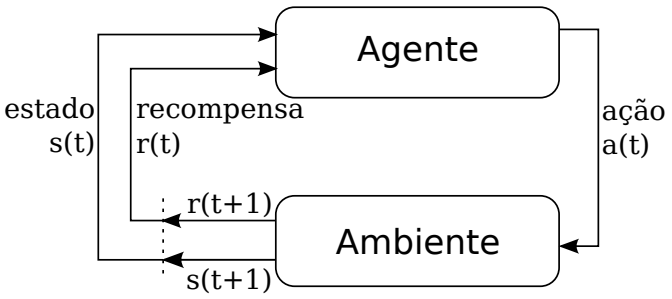


Fig. 4.1: Elementos do problema de aprendizagem por reforço.

4.1 Definição e terminologia

O problema de aprendizagem por reforço envolve dois elementos, representados na Fig. 4.1: o **ambiente** e o **agente**, que, na área de controle, se traduzem por *planta* e *controlador*. A cada instante de tempo, o ambiente se encontra em um estado, cuja evolução é condicionada pelas ações que o agente toma. O agente, por sua vez, é capaz de ler, a cada momento, o estado do ambiente, e responder com uma ação. Após cada ação, o agente recebe um sinal de recompensa do ambiente. O objetivo do agente é maximizar a recompensa acumulada ao longo do tempo.

A separação entre agente e ambiente é flexível, e depende da formulação do problema: o agente pode ser, por exemplo, nosso corpo, recebendo sinais físicos através dos sentidos, ou então apenas o cérebro, recebendo sinais eletroquímicos do resto do corpo através dos neurônios. Ou ainda, o agente pode ser um grupo de neurônios interagindo com o resto do cérebro. Por isso, o fato de a recompensa ser externa ao agente não significa que ela vem de fora do corpo humano, nesse caso.

Note também que uma determinada ação executada pelo agente pode levar a períodos intermediários de recompensa negativa, para só mais tarde trazer bons resultados. Nesse sentido, o agente deve ser capaz de realizar um planejamento visando não simplesmente a uma boa recompensa imediata, mas sim a uma boa recompensa acumulada a longo prazo. Em outras palavras, o agente não deve ser guloso em relação à recompensa imediata.

Os elementos esboçados até agora podem ser definidos de maneira mais formal: o ambiente é visto como um processo de decisão de Markov, e o agente consiste em uma política, que associa estados do ambiente com probabilidades de executar cada uma das ações pertencentes a um conjunto de ações possíveis.

4.1.1 O Ambiente: Processo de Decisão de Markov (MDP)

Um processo de decisão de Markov (MDP, do inglês *Markov Decision Process*) consiste em um sistema que evolui ao longo do tempo, em passos discretos, $t = \{0, 1, 2, \cdots\}$. A cada passo t, o

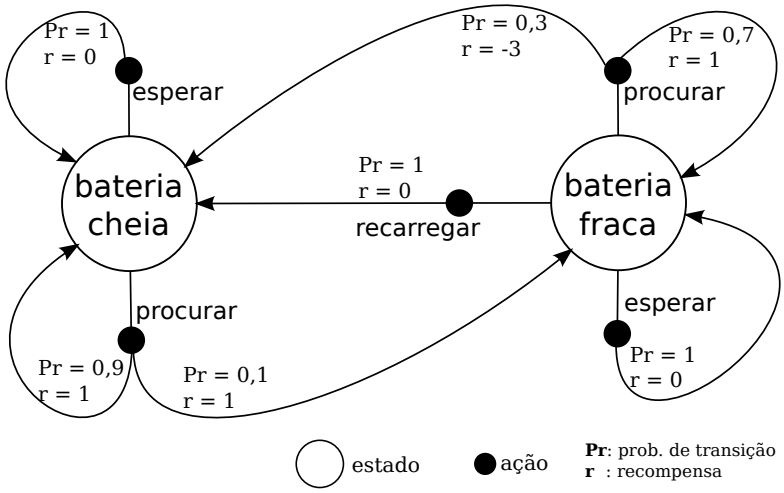


Fig. 4.2: Exemplo de Processo de Decisão de Markov: robô lixeiro.

sistema se encontra em um estado s_t. É possível interagir com o sistema, a cada passo, através de ações. O estado do sistema no tempo seguinte, s_{t+1}, depende estocasticamente apenas do estado atual s_t e da última ação a_t tomada. Além da transição de estados, o sistema fornece uma recompensa estocástica r_{t+1}, que depende dos estados s_t e s_{t+1} e da ação tomada a_t.

Formalmente, um MDP consiste numa tupla $(\mathcal{S}, \mathcal{A}_s, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$, com:

$$\begin{aligned} \mathcal{P}^a_{ss'} &\equiv \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}, \\ \mathcal{R}^a_{ss'} &\equiv \mathrm{E}\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}. \end{aligned}$$

Na equação, $\mathcal{S}$ é o conjunto de estados, $\mathcal{A}_s$ é o conjunto de ações disponíveis no estado s, $\mathcal{P}^a_{ss'}$ é a probabilidade de transição do estado s para o estado s' quando a ação a é executada, e $\mathcal{R}^a_{ss'}$ é a recompensa esperada quando se toma a ação a no estado s e se chega ao estado s'.

Os conjuntos $\mathcal{S}$ e $\mathcal{A}_s$ podem ser finitos, infinitos contáveis, ou subconjuntos de espaços métricos contínuos satisfazendo algumas restrições. Além disso, há formulações para tempo contínuo. Portanto, MDPs são gerais o suficiente para representar variados problemas de decisão.

Um MDP é um processo markoviano, ou seja, o estado s_{t+1} no tempo $t+1$ depende apenas do estado s_t e da ação a_t. Dessa forma, tem-se[1]:

$$\Pr\{s_{t+1} = s' \mid s_{1:t-1}, a_{1:t-1}, s_t = s, a_t = a\} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} = \mathcal{P}^a_{ss'}, \tag{4.1}$$

ou seja, o estado s_{t+1}, dados s_t e a_t, é probabilisticamente independente de todos os estados $s_1 \ldots s_{t-1}$ e ações $a_1 \ldots a_{t-1}$ anteriores.

[1]Ao longo desse capítulo, utiliza-se a notação $o_{1:t} \equiv o_1, o_2, \ldots, o_t$ por motivos de abreviação e clareza.

A Fig. 4.2 é um exemplo de MDP: trata-se de um robô lixeiro em seu ambiente. Este MDP possui dois estados: bateria cheia e bateria fraca. A transição de estados depende da ação tomada, assim como a recompensa. Por exemplo, se o robô está com *bateria fraca* e toma a ação *procurar*, com probabilidade $0,3$ sua bateria acaba e ele deve ser transportado para sua base por um agente externo. Por isso, a recompensa é fortemente negativa, -3. Mas, com probabilidade $0,7$, a bateria não acaba. O estado não muda, mas o robô recebe uma recompensa positiva 1 por ter procurado lixo.

4.1.2 O agente

Um agente inserido em um processo de decisão de Markov deve, para cada estado, escolher uma ação. Assim, ele deve possuir uma **política**, que pode ser estocástica ou determinística. No caso estocástico, a política consiste em uma função $\pi : \mathcal{S} \times \mathcal{A} \mapsto (0, 1)$, que atribui uma probabilidade a cada par estado-ação. Assim, $\pi(s, a)$ é a probabilidade de se escolher a ação a quando o ambiente se encontra no estado s, seguindo a política π.

O objetivo do agente é maximizar o retorno, ou seja, a recompensa acumulada ao longo do tempo. Há várias formas de se definir o retorno. Em episódios finitos, o retorno pode ser simplesmente a soma das recompensas de cada instante de tempo, desde o instante inicial. Para episódios não limitados, porém, o retorno deve ser definido de forma a evitar que se chegue a valores infinitos. Uma forma é descontar a importância de recompensas mais longínquas no futuro. Assim, define-se o retorno R_t no tempo t como:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{4.2}$$

onde o fator de desconto $0 < \gamma < 1$ faz variar a importância dada a recompensas futuras. Note que recompensas passadas não influenciam no retorno, pois o agente não pode mais ter influência sobre elas.

Agora, se um agente segue uma política fixa π, a partir de um estado s, a trajetória futura do sistema está estocasticamente definida, já que o processo é markoviano, ou seja, não depende dos estados anteriores. Assim, define-se o valor $V_\pi(s)$ do estado s como sendo o retorno esperado se o agente seguir a política fixa π:

$$V^\pi(s) = \mathrm{E}_\pi\{R_t \mid s_t = s\}. \tag{4.3}$$

A função $V : \mathcal{S} \mapsto \mathbb{R}$, definida acima, é chamada de função-valor. Analogamente, pode-se definir a função-valor da ação $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, da seguinte forma:

$$Q^\pi(s, a) = \mathrm{E}_\pi\{R_t \mid s_t = s, a_t = a\}, \tag{4.4}$$

ou seja, o retorno esperado a partir do estado s, tomando-se a ação a, e depois seguindo a política π. A função $Q^\pi(s,a)$ é especialmente útil quando não se conhece o modelo do sistema, como será mostrado adiante.

4.1.3 O Problema

Para conseguir acumular o maior retorno possível, o agente deve encontrar e seguir uma política ótima π^* cuja função-valor V^* seja tal que $V^{\pi^*}(s) \geq V^\pi(s), \quad \forall \pi, s$. Se o agente já conhece o modelo do ambiente, ou seja, conhece o processo de decisão de Markov, trata-se de um problema de planejamento, e a solução mais conhecida é através da **programação dinâmica**, técnica clássica da área de pesquisa operacional.

Porém, quando o ambiente não é conhecido, o agente deve, ao mesmo tempo, aprender o ambiente e planejar a melhor estratégia possível. Este problema consiste na aprendizagem por reforço, e a solução se dá por meio de uma adaptação do método de programação dinâmica.

4.2 A Solução

O elemento fundamental para a solução, presente em quase todos os algoritmos relacionados à programação dinâmica, é a equação de Bellman:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')], \tag{4.5}$$

onde γ é o fator de desconto. Toda função-valor satisfaz essa equação, dada uma política π. Uma política ótima, por sua vez, satisfaz a equação de *otimalidade* de Bellman:

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]. \tag{4.6}$$

O método da programação dinâmica compreende duas operações: *E*: avaliação da política e *I*: melhoria da política. Inicia-se com uma política qualquer, π_0, e intercalam-se as duas operações, como segue:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^*.$$

A avaliação da política (*E*) pode ser efetuada de forma direta, resolvendo-se a eq. 4.5, que, para conjuntos $\mathcal{S}$ e $\mathcal{A}$ finitos, consiste em um sistema de equações lineares. Em geral, no entanto, utiliza-se uma solução iterativa, em que a igualdade na eq. 4.5 é considerada como uma atribuição, e repete-se até que a função $V^\pi(s)$ se estabilize. A melhoria da política (*I*) é obtida simplesmente criando-se uma

política π_{i+1} que seja gulosa em relação a V^{π_i} da política anterior, ou seja: em cada estado, escolhe-se a ação que tenha maior probabilidade de levar a um estado de maior valor. Através da intercalação dessas duas operações, *E* e *I*, a convergência à política ótima π^* é garantida.

O método da programação dinâmica, descrito acima, só pode ser aplicado quando as transições de estado e as recompensas forem conhecidas. Quando o modelo do sistema é desconhecido, é necessário aprender o valor $Q^\pi(s, a)$ das ações, não apenas o valor $V^\pi(s)$ do estado, como será visto a seguir. Além disso, tal função deve ser estimada a partir da experiência, ou seja, do agente atuando no ambiente real ou simulado.

O método mais direto para se obter tais estimativas é Monte Carlo (Metropolis, 1987). A partir de uma política inicial π_0, faz-se o agente atuar durante algum tempo (um episódio) no ambiente. A seguir, de posse de toda a sequência de recompensas, é possível estimar o retorno originário de cada par estado-ação, $Q_{\pi_0}(s, a)$. Após um número satisfatório de episódios, obtém-se uma estimativa aproximada para a função-valor da política π_0.

Para obter uma melhoria da política, toma-se a política gulosa em relação à função-valor. É nesse momento em que a função-valor da *ação* se torna necessária: sem um modelo do ambiente disponível, não haveria como saber o efeito das ações sobre o ambiente. De posse dos valores de cada ação, basta escolher a ação a que maximize $Q^\pi(s, a)$ para um dado estado s.

Assim, o método de Monte Carlo também consiste em uma intercalação das operações de avaliação e melhoria da política. Há duas limitações importantes nesse método, porém: 1) é preciso pelo menos um episódio inteiro para que haja atualização da política e 2) a variância da estimativa do retorno é grande, pois consiste no acúmulo de recompensas estocásticas ao longo de todo um episódio. Essas limitações são contornadas através do método da diferença temporal (TD, do inglês *temporal difference*), que consiste em um compromisso entre programação dinâmica e o método de Monte Carlo.

O método da diferença temporal foi intuitivamente usado várias vezes ao longo da história da inteligência artificial, como no jogador de Damas de Samuel (Samuel, 1959) e no método *bucket brigade* (Holland, 1986), como forma de aprendizagem e predição. Em Sutton (1988), a ideia por trás dessas aplicações é clarificada: faz-se uma apresentação formal e dá-se uma prova de convergência.

No método da diferença temporal, o valor do par estado-ação é atualizado a cada intervalo de tempo, a partir da experiência adquirida no tempo anterior, sem que haja necessidade de um episódio inteiro. A diferença temporal δ_{t+1} é definida como sendo a diferença entre o retorno esperado anteriormente $V(s_t)$ e o retorno esperado dada a recompensa recebida. Pela definição de retorno

apresentada na eq. 4.2, tem-se:

$$\begin{aligned} \mathrm{E}\{R_t \mid r_t\} &= r_t + \mathrm{E}\left\{\sum_{k=1}^{\infty} \gamma^k r_{t+k}\right\} = r_t + \gamma \mathrm{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}\right\} \\ &= r_t + \gamma \mathrm{E}\{R_{t+1}\} \\ &= r_t + \gamma V(s_{t+1}). \end{aligned}$$

Assim, a diferença temporal δ_t no tempo t é:

$$\begin{aligned} \delta_t &\equiv \mathrm{E}\{R_t \mid r_t\} - \mathrm{E}\{R_t\} \\ &= r_t + \gamma V(s_{t+1}) - V(s_t). \end{aligned} \tag{4.7}$$

Essa diferença temporal pode então ser utilizada para atualizar a estimativa do valor do estado anterior:

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_t, \tag{4.8}$$

onde α é a taxa de aprendizagem. Nota-se que a introdução do erro temporal δ_t transforma o problema de avaliação da política em um problema de aprendizagem supervisionada. É provado que esse método faz a função-valor convergir com probabilidade 1.

É interessante citar que, apesar da origem e das aplicações relacionadas à aprendizagem de máquina, recentemente o método de aprendizado por diferenças temporais foi utilizado para modelar de forma satisfatória o mecanismo de disparo de neurônios dopaminérgicos no cérebro (Ludvig et al., 2008).

A aplicação do método da diferença temporal na função-valor da ação permite que o ambiente seja desconhecido, e leva a um dos métodos mais difundidos da aprendizagem por reforço, o Q-learning.

4.2.1 Q-learning

O método *Q-learning*, cuja convergência foi provada por Watkins & Dayan (1992), é um método livre de modelo, ou seja, não é necessário o conhecimento das transições de estado, nem das recompensas esperadas em cada estado. Neste método, a diferença temporal é calculada a cada passo a partir dos valores de ação:

$$\delta_t = r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t), \tag{4.9}$$

e o valor da ação é, então, atualizado estocasticamente com uma taxa de aprendizagem α:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t. \tag{4.10}$$

A política pode ser obtida diretamente a partir da função Q. Basta, a cada estado s, tomar a ação a que maximize $Q(s, a)$. Porém, para que a aprendizagem ocorra de maneira satisfatória, não se pode seguir sempre uma política gulosa, sob o risco de não se visitar estados desconhecidos mas com potencial. Surge, então, o compromisso entre explotação e exploração: é necessário seguir uma política quase-gulosa (explotação) para obter boas recompensas, mas é também necessário explorar para conhecer $Q(s, a)$ para todos os possíveis estados e ações. Há várias técnicas para lidar com o compromisso explotação-exploração. Uma das mais simples é a chamada política ϵ-greedy, definida como segue:

$$\pi(s_i, a_i) = \begin{cases} 1 - \epsilon & \text{se} \quad a_i = \arg\max_a Q(s_i, a) \\ \frac{\epsilon}{|\mathcal{A}|-1} & \text{c.c.} \end{cases} \tag{4.11}$$

onde $|\mathcal{A}|$ é o número de ações disponíveis. Dessa forma, a política é gulosa com probabilidade $1 - \epsilon$, e exploratória com probabilidade ϵ. Essa constante ϵ pode começar alta e ser reduzida ao longo do tempo, quando a função $Q(s, a)$ já estiver bem conhecida.

4.2.2 Traço de elegibilidade, TD(λ) e Q(λ)

Nos métodos de diferença temporal descritos acima, a cada intervalo de tempo, o único valor atualizado é o do estado imediatamente anterior, $V(s_t)$, ou, no caso do *Q-learning*, do par estado-ação imediatamente anterior, $Q(s_t, a_t)$, conforme as eqs. 4.8 e 4.10, respectivamente. Porém, mantendo-se um histórico dos últimos estados e ações $(s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \cdots)$, é possível também atualizar o valor de tais estados, já que a recompensa recebida r_{t+1} também os influencia. Define-se, então, o traço de elegibilidade, uma função $\tau_t : \mathcal{S} \mapsto \mathbb{R}$ que, para cada estado, leva a um número. Similarmente, para o *Q-learning*, define-se $\tau_t : \mathcal{S} \times \mathcal{A} \mapsto R$. A cada instante de tempo, a função τ é atualizada da seguinte maneira:

$$\tau_{t+1}(s) = \begin{cases} \lambda\gamma\tau_t(s) + 1 & \text{se} \quad s = s_t \\ \lambda\gamma\tau_t(s) & \text{c.c.} \end{cases} \tag{4.12}$$

ou, no caso do *Q-learning*,

$$\tau_{t+1}(s, a) = \begin{cases} \lambda\gamma\tau_t(s, a) + 1 & \text{se} \quad s = s_t, a = a_t \\ \lambda\gamma\tau_t(s, a) & \text{c.c.} \end{cases} \tag{4.13}$$

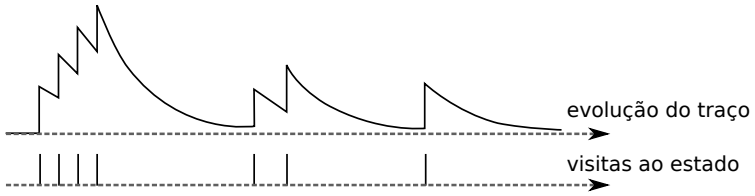


Fig. 4.3: Evolução do traço de elegibilidade ao longo do tempo.
Evolução do traço de elegibilidade ao longo do tempo, para um dado estado.

Assim, o estado recém-visitado tem seu traço $\tau(s_t)$ incrementado, enquanto os outros têm seu traço reduzido de um fator $\lambda\gamma$, onde γ é o desconto definido anteriormente, e $0 \leq \lambda \leq 1$ é o fator de esquecimento, cujo significado ficará mais claro adiante. O mesmo ocorre no caso do *Q-learning*, salvo que apenas é incrementado o traço do par estado-ação recém-visitado, $\tau(s_t, a_t)$. A Fig. 4.3 mostra o aspecto geral da evolução do traço ao longo do tempo, para um dado estado.

Os métodos TD(λ) e Q(λ) assemelham-se muito aos métodos TD e Q, apresentados nas seções anteriores. A diferença temporal δ_t é calculada da mesma forma, pelas eqs. 4.7 e 4.9, respectivamente. Porém, em vez de apenas o valor do estado anterior ser atualizado, todos os estados têm seu valor atualizado, modulado pelo traço. Para o TD(λ), tem-se:

$$V(s) \leftarrow V(s) + \alpha\tau(s)\delta_t, \quad \forall s \in \mathcal{S} \tag{4.14}$$

Similarmente, para o Q(λ),

$$Q(s,a) \leftarrow Q(s,a) + \alpha\tau(s,a)\delta_t, \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{4.15}$$

Dessa forma, estados que não foram recentemente visitados, ou ações que não foram executadas recentemente, por possuírem traço mais fraco, serão apenas levemente atualizadas, enquanto pares estado-ação recentes serão mais fortemente atualizados. No caso limite em que $\lambda = 0$, apenas o par estado-ação (s_t, a_t) será atualizado, exatamente como nos métodos já apresentados. Portanto, aqueles método são genericamente chamados de TD(0) e Q(0). Em contrapartida, quando $\lambda = 1$, todos os estados percorridos desde o início do episódio serão atualizados, como faz o método de Monte Carlo. Por isso, Monte Carlo pode ser genericamente chamado de TD(1) ou Q(1). Experimentalmente, verifica-se que valores de λ intermediários levam a resultados melhores, por promoverem um melhor compromisso entre viés e variância.

4.2.3 Funções aproximadoras

Os métodos descritos até agora funcionam bem quando o número de estados e de ações é limitado. Porém, em qualquer problema real, é muito fácil que o número de estados cresça exponencialmente. Por exemplo, em um robô com N sensores, em que cada sensor pode assumir M valores, o número de estados possíveis lidos pelo robô é $|\mathcal{S}| = M^N$. Assim, com apenas 10 sensores e 50 valores possíveis de leitura, são 50^{10} estados, um número proibitivo. Além disso, o número de ações disponíveis para um robô também é muito elevado: a cada atuador pode-se aplicar vários níveis de força. Se o valor de cada par estado-ação $Q(s, a)$ for armazenado em uma tabela, seria necessário um espaço proibitivo, e uma quantidade de experiência gigantesca para se aprender cada valor. Este problema foi originalmente descrito por Bellman (1957) como a *maldição da dimensionalidade*, nome pelo qual é conhecido até hoje.

Devido à impossibilidade da utilização de tabelas para problemas reais, deve-se recorrer a uma forma mais compacta de representação que, além de possuir um número menor de variáveis a serem aprendidas, seja capaz de generalizar o aprendizado para casos similares. Introduz-se então o conceito de aproximação paramétrica da função-valor. A função-valor passa a ser $V(s, \theta)$ ou, no caso do *Q-learning*, $Q(s, a, \theta)$, onde $\theta \in \mathbb{R}^n$ é um conjunto de n parâmetros reais que definem completamente a função para qualquer valor de s e a. Uma escolha criteriosa de $V(\cdot, \theta)$ ou $Q(\cdot, \cdot, \theta)$ pode ser capaz de generalizar de forma satisfatória, reduzindo bastante a necessidade de experiência no aprendizado. Além disso, há uma grande flexibilidade na escolha da complexidade do problema, já que o número de parâmetros pode ser escolhido dependendo da precisão que se deseja atingir.

O método de diferenças temporais é facilmente adaptável à parametrização descrita acima. Pode-se utilizar, por exemplo, o método do gradiente descendente, em que os parâmetros da função são atualizados estocasticamente segundo δ_t. Dessa forma, em vez de atualizar diretamente a função-valor pela eq. 4.8, atualizam-se os parâmetros da função-valor, da seguinte forma:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\delta_t \nabla_\theta V(s_t, \theta_t). \tag{4.16}$$

A parametrização também é possível para o caso em que $\lambda > 0$. Nesse caso, o traço de elegibilidade é definido como um vetor $\tau_t \in \mathbb{R}^n$ no espaço de parâmetros, e atualizado da seguinte forma:

$$\tau_{t+1} = \lambda\gamma\tau_t + \nabla_\theta V(s_t, \theta_t), \tag{4.17}$$

e os parâmetros são atualizados como segue:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\delta_t\tau_{t+1}. \tag{4.18}$$

Existe na literatura uma grande quantidade de formas da função aproximadora. Alguns exemplos empregando modelos não-lineares: redes neurais artificiais (em que os parâmetros correspondem aos pesos das sinapses), funções interpoladoras (Baird & Klopf, 1993), árvores de decisão (Ernst et al., 2006). Há também inúmeros exemplos de parametrização linear, como rede RBF com centros fixos, agregação de estados (Singh et al., 1995), vetor de características etc. Em geral, pode-se utilizar qualquer método de aprendizagem supervisionada *online* (ou seja, com ajuste imediatamente após a apresentação de cada estímulo) para atualizar os parâmetros, bastando alimentá-los com o sinal de erro temporal, δ_t.

Uma das aplicações mais conhecidas de aprendizagem por reforço, o jogador de Gamão de Tesauro (1992), utiliza o método TD(λ) com aproximação de função-valor por rede neural. Este jogador de Gamão aprendeu a jogar consigo mesmo, sem intervenção humana, e atingiu o nível dos melhores jogadores humanos.

Infelizmente, não se pode garantir convergência dos algoritmos de aprendizagem por reforço para qualquer tipo de função aproximadora. A convergência foi provada, porém, para o caso linear (Bertsekas, 2007), mas mantendo a política fixa, ou seja, durante a operação de avaliação da política. Quanto à operação de melhoria da política, da forma como foi apresentada anteriormente (ϵ-greedy), nem sempre há convergência.

Existe, porém, uma classe de métodos conhecidos como métodos de gradiente de política (Sutton et al., 2000; Peters & Schaal, 2008a), em que, assim como a função Q, também a política é parametrizada. Dessa forma, é possível generalizar para um número arbitrário de ações, e pode-se usar métodos de gradiente estocástico da política para fazê-la tender garantidamente a um máximo local.

4.2.4 Métodos de gradiente de política

Segundo Sutton et al. (2000), métodos baseados na maximização da função valor, como o Q-learning apresentado anteriormente, foram predominantes na década de 90, devido à maior velocidade de aprendizado. Porém, com a necessidade de utilização de funções parametrizadoras em problemas contínuos ou mais complexos, surgiu o problema da não convergência, discutido acima. Isso fez com que, na última década, a atenção se voltasse ao estudo de algoritmos de gradiente de política, que apresentam a interessante propriedade de convergência a um máximo local de retorno.

Métodos de gradiente de política assumem que a política do agente, ou seja, a probabilidade de executar uma ação a dado um estado s, seja representada por uma função π da ação a e do estado s, parametrizada por um conjunto de parâmetros θ, da seguinte forma:

$$\Pr\{a \mid s\} = \pi_\theta(s, a). \tag{4.19}$$

A função π pode ser, por exemplo, no caso de um número finito de ações, um classificador logístico cujas saídas sejam a probabilidade de executar cada uma das ações. No caso de ações contínuas, pode-se tomar uma rede neural com um ruído gaussiano na saída. Note que os parâmetros de exploração, como por exemplo o desvio padrão do ruído gaussiano, também podem fazer parte do conjunto de parâmetros θ da função π, permitindo assim que o algoritmo de aprendizado controle o compromisso entre exploração e explotação.

Como visto anteriormente, o retorno R obtido pelo agente após uma sequência de ações depende da política. Assim, uma variação $\Delta\theta$ nos parâmetros da política deve causar uma variação correspondente ΔR no retorno. A aproximação de primeira ordem dessa variação é expressa por

$$\Delta R \approx \nabla_\theta R \cdot \Delta\theta. \tag{4.20}$$

O vetor $\nabla_\theta R$ é o gradiente do retorno em relação aos parâmetros da política.

Sabe-se que o vetor gradiente aponta na direção de maior crescimento da função. Variando-se os parâmetros θ de uma função na direção do gradiente, com um passo α pequeno o suficiente, garante-se crescimento. Assim, fazendo-se, iterativamente,

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta R, \tag{4.21}$$

garante-se convergência para um máximo local de retorno (Williams, 1992).

Porém, visto que, pela definição do problema de aprendizado por reforço, o modelo do sistema não é conhecido, não há uma forma explícita de calcular $R(\theta)$ ou o gradiente $\nabla_\theta R(\theta)$. É necessario, portanto, estimar o gradiente a partir de recompensas recebidas por experiência.

O primeiro estimador não-viesado conhecido para o gradiente do retorno foi proposto por Williams (1992), na sua classe de algoritmos REINFORCE. Porém, o método proposto era aplicável apenas para os casos de retorno imediato ($\gamma = 0$) e retorno episódico ($\gamma = 1$ em um episódio finito). A generalização viria 8 anos depois.

Sutton et al. (2000) enuncia o teorema do gradiente de política, segundo o qual, para qualquer MDP, vale:

$$\nabla_\theta R = \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) \, [Q^\pi(s,a) - b(s)] \, da \, ds. \tag{4.22}$$

Na equação acima, $d^\pi(s)$ representa uma distribuição ponderada sobre estados, partindo-se de um estado específico s_0 e seguindo-se a política π:

$$d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t \Pr\{s_t = s \mid s_o, \pi\}. \tag{4.23}$$

O termo $\nabla_\theta \pi_\theta(a|s)$, por sua vez, é o *gradiente da política* em relação aos seus parâmetros θ (daí o nome dos algoritmos resultantes). A forma analítica de tal termo é geralmente conhecida, devido à escolha de parametrização da política (rede neural, classificador logístico, etc). O termo $Q^\pi(s, a)$ é a função-valor de ação da política π.

Finalmente, o termo $b(s)$ representa uma *baseline* que pode variar com o estado s mas não com a ação a. Tal *baseline* não altera o valor da integral, pois como $\int_\mathcal{A} \pi_\theta(a|s)\ da = 1$ (já que π é uma distribuição de probabilidades), então $\int_\mathcal{A} \nabla_\theta \pi_\theta(a|s)\ da = 0$. Por isso, qualquer termo independente de a ($b(s)$, por exemplo) no interior da integral sobre as ações não altera o resultado.

A eq. 4.22 pode ser reescrita para representar uma esperança calculada sobre a distribuição conjunta de estados e ações. Tem-se, então:

$$\begin{aligned}
\nabla_\theta R &= \int_\mathcal{S} d^\pi(s) \int_\mathcal{A} \pi_\theta(a|s)\ \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}\ [Q^\pi(s, a) - b(s)]\ da\ ds \\
&= \mathrm{E}_{s\sim d^\pi, a\sim\pi}\Big\{ \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}\ [Q^\pi(s, a) - b(s)] \Big\} \\
&= \mathrm{E}_{s\sim d^\pi, a\sim\pi}\Big\{ \nabla_\theta \ln \pi_\theta(a|s)\ [Q^\pi(s, a) - b(s)] \Big\}. \qquad (4.24)
\end{aligned}$$

No primeiro passo do desenvolvimento acima, a política $\pi_\theta(a|s)$ é multiplicada e dividida, sem alterar o resultado. Este artifício é chamado de razão de verossimilhança (Glynn, 1990), que encontra uso aqui, bem como em outros problemas de otimização estocástica[2], por permitir a estimação do gradiente de uma quantia estocástica através da média temporal.

De fato, é possível obter uma estimativa do gradiente de política expresso como na eq. 4.24 tomando-se a média sobre uma sequência de trajetórias efetuadas pelo agente seguindo uma política fixa π:

$$\nabla_\theta R = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} \sum_{t=1}^{\infty} \gamma^t \nabla_\theta \ln \pi_\theta(a|s)\ [Q^\pi(s, a) - b(s)]. \qquad (4.25)$$

Note que a distribuição $d^\pi(s)$, desconhecida, não aparece. Porém, há ainda a função-valor $Q^\pi(s, a)$ que também deve ser estimada. O método REINFORCE de Williams (1992) é obtido substituindo-se tal função valor pelo retorno R_t obtido ao fim do episódio (para $\gamma = 1$) ou pelo retorno imediato r_t (para $\gamma = 0$). De forma independente, Baxter et al. (2001) chegaram a uma estimação *online* do gradiente $\nabla_\theta R$ compatível com a eq. 4.25, usando também uma estimação *online* de $Q^\pi(s, a)$ a partir das recompensas imediatas r_t recebidas ao longo do tempo.

Se na fórmula exata da eq. 4.22 a *baseline* $b(s)$ não fazia diferença, na estimação baseada na média temporal da eq. 4.25, tal *baseline* torna-se extremamente importante, para evitar o crescimento

[2] A razão da verossimilhança é alternadamente expressa na literatura por $\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$ e $\nabla_\theta \ln \pi_\theta(a|s)$. As duas formas são equivalentes, pois, em geral, $\ln'(x) = \frac{x'}{x}$. Adota-se aqui a segunda forma, por ser mais compacta.

exacerbado da variância do estimador, à medida que a janela de tempo utilizada aumenta. Focando no objetivo de minimização da variância do estimador, Williams (1992) propõe um *baseline* ótimo, que também pode ser estimado ao longo do tempo.

Mesmo com um *baseline* ótimo, a estimação da função-valor de ação $Q^\pi(s,a)$ diretamente a partir do retorno R_t obtido é altamente suscetível a ruído, ou seja, apresenta uma grande variância. Alternativamente é possível parametrizar $Q^\pi(s,a)$ através de funções aproximadoras, como mostrado na seção 4.2.3. A respeito da parametrização da função-valor de ação, Sutton et al. (2000) enuncia o seguinte terema. Define-se, primeiramente, a função $f_w(s,a)$, linear no vetor de parâmetros w, como segue:

$$f_w(s,a) \equiv w^T \nabla_\theta \ln \pi_\theta(a|s), \tag{4.26}$$

O teorema afirma então que substituir $Q^\pi(s,a)$ por $f_w(s,a)$ na eq. 4.24 não altera o resultado. Pondo de outra forma, a parametrização de $Q^\pi(s,a)$ por $f_w(s,a)$ é exata para fins de cálculo do gradiente $\nabla_\theta R$. Por isso, $f_w(s,a)$ é chamada de "parametrização compatível".

Substituindo-se a parametrização $f_w(s,a)$ na eq. 4.24, e escolhendo-se arbitrariamente o *baseline* $b(s) = 0$, obtém-se:

$$\nabla_\theta R = \mathbf{G}_\theta w, \tag{4.27}$$

onde $\mathbf{G}_\theta$ é uma matriz quadrada definida como:

$$\mathbf{G}_\theta = \mathrm{E}_{s\sim d^\pi, a\sim\pi}\Big\{ \nabla_\theta \ln \pi_\theta(a|s)\ \nabla_\theta \ln \pi_\theta(a|s)^{\mathrm{T}} \Big\}. \tag{4.28}$$

Gradiente natural. Um dos problemas de se utilizar o gradiente $\nabla_\theta R$ diretamente na atualização dos parâmetros, como na eq. 4.21, é que a norma do gradiente, ou seja, a quandidade $\sqrt{\nabla_\theta R \cdot \nabla_\theta R}$, que determina o quanto os parâmetros θ vão variar, depende fortemente da parametrização da política. Esse fato pode levar a uma aprendizagem lenta dependendo do problema (Kakade, 2001). O ideal seria, então, se chegar a uma subida covariante no gradiente, ou seja, uma modificação de parâmetros independente da parametrização.

Da disciplina de geometria da informação (Amari & Nagaoka, 2007), tem-se que a matriz de informação de Fisher $\mathbf{F}_\theta$ oferece uma métrica do ganho esperado de informação (ou perda de entropia) dada uma alteração dos parâmetros de uma distribuição. Tal matriz é definida como uma esperança sobre o espaço $\mathcal{T}$ de possíveis trajetórias no espaço de estados:

$$\mathbf{F}_\theta = \mathrm{E}_{\tau|\theta}\Big\{ \nabla_\theta \ln p(\tau|\theta)\ \nabla_\theta \ln p(\tau|\theta)^{\mathrm{T}} \Big\}, \tag{4.29}$$

onde $p(\tau|\theta)$ é a probabilidade de uma trajetória τ no espaço de estados, dados os parâmetros θ da política.

O *gradiente natural*, definido como

$$\tilde{\nabla}_\theta R = \mathbf{F}^{-1} \nabla_\theta R, \tag{4.30}$$

é então invariante em relação à parametrização da métrica (Peters & Schaal, 2008b). Fazendo-se a atualização dos parâmetros θ com o gradiente natural, da seguinte forma:

$$\theta_{k+1} = \theta_k + \alpha_k \tilde{\nabla}_\theta R, \tag{4.31}$$

obtém-se um aprendizado mais rápido e mais robusto, por não ser suscetível à escolha dos parâmetros da função. O gradiente natural também é usado com sucesso para aprendizado supervisionado (Amari, 1998).

Há ainda um resultado que simplifica em muito a implementação de algoritmos de gradiente natural para aprendizado por reforço. Mostra-se (Bagnell & Schneider, 2003; Peters et al., 2003) que a matriz $\mathbf{G}_\theta$, definida na eq. 4.28, é equivalente à matriz de informação de Fisher $\mathbf{F}_\theta$. Assim, ao calcular-se o gradiente natural do retorno, dada uma parametrização compatível da função valor, tem-se, a partir das eqs. 4.30 e 4.27:

$$\tilde{\nabla}_\theta R = \mathbf{F}_\theta^{-1} \mathbf{G}_\theta w = w. \tag{4.32}$$

Assim, tornam-se desnecessárias as estimações da matriz $\mathbf{G}_\theta$ e $\mathbf{F}_\theta$. Portanto, utilizar o gradiente natural, além de uma melhoria em relação à velocidade de aprendizado, também torna o algoritmo muito mais simples.

Falta ainda apresentar uma forma de se estimar os parâmetros w da função-valor da ação. Primeiramente, mostra-se (Sutton et al., 2000) que a parametrização $f_w(s, a)$, apesar de ser uma aproximação da função-valor da ação $Q^\pi(s, a)$, na verdade tem capacidade de representar apenas a função-vantagem (*advantage function*) $A(a, s)$, definida como:

$$A(a, s) = Q(a, s) - V(s), \tag{4.33}$$

ou seja, representa a vantagem de se tomar uma ação em relação a outra, dado o estado. Dessa forma, torna-se impossível utilizar os métodos de diferença temporal diretamente em $f_w(s, a)$. Porém, é possível parametrizar, além da função-vantagem, a função valor. Dessa forma, tem-se:

$$\tilde{Q}_{v,w}(a, s) = \tilde{V}_v(s) + f_w(a, s), \tag{4.34}$$

ou seja, parametriza-se $V(s)$ por uma função $\tilde{V}_v(s)$ com parâmetros v. Se tal parametrização for linear, é possível usar o fato que $Q(s_t, a_t) = \mathrm{E}\{R_t \mid a_t, s_t\}$ e aplicar uma regressão linear nos retornos R_t obtidos ao longo do tempo para se estimar os parâmetros w e v. Conhecendo-se w, atualiza-se a política através das eqs. 4.32 e 4.31 é utilizado para o gradiente da política. O papel dos parâmetros v é de reduzir de forma significativa a variância da regressão linear. Um algoritmo de gradiente natural de política seguindo tais considerações será deduzido e apresentado em detalhes na seção 6.4.

4.3 Resumo

O capítulo apresentou o problema de aprendizagem por reforço e alguns métodos de solução.

- O problema de aprendizagem por reforço é equivalente ao problema apresentado no contexto de sistemas classificadores, no capítulo anterior: um agente atua sobre um ambiente com base em suas leituras sensoriais. Além disso, um sinal de reforço é recebido pelo agente a cada instante de tempo. O objetivo do agente é maximizar as recompensas recebidas a longo prazo.

- No modelo de aprendizagem por reforço proposto por Sutton & Barto (1998), o ambiente é representado por um processo de decisão de Markov (MDP), cujos parâmetros são inicialmente desconhecidos. O agente, por sua vez, possui uma política, que consiste em uma função ligando cada estado do ambiente a uma ação.

- O conceito de função-valor de estado e função-valor de ação é fundamental para a resolução do problema de aprendizagem por reforço. Tais funções incorporam a noção de recompensa a longo prazo. A solução para o problema consiste em encontrar uma política que maximize tais funções.

- O método de diferença temporal permite resolver o problema sem que seja necessário aprender diretamente os parâmetros do MDP.

- Para superar a maldição da dimensionalidade, que consiste na explosão do número de estados em problemas reais, apresenta-se o conceito de parametrização da função-valor. Dessa forma, é possível encontrar soluções aproximadas para problemas complexos.

- O método de gradiente de política é uma técnica alternativa ao método de diferença temporal, e utiliza o conceito de política parametrizada. Dessa forma, é possível encontrar soluções para um número infinito de ações. Além disso, garante-se atingir um máximo de recompensa a longo prazo.

Capítulo 5

Processos parcialmente observáveis (POMDPs)

Ao se utilizar um MDP para representar o ambiente em que o robô navega, da forma como foi apresentado no capítulo anterior, uma das hipóteses consideradas é que o agente tem acesso completo ao estado s_t do ambiente em cada instante de tempo. Porém, essa hipótese, associada à hipótese de Markov, ou seja, à hipótese de que o estado s_{t+1} depende apenas do estado s_t e da ação a_t e de nenhum estado ou ação anteriores, tornam MDPs pouco realistas no contexto de navegação em robótica móvel.

O presente capítulo se inicia expondo as limitações dos MDPs. Em seguida, os processos de decisão de Markov parcialmente observáveis (POMDPs, do inglês *Partially Observable Markov Decision Processes*) são destacados dentre as variadas soluções encontradas na literatura para tais limitações. Após uma apresentação conceitual dos POMDPs, os mesmos são comparados às redes imunológicas de Cazangi, que foram apresentadas no Capítulo 3.

5.1 Limitação do MDP: *Perceptual aliasing*

Ao aplicar o formalismo de MDP à robótica móvel, surge uma questão: como o agente acessa o estado s_t do ambiente? Uma solução simples é associar a leitura sensorial do agente o_t ao estado do ambiente, ou seja, faz-se $s_t \equiv o_t$. Isto, porém, pode levar a sistemas não-markovianos. Um exemplo simples desse fenômeno é mostrado na Fig. 5.1. No exemplo, o robô, dotado de sensores de distância, recebe as mesmas leituras o_t dos sensores em cada uma das quatro quinas. Porém, ao tomar a mesma ação a_t (girar 90 graus à direita, por exemplo), o robô é levado a uma observação o_{t+1} diferente para cada uma das quatro quinas. Esta situação de ambiguidade poderia ocorrer em várias outras posições do robô. Isso mostra que ou o ambiente é intrinsicamente não-markoviano, ou então

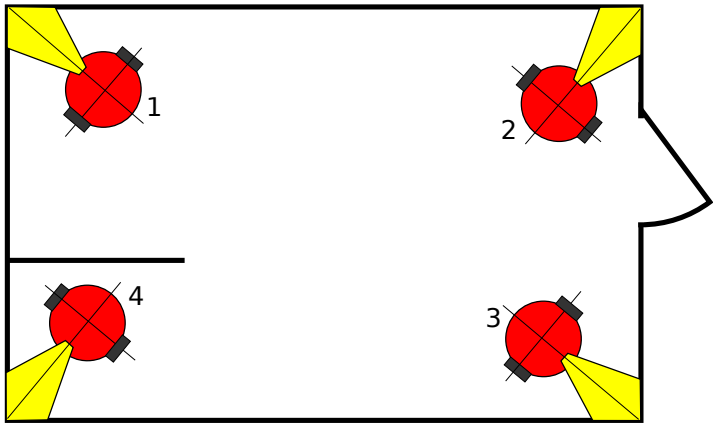


Fig. 5.1: Ilustração do efeito *perceptual-aliasing*, em que diferentes poses levam a uma mesma observação.

há um erro em considerar $s_t \equiv o_t$. No caso da Fig. 5.1, nota-se que o problema está em se igualar estado a observação atual. Este é, aliás, um exemplo do problema de *perceptual aliasing* (Chrisman, 1992), já citado na seção 2.1.3: diferentes poses do robô podem levar a leituras idênticas de sensor.

Há basicamente duas classes de abordagens para contornar este problema (Littman et al., 2002), ambas desvinculando a noção de estado da noção de observação. São elas: as abordagens baseadas em histórico, das quais são exemplos os modelos de Markov de ordem k, e as abordagens generativas, cujos mais tradicionais representantes são os POMDPs.

5.1.1 Abordagens baseadas em histórico: Modelos de Markov de ordem k

Uma forma de contornar o problema do *perceptual aliasing* é considerar não apenas a observação atual, mas também as $k-1$ observações anteriores e as $k-1$ últimas ações tomadas, como parte integrante do estado. Tem-se então,

$$s_t = (o_t,\ a_{t-1},\ o_{t-1},\ a_{t-2},\ o_{t-2},\ \ldots,\ a_{t-k},\ o_{t-k}). \tag{5.1}$$

Dessa forma, a chance de ocorrer *perceptual aliasing* é muito menor. No exemplo da Fig. 5.1, o robô que tivesse atingido uma das quinas teria seu estado determinado não apenas pela observação atual, mas também pelas observações e ações recentes, ou seja, as observações e ações tomadas durante o caminho que o fizeram chegar até a quina. Essa abordagem gera modelos conhecidos como Markov de ordem k, já que o próximo estado depende das k observações e ações anteriores.

A principal limitação dos modelos de Markov de ordem k é a dificuldade na determinação da janela k. Na verdade, em alguns problemas, não há k grande o suficiente para que o efeito de *aliasing* desapareça completamente. Por outro lado, um k grande demais cria estados cuja representação torna-

se impraticável. Além disso, em geral nem todo estado necessita de todas as k últimas observações. Pode ocorrer que, num mesmo ambiente, alguns estados sejam determinados a partir de uma única leitura de sensor, enquanto que outros necessitam de várias leituras, não necessariamente adjacentes no tempo. Mais ainda, por vezes nem toda a informação contida em uma observação é necessária para se determinar o estado. Por exemplo, em um robô cuja observação é obtida por sensores de cor e de distância, seria eventualmente possível que algum estado $s(t)$ fosse determinado completamente pela leitura de cor no instante $t-1$ e pela leitura de distância no instante t.

Dessa forma, mesmo se as abordagens baseadas em histórico trazem uma forma simples de se tratar o problema de *perceptual aliasing*, claramente a solução pode não ser a mais econômica em termos de espaço e nem a mais adequada em termos de informação disponível *a priori*.

5.1.2 Abordagens generativas: POMDPs

Diferentemente de modelos baseados em histórico, que são inteiramente descritos em termos de dados observados, os modelos generativos buscam explicar as observações a partir de uma sequência de estados escondidos. Assim, a noção de estado é afastada da noção de observação.

A classe mais conhecida de modelos generativos controláveis são os processos de decisão de Markov parcialmente observáveis (POMDPs). Em tais modelos, são incorporadas todas as características dos MDPs. Além do conjunto $\mathcal{S}$ de estados, já presente nos MDPs, os POMDPs introduzem o conjunto Ω de observações. A princípio, os dois conjuntos são de naturezas diferentes e podem conter diferentes números de elementos. Em consequência, torna-se necessário conhecer a lei generativa, ou seja, uma lei probabilística que rege a ocorrência de observações a partir de cada estado. Em geral, a lei generativa, por ser probabilística, permite que cada estado dê origem a diferentes observações. Em contrapartida, cada observação pode ser originada por diferentes estados.

5.2 Definição e terminologia

Formalmente, um POMDP (Kaelbling et al., 1998; Lovejoy, 1991; Monahan, 1982) é uma tupla $(\mathcal{S}, \mathcal{A}_s, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \mathcal{O}^o_s, \Omega)$, onde $(\mathcal{S}, \mathcal{A}_s, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$ é um MDP como definido na seção 4.1.1, Ω é o conjunto de observações e $\mathcal{O}^o_s$ é a probabilidade de ocorrer uma observação o no estado s (lei generativa de observação), ou seja:

$$\begin{aligned}
\mathcal{P}^a_{ss'} &= \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}, \\
\mathcal{O}^o_s &= \Pr\{o_t = o \mid s_t = s\}, \\
\mathcal{R}^a_{ss'} &= \mathrm{E}\{r_t \mid s_t = s, a_t = a, s_{t+1} = s'\}.
\end{aligned}$$

O conjunto Ω de observações, assim como os estados e ações em um POMDP, pode ser finito, contável ou contínuo, limitado ou não.

Além da propriedade de Markov, expressa pela eq. 4.1, um POMDP possui a propriedade de que a observação o_t depende apenas do estado atual s_t, e de nenhuma ação nem de estados anteriores ou futuros. Assim,

$$\Pr\{o_t = o \mid s_{1:t-1}, a_{1:t-1}, s_t = s, a_t = a\} = \Pr\{o_t = o \mid s_t = s\} = \mathcal{O}^o_s, \tag{5.2}$$

ou seja, a observação o_t, condicionada ao estado s_t, é probabilisticamente independente de quaisquer estados, anteriores ou futuros, e de qualquer ação tomada pelo agente.

Primeiramente, deve-se notar que, num POMDP, o ambiente continua sendo representado por um processo de decisão de Markov. O que muda é que o agente não tem mais acesso ao estado s_t do ambiente, mas apenas à observação o_t a cada instante de tempo t. A decisão de qual ação tomar a cada instante de tempo deve levar em conta apenas a informação disponível, ou seja, a sequência $o_t, a_t, o_{t-1}, a_{t-1}, o_{t-2} \cdots$.

Isto posto, a princípio não parece haver distinção entre um POMDP e um modelo de Markov de ordem k, apresentado anteriormente, já que ambos utilizam o histórico de observações para a tomada de decisão. A diferença, porém, é que, num POMDP, conhecendo-se os modelos de transição $\mathcal{P}^a_{ss'}$ e de observação $\mathcal{O}^o_s$, não é necessário armazenar todo esse histórico. De fato, é possível representar, sem perda de informação, toda a sequência de observações e ações passadas em uma estatística suficiente, um vetor de tamanho fixo chamado de crença do agente.

A **crença** $b_t(s)$ do agente no instante de tempo t é uma distribuição de probabilidades *a posteriori* sobre os estados do sistema, condicionadas a todas as leituras de sensores e ações tomadas anteriormente pelo agente, e conhecendo-se as probabilidades de transição e observação do sistema, aqui chamadas conjuntamente de $\lambda = (\mathcal{P}^a_{ss'}, \mathcal{O}^o_s)$. Assim:

$$b_t(s) \equiv \Pr\{s_t = s \mid o_{1:t}, a_{1:t-1}, \lambda\}. \tag{5.3}$$

O conjunto de crenças do agente é escrito como $\Delta(\mathcal{S})$. O operador Δ é chamado de operador de *randomização* que, aplicado ao conjunto $\mathcal{S}$ de estados, retorna o conjunto de todas as possíveis distribuições de probabilidade sobre o espaço $\mathcal{S}$.

Na definição dada pela eq. 5.3 acima, a crença é calculada considerando-se que já se conhece a leitura sensorial o_t obtida após a execução da ação a_{t-1}. É possível também calcular uma probabilidade *a posteriori* antes de receber a nova leitura. Tem-se então

$$\bar{b}_t(s) \equiv \Pr\{s_t = s \mid o_{1:t-1}, a_{1:t-1}, \lambda\}. \tag{5.4}$$

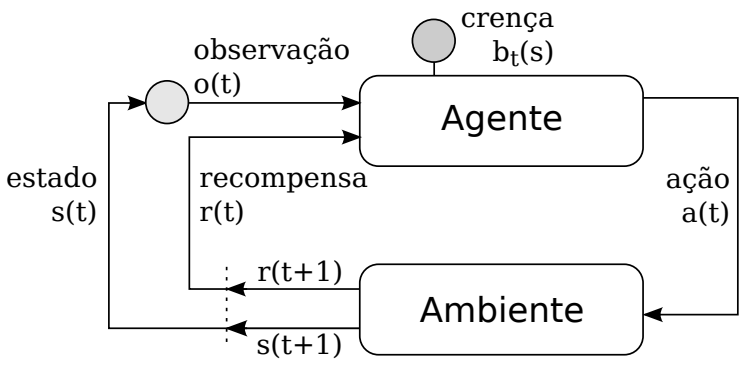


Fig. 5.2: Elementos de interação do agente com o ambiente num ambiente parcialmente observável.

A distribuição $\bar{b}_t(s)$ é chamada de **predição**, pois prediz o estado no tempo t antes de incorporar a observação no tempo t. O cálculo de $b_t(s)$ a partir de $\bar{b}_t(s)$ é chamado de *correção* ou *atualização*.

A Fig. 5.2 resume os elementos de um agente navegando em um ambiente parcialmente observável. Comparando-a com a Fig. 4.1, nota-se que há duas diferenças importantes em relação a ambientes observáveis: a primeira é a diferenciação entre estado e observação, e a segunda é o aparecimento de um elemento a mais no agente: a crença. No caso de ambientes completamente observáveis, o agente é desprovido de qualquer estado interno ou memória, porque pode observar completamente o ambiente a cada intervalo de tempo. No caso de ambientes parcialmente observáveis, o agente necessita de uma crença, que corresponde a um estado interno ou memória temporária, e que é atualizada ao longo do tempo. Assim, a crença é também chamada de estado de informação, porque reflete a informação que o agente possui sobre o ambiente a cada intervalo de tempo.

5.2.1 Filtro de Bayes

Considerando-se conhecidos os parâmetros $\lambda = (\mathcal{P}^a_{ss'}, \mathcal{O}^o_s)$ do POMDP, e conhecendo-se a observação o_t e a ação a_t tomada no tempo t, além da crença $b_{t-1}(s)$ no tempo $t-1$, é possível utilizar o teorema de Bayes para atualizar a crença, ou seja, para calcular $b_t(s)$ para cada estado s. Tal procedimento é chamado de filtro de Bayes. Trata-se de um estimador recursivo, pois utiliza a estimativa de estado no tempo $t-1$ para estimar o estado no tempo t. O filtro de Bayes encontra amplo uso na engenharia, desde a área de processamento de sinais até controle.

O filtro de Bayes compreende dois passos: predição e atualização. Na **predição**, é calculado $\bar{b}_t(s)$ para cada estado s, conhecendo-se a ação a_{t-1}, mas antes de realizar a observação o_t. A dedução a seguir usa as propriedades de Markov e de observação, expressas pelas eqs. 4.1 e 5.2, respectivamente.

A dependência em λ é omitida por clareza. Tem-se:

$$\begin{aligned}
\bar{b}_t(s') &= \Pr\{s_t = s' \mid o_{1:t-1}, a_{1:t-1}\} \\
&= \int_{\mathcal{S}} \Pr\{s_t = s', s_{t-1} = s \mid o_{1:t-1}, a_{1:t-1}\} ds \\
&= \int_{\mathcal{S}} \Pr\{s_t = s' \mid s_{t-1} = s, o_{1:t-1}, a_{1:t-1}\} \Pr\{s_{t-1} = s \mid o_{1:t-1}, a_{1:t-1}\} ds \\
&= \int_{\mathcal{S}} \Pr\{s_t = s' \mid s_{t-1} = s, a_{t-1}\} \Pr\{s_{t-1} = s \mid o_{1:t-1}, a_{1:t-1}\} ds \\
&= \int_{\mathcal{S}} \mathcal{P}_{ss'}^{a_{t-1}}\, b_{t-1}(s) ds, \quad \forall s' \in \mathcal{S}
\end{aligned} \tag{5.5}$$

No segundo passo, chamado de **atualização**, utiliza-se a observação atual para corrigir a predição. A dedução a seguir usa as propriedades das eqs. 4.1 e 5.2, e o teorema de Bayes:

$$\begin{aligned}
b_t(s') &= \Pr\{s_t = s' \mid o_{1:t}, a_{1:t-1}\} \\
&= \frac{\Pr\{s_t = s', o_t \mid o_{1:t-1}, a_{1:t-1}\}}{\Pr\{o_t \mid o_{1:t-1}, a_{1:t-1}\}} \\
&= \frac{\Pr\{s_t = s', o_t \mid o_{1:t-1}, a_{1:t-1}\}}{\int_{\mathcal{S}} \Pr\{s_t = s, o_t \mid o_{1:t-1}, a_{1:t-1}\} ds} \\
&= \frac{\Pr\{s_t = s' \mid o_{1:t-1}, a_{1:t-1}\} \Pr\{o_t = o \mid s_t = s'\}}{\int_{\mathcal{S}} \Pr\{s_t = s \mid o_{1:t-1}, a_{1:t-1}\} \Pr\{o_t = o \mid s_t = s\} ds} \\
&= \frac{\mathcal{O}_{s'}^{o_t} \bar{b}_t(s')}{\int_{\mathcal{S}} \mathcal{O}_{s}^{o_t} \bar{b}_t(s) ds} = \frac{1}{c_t} \mathcal{O}_{s'}^{o_t} \bar{b}_t(s'), \quad \forall s \in \mathcal{S}
\end{aligned} \tag{5.6}$$

onde

$$c_t \equiv \int_{\mathcal{S}} \mathcal{O}_{s}^{o_t} \bar{b}_t(s) ds \tag{5.7}$$

é a constante correspondente ao denominador na fórmula do teorema de Bayes. Tal valor tem por função escalonar $b_t(s)$ para que esta seja uma distribuição de probabilidades com integral unitária. Além disso, como será visto mais tarde, os valores c_t podem ser utilizados para o cálculo da verossimilhança de uma sequência de observações.

A eq. 5.5, na fase de predição, e a eq. 5.6, na fase de atualização, possuem integrais que, dependendo do modelo do sistema, podem não ser calculáveis analiticamente. Na verdade, há poucos modelos conhecidos com fórmulas analíticas para o filtro de Bayes. O mais conhecido deles é o filtro de Kalman, no qual a distribuição de probabilidades de estado é gaussiana e a transição de estados é linear.

O filtro de Bayes também pode ser calculado facilmente se o número de estados for finito. Nesse

caso, a integral da eq. 5.5 se torna um somatório,

$$\bar{b}_t(s') = \sum_{s \in \mathcal{S}} \mathcal{P}_{ss'}^{a_t} b_{t-1}(s), \tag{5.8}$$

e o mesmo ocorre na eq. 5.7,

$$c_t \equiv \sum_{s \in \mathcal{S}} \mathcal{O}_s^{o_t} \bar{b}_t(s). \tag{5.9}$$

Assim, quando o número de estados é finito, torna-se possível implementar o filtro de Bayes utilizando um número finito de operações aritméticas.

5.2.2 Política ótima e função-valor em POMDPs

Num MDP, a política ótima $\pi(s)$ ou a função-valor $V(s)$ da política ótima representam o conhecimento necessário e suficiente para que o agente tome a melhor decisão para qualquer situação que encontrar. Num ambiente observável, o conjunto de situações a que um agente está sujeito corresponde ao conjunto de estados $\mathcal{S}$ do ambiente. Por isso, tanto a política como a função-valor são funções definidas sobre esse conjunto de estados. Num ambiente com três estados, por exemplo, uma tripla de valores define completamente a função-valor.

Num ambiente parcialmente observável, porém, o agente pode se deparar com situações de informação parcial, ou de incerteza, que influenciam na ação ótima a tomar. Por exemplo, quando o agente tem pouca certeza em relação ao estado do ambiente, pode ser melhor tomar uma ação exploratória de custo garantidamente baixo do que uma ação gulosa que pode trazer retornos ótimos ou péssimos, dependendo do estado real do ambiente. Assim, a política do agente e sua função-valor devem ser função não do estado do ambiente, mas da informação que o agente possui sobre o ambiente, ou seja, da crença. Dessa forma, em POMDPs, a função-valor passa a ser $V(b)$ e a política, $\pi(b)$. O exemplo que será dado posteriormente mostra como os POMDPs podem determinar a necessidade de se tomar ações informativas, ou seja, ações que levam à diminuição da incerteza sobre o ambiente.

Tal vantagem, porém, vem com um preço alto. Uma consequência de a função-valor ser função da crença é que, mesmo em um POMDP com número finito de estados, a solução torna-se um problema contínuo, visto que a crença é uma combinação convexa dos estados. Como mostra a Fig. 5.3, enquanto para um MDP com três estados, $\mathcal{S} = \{s_1, s_2, s_3\}$, a solução consiste em encontrar os valores $V(s_1)$, $V(s_2)$ e $V(s_3)$, num POMDP com o mesmo conjunto de três estados a solução é encontrar a definição completa da função $V(b)$, onde $b \in \Delta(\mathcal{S})$. A randomização $\Delta(\mathcal{S})$ é o conjunto de todos os pontos na combinação convexa das crenças puras $b(s_1) = 1$, $b(s_2) = 1$ e $b(s_3) = 1$, ou seja, qualquer ponto do polígono mostrado na figura.

A análise de complexidade de métodos de solução de POMDPs e MDPs em suas versões de de-

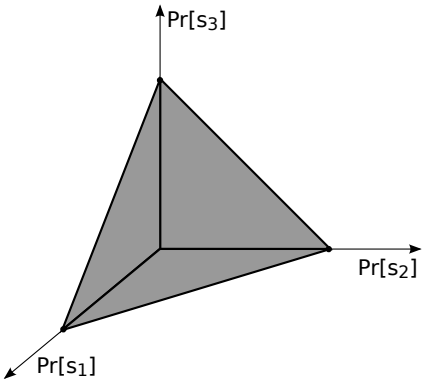


Fig. 5.3: Para um sistema com três estados, a crença pode tomar qualquer valor no interior do polígono da figura, ou seja, qualquer combinação convexa dos três estados.

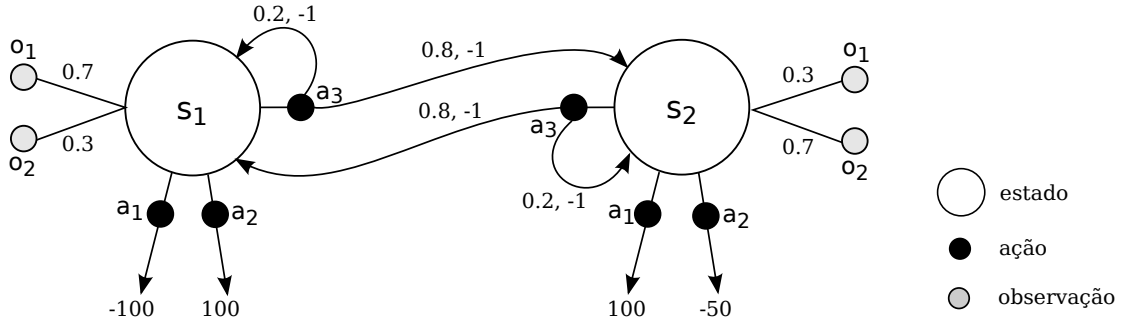


Fig. 5.4: Exemplo simples de POMDP. Na figura, os números sobre os arcos representam a probabilidade de transição seguida da recompensa recebida.

cisão mostra que os MDPs pertencem à classe P-completos, enquanto que os POMDP são PSPACE-completos (Papadimitriou & Tsitsiklis, 1987). A classe PSPACE contém a classe NP, ou seja, problemas PSPACE-completos podem ser considerados mais *complexos* do que problemas NP-completos.

5.2.3 Exemplo

O seguinte exemplo de POMDP foi retirado de Thrun et al. (2006, seção 15.2.1). O exemplo é simples e permite vislumbrar características essenciais de um POMDP. Nesse exemplo, os parâmetros do POMDP são dados. Isso equivale a conhecer o modelo de mundo, dado pelos parâmetros $\mathcal{P}^a_{ss'}$ e $\mathcal{R}^a_{ss'}$, e o modelo de observação do agente, dado pelos parâmetros $\mathcal{O}^a_s$. O problema de aprendizado de tais parâmetros será atacado em seções posteriores.

Considere o mundo de dois estados da Fig. 5.4. Este mundo consiste num MDP com $\mathcal{S} = \{s_1, s_2\}$ e $\mathcal{A} = \{a_1, a_2, a_3\}$. As probabilidades de transição são definidas da seguinte maneira:

$$\mathcal{P}^{a_3}_{s_1,s_1} = 0{,}2 \quad \mathcal{P}^{a_3}_{s_1,s_2} = 0{,}8$$
$$\mathcal{P}^{a_3}_{s_2,s_2} = 0{,}2 \quad \mathcal{P}^{a_3}_{s_2,s_1} = 0{,}8$$

As duas outras ações, a_1 e a_2, levam ao término do processo, que não está representado na figura. As recompensas imediatas são definidas como a seguir:

$$\mathcal{R}_{s_1}^{a_1} = -100 \quad \mathcal{R}_{s_1}^{a_2} = +100 \quad \mathcal{R}_{s_1}^{a_3} = -1$$
$$\mathcal{R}_{s_2}^{a_1} = +100 \quad \mathcal{R}_{s_2}^{a_2} = -50 \quad \mathcal{R}_{s_2}^{a_3} = -1$$

Até aqui, esta poderia ser a definição de um MDP como o da seção 4.1.1. Porém, nesse exemplo, o agente não tem acesso ao estado em que o mundo se encontra a cada instante de tempo. O agente obtém informação sobre o mundo através das observações, que a cada instante de tempo podem tomar um dos dois valores no conjunto $\Omega = \{o_1, o_2\}$. As observações permitem ao agente aumentar o conhecimento sobre o estado atual do sistema. Para isso, deve-se ter conhecimento das probabilidades de observação condicionadas ao estado, que neste exemplo são definidas como a seguir:

$$\mathcal{O}_{s_1}^{o_1} = 0{,}7 \quad \mathcal{O}_{s_1}^{o_2} = 0{,}3$$
$$\mathcal{O}_{s_2}^{o_1} = 0{,}3 \quad \mathcal{O}_{s_2}^{o_2} = 0{,}7$$

O sistema corresponde a um mundo com dois estados. O agente pode sempre escolher entre três ações: a ação a_3 é uma ação exploratória de custo relativamente baixo (-1): o agente pode usá-la para ter mais certeza do estado em que se encontra. Essa ação causa uma mudança de estado com probabilidade 0,8. Já as ações a_1 e a_2 são ações terminais: o agente sabe que tais ações levam ao fim do processo, e podem ter uma recompensa altamente positiva ou altamente negativa, dependendo do estado em que o mundo se encontra. Um agente que agisse de forma ótima deveria saber usar a ação exploratória a_3 um número adequado de vezes antes de executar uma ação terminal, para maximizar a esperança do retorno.

O método que conduz à solução exata para POMDPs consiste em realizar uma busca através do horizonte futuro de decisão, um passo por vez, e para cada ação disponível, e montando a função-valor $V(b)$ com horizonte limitado. Para um horizonte de passo 1, basta avaliar o valor resultante de cada uma das ações a_1, a_2 e a_3. Para um horizonte de passo 2, é necessário, para cada ação do passo 1, testar todas as ações do passo 2. Assim, para um horizonte de passo n, é necessário, a princípio, buscar no espaço de 3^n sequências de ações. Tal número de avaliações é impeditivo mesmo para problemas muito simples, como o deste exemplo.

Em geral, para ambientes com um número finito de estados e para um horizonte de profundidade finita, existe uma propriedade segundo a qual a função-valor é um máximo de funções lineares, ou seja, corresponde a uma função linear por partes. Dessa forma, é possível, de maneira relativamente simples, remover algumas das funções lineares que integram a função-valor, mas que são inúteis devido ao critério de máximo. Essas e outras propriedades levam a uma simplificação dos algoritmos, mas, mesmo assim, soluções exatas para POMDPs são limitadas a problemas com no máximo algumas dezenas estados e para um horizonte relativamente curto.

5.2.4 QMDP

Como mostrado, é inviável utilizar algoritmos para obter a solução exata de POMDPs em ambientes reais com mais de algumas dezenas de estados. Em tais casos, parece ser necessária a utilização de alguma aproximação. Com a aproximação, perde-se a garantia de uma política ótima, e surgem algumas limitações, mas em muitos casos a vantagem trazida pela aproximação suplanta a desvantagem de uma política sub-ótima. Além disso, por continuar seguindo o formalismo de POMDPs, é possível aplicar soluções híbridas, em que a solução exata é usada até um certo ponto, a partir do qual podem-se utilizar aproximações.

A aproximação mais simples, que será utilizada ao longo da pesquisa, é o QMDP (Littman et al., 1995). No QMDP, generaliza-se a função-valor do MDP, calculada no espaço de estados, para o espaço de crenças através do cálculo da esperança. No caso em que o número de estados é finito, tem-se:

$$\begin{aligned} V(b) &\approx \mathrm{E}_s\{V(s)\} \\ &\approx \sum_{s\in\mathcal{S}} \Pr\{s_t = s \mid o_{1:t}, a_{1:t-1}, \lambda\} V(s) \\ &= \sum_{s\in\mathcal{S}} b_t(s) V(s). \end{aligned} \tag{5.10}$$

Esta estratégia simples faz com que a função-valor de um QMDP seja tão simples de calcular quanto a de MDPs. Em contrapartida, para obter essa aproximação, deve-se assumir que o estado do sistema será conhecido em $t+1$, o que geralmente é falso. Dessa forma, perde-se a capacidade de avaliar o valor de uma ação informativa a longo prazo, ou seja, o agente perde a capacidade de utilizar ações para ganhar informação sobre o estado do sistema.

É possível, porém, usar estratégias híbridas com passos POMDP em horizontes de profundidade menor e, a seguir, usar passos QMDP num horizonte mais distante.

Finalmente, uma conexão importante entre os QMDPs e as funções aproximadoras da seção 4.2.3 pode ser tecida considerando-se o que segue. Para cada POMDP existe um MDP formalmente equivalente, no qual o conjunto de estados corresponde ao conjunto de crenças do POMDP original. Dessa forma, o sistema pode ser interpretado como completamente observável no espaço de crenças, já que a cada instante de tempo t, a observação o_t, a crença atual $b_t(s)$ e a ação tomada a_t determinam univocamente a próxima crença, $b_{t+1}(s)$. Pensando dessa forma, seria possível, a princípio, aplicar em POMDPs os métodos de planejamento e aprendizagem por reforço da Seçao 4.2, concebidos originalmente para atuar sobre MDPs. O problema que surge é justamente a explosão do tamanho do conjunto de estados, apresentada na seção anterior através da Fig. 5.3: o conjunto de estados do MDP

corresponderia à combinação convexa do conjunto de estados do POMDP original.

O problema da explosão do número de estados, no caso de MDPs, é atacado através das funções aproximadoras parametrizadas, apresentadas anteriormente na seção 4.2.3. Considerando a interpretação de POMDPs como MDPs, é possível considerar a aproximação QMDP como um MDP definido sobre o conjunto de crenças, mas cuja função-valor é uma função linear definida na base finita de crenças puras (ou seja, crenças em que um dos estados possui probabilidade 1). Esta interpretação será utilizada na seção 6.3 para possibilitar a aprendizagem *online* da função-valor em POMDPs.

Ao longo da pesquisa, será utilizada a aproximação QMDP, por ser a mais simples e por trazer bons resultados em geral, sem levar ao aumento de complexidade dos algoritmos. Devido ao formalismo usado, porém, é possível futuramente implementar aproximações mais elaboradas.

5.3 Aprendizagem em POMDPs

Até aqui, considerou-se apenas o problema do planejamento em POMDPs, ou seja, encontrar a função-valor e a política ótima do agente, considerando conhecidos os parâmetros $\mathcal{P}^a_{ss'}$, $\mathcal{R}^a_{ss'}$ e $\mathcal{O}^a_s$ do POMDP. Quanto ao problema da aprendizagem de parâmetros, não há na literatura, até o momento, grande progresso (Thrun et al., 2006, seção 15.7).

Uma importante limitação da aprendizagem *online* de POMDPs é a impossibilidade de aprender os parâmetros diretamente, já que os parâmetros são função do estado, mas o estado em que o sistema se encontra é desconhecido. Uma aprendizagem *offline* pode, em um primeiro momento, facilitar a solução, já que é possível inferir com maior precisão o estado a partir das experiências passadas e também das futuras.

Outra dificuldade encontrada na aprendizagem de POMDPs é que pode haver mais de um conjunto de parâmetros que leva a uma dada sequência de observações, sob as mesmas ações. Isso vem da natureza abstrata dos estados. Se o agente inicia *tabula rasa*, não há uma regra pré-estabelecida a respeito do que são os estados. Isso é evidenciado pela ampla gama de escolhas de representação de estados encontrada na literatura: em geral, na robótica probabilística, o estado corresponde à pose (posição e orientação) do robô, ou então o nó ativo em um mapa topológico do ambiente. No modelo de Markov de ordem k apresentado na seção 5.1.1, um estado corresponde a uma sequência de observações e ações. Estados podem ainda corresponder a entidades mais abstratas, humanamente interpretáveis ou não. Há ainda a possibilidade de identificar estados a protótipos que levam a uma gama de observações, o que implica uma certa noção de similaridade entre observações.

Um algoritmo completo de aprendizagem tem a responsabilidade de criar uma representação de estados a partir de um agente inicialmente sem conhecimento algum de seu ambiente. O objetivo é encontrar o conjunto de estados mais compacto possível e que, ao mesmo tempo, torne o sistema o

mais markoviano possível. Note, porém, que esses dois objetivos são contraditórios, já que, quanto menor o número de estados, maior a possibilidade de uma quebra da propriedade de Markov.

Um resumo de técnicas de aprendizado em POMDPs encontra-se em (Aberdeen, 2003). Entre as técnicas encontradas, estão os métodos baseados em modelos ocultos de Markov (HMM, do inglês *Hidden Markov Models*), cujo aprendizado é feito através do algoritmo Baum-Welch. Um HMM (Rabiner, 1989) nada mais é do que um POMDP não controlável, ou seja, o sistema possui estados e as observações, mas o agente não tem poder de controlar o sistema através de ações, e não há a noção de recompensa. Na literatura, encontram-se várias extensões do HMM para o caso controlável. Por exemplo, o *Input Output HMM*, proposto por Bengio & Frasconi (1995), inclui um sinal de controle como parâmetro da matriz de transição do HMM. Citam-se também Chrisman (1992); Shatkay & Kaelbling (1997).

Grande parte das propostas citadas acima limita-se a um número finito de estados e observações. No próximo capítulo, será introduzido o mesmo método de aprendizagem de POMDPs, mas com um conjunto Ω de observações contínuo e multi-dimensional.

O algoritmo Baum-Welch pertence à classe de métodos de estimação de parâmetros baseados em maximização da verossimilhança. A seguir, são explicados tais métodos em geral e, posteriormente, o Baum-Welch em particular, adaptado para a utilização em POMDPs.

5.3.1 Aprendizagem de POMDPs por maximização da verossimilhança

Introduz-se aqui a classe de métodos de estimação de parâmetros de um POMDP baseados na maximização da verossimilhança. Tais métodos são apresentados em sua versão *offline*, ou seja, o treinamento dos parâmetros é efetuado apresentando-se os dados em batelada.

Considere um agente observando e executando ações em um ambiente que pode ser representado por um POMDP, mas cujas probabilidades de transição de estados $\mathcal{O}_s^a$ e probabilidades de observação $\mathcal{P}_{ss'}^a$ são desconhecidas. Deseja-se aprender os valores de tais parâmetros, aqui conjuntamente chamados de $\lambda = (\mathcal{O}_s^a, \mathcal{P}_{ss'}^a)$. A princípio, o número de estados também não é conhecido, mas, no presente trabalho, considera-se um número fixo e finito de estados, ou seja, $|\mathcal{S}| = n$, onde n é um parâmetro fornecido ao algoritmo. O conjunto Ω de observações, em contrapartida, pode ser infinito e contínuo, caso no qual $\mathcal{O}_s^a$ deve ser uma função parametrizada, como explicado mais adiante no texto. As recompensas $\mathcal{R}_{ss'}^a$ do POMDP não são consideradas pelo algoritmo, e devem ser aprendidas posteriormente.

Deixa-se o agente navegar pelo ambiente durante um período fixo de T instantes de tempo. Não importa qual a política usada pelo agente, desde que seja uma política exploratória, ou seja, todas as ações devem ter chance de ser executadas sob quaisquer circunstâncias. Ao fim da execução, obtém-se a sequência de observações, $\mathbf{O} = (o_1, o_2 \dots o_T)$, a sequência de ações tomadas, $\mathbf{A} =$

$(a_1, a_2 \ldots a_{T-1})$, e a sequência de recompensas recebidas, $\mathbf{R} = (r_1, r_2 \ldots r_T)$. A ação no tempo T não é importante para efeitos de predição, pois não se conhece sua consequência. Dados um número n pré-definido de estados, e as sequências $(\mathbf{A}, \mathbf{O})$, deseja-se encontrar os parâmetros ótimos λ^* que maximizem a verossimilhança das observações efetuadas, dada a sequência de ações, ou seja:

$$\lambda^* = \arg\max_{\lambda} \log \Pr\{\mathbf{O} \mid \mathbf{A}, \lambda\}. \tag{5.11}$$

Há diferentes maneiras de se resolver o problema acima. Pode-se utilizar o método do gradiente, por exemplo. Uma outra opção é o uso do algoritmo Baum-Welch, que pertence à classe de algoritmos EM (do inglês *Expectation-Maximization*).

5.3.2 Algoritmos Expectation-Maximization - EM

Os algoritmos *Expectation-Maximization* (Dempster et al., 1977) constituem uma classe de algoritmos destinados a encontrar os parâmetros de um modelo probabilístico que maximizem a verossimilhança das observações, como na eq. 5.11. Este método é usado em modelos que possuam, além das variáveis observáveis, variáveis ocultas ou latentes.

Um algoritmo EM fatora a verossimilhança de observação da eq. 5.11 da seguinte forma:

$$\Pr\{\mathbf{O} \mid \mathbf{A}, \lambda\} = \sum_{S} \Pr\{\mathbf{O} \mid \mathbf{S}, \lambda\} \Pr\{\mathbf{S} \mid \lambda\}, \tag{5.12}$$

onde $\mathbf{S}$ representa as variáveis latentes do problema, que, por exemplo, no caso de HMMs e POMDPs, consistem na sequência de estados do sistema.

Os algoritmos EM são iterativos: a cada iteração τ, os parâmetros $\lambda^{(\tau)}$ do modelo são reestimados. Cada iteração do EM é separada em dois passos: o passo de esperança (*E-step*) e o de maximização (*M-step*). O passo de esperança consiste na estimação das variáveis latentes $\mathbf{S}$ condicionadas aos parâmetros $\lambda^{(\tau)}$ e as observações $\mathbf{O}$:

$$\bar{\mathbf{S}}^{(\tau)} = \Pr\{\mathbf{S} \mid \mathbf{O}, \lambda^{(\tau)}\}. \tag{5.13}$$

O passo de maximização, por sua vez, utiliza a estimação $\bar{\mathbf{S}}^{(\tau)}$ e os parâmetros $\lambda^{(\tau)}$ para reestimar os parâmetros $\lambda^{(\tau+1)}$. Isso é feito de forma a resolver o seguinte problema de maximização:

$$\lambda^{(\tau+1)} = \arg\max_{\lambda} \mathrm{E}_{\mathbf{S}|\mathbf{O},\lambda^{(\tau)}} \Big\{ \log \Pr\{\mathbf{O}, \mathbf{S} \mid \lambda\} \Big\}. \tag{5.14}$$

Após número suficiente de iterações, é garantida a convergência para um máximo local da função de verossimilhança. A vantagem de se utilizar o método EM em relação a outros métodos de

maximização de verossimilhança é que o problema de otimização da eq. 5.14 pode ser resolvido analiticamente em vários modelos para os quais o problema original da eq. 5.11 é intratável. Além disso, para modelos com variáveis latentes, o *framework* do método EM se adequa de forma natural, e, além dos parâmetros do modelo, fornece uma estimação para as variáveis latentes.

A primeira iteração do algoritmo EM inicia-se com um passo de estimação. Porém, para que esse passo possa ser executado, é necessário fornecer valores iniciais $\lambda^{(1)}$ para os parâmetros do modelo. Em geral, tais parâmetros são inicializados de forma aleatória. Porém, como os algortimos EM convergem para um máximo *local*, diferentes escolhas para os parâmetros iniciais podem levar a diferentes qualidades para o resultado final do algoritmo. Há várias técnicas para uma escolha que leve a bons ótimos locais.

5.3.3 Algoritmo Baum-Welch para POMDPs

O algoritmo Baum-Welch é um algoritmo do tipo *Expectation-Maximization* originalmente proposto por Baum et al. (1970) para o aprendizado em HMM, e encontra amplo uso, principalmente na área de reconhecimento de fala (Rabiner, 1989). A seguir, é apresentada uma versão deste algoritmo adaptada para uso em POMDPs.

No algoritmo Baum-Welch, o **passo de estimação** consiste em estimar as probabilidades *a posteriori* de estado $\gamma_t(s)$ e de transição $\xi_t(s, s')$, para cada estado $s, s' \in \mathcal{S}$ e para cada instante de tempo $t \in \{1, 2, \ldots, T\}$. Tais probabilididades são assim definidas em cada iteração τ do algoritmo:

$$\gamma_t^{(\tau)}(s) = \Pr\{s_t = s \mid \mathbf{O}, \mathbf{A}, \lambda^{(\tau)}\}, \tag{5.15}$$

$$\xi_t^{(\tau)}(s, s') = \Pr\{s_t = s, s_{t+1} = s' \mid \mathbf{O}, \mathbf{A}, \lambda^{(\tau)}\}. \tag{5.16}$$

A estimação $\gamma_t(s)$ é semelhante à definição de crença da eq. 5.3. Porém, diferentemente da crença $b_t(s)$, que consiste numa estimação do estado apenas a partir das observações e ações passadas, $\gamma_t(s)$ leva em conta também as observações e ações futuras ao instante de tempo t. Isso só é possível porque o algoritmo é executado *offline*. O método de obtenção de $\gamma_t(s)$ e $\xi_t(s, s')$ é chamado de *forward-backward*, e será apresentado na próxima subseção.

O **passo de maximização** do algoritmo Baum-Welch, que consiste na reestimação de parâmetros do modelo, segue a eq. 5.14. Para implementar tal equação, deve-se primeiramente encontrar uma maneira de calcular $\Pr\{\mathbf{O}, \mathbf{S} \mid \lambda\}$. Usando as propriedades de POMDP, tem-se:

$$\Pr\{o_{1:t}, s_{1:t} \mid \lambda\} = \Pr\{o_t \mid s_t, \lambda\} \Pr\{s_t \mid s_{t-1}, \lambda\} \Pr\{o_{1:t-1}, s_{1:t-1} \mid \lambda\}. \tag{5.17}$$

Resolvendo-se a recursão,

$$\Pr\{\mathbf{O}, \mathbf{S} \mid \lambda\} = \Pr\{o_{1:T}, s_{1:T} \mid \lambda\} \tag{5.18}$$

$$= \Pr\{s_1 \mid \lambda\} \Pr\{o_T \mid s_T, \lambda\} \prod_{t=1}^{T-1} \Pr\{o_t \mid s_t, \lambda\} \Pr\{s_{t+1} \mid s_t, \lambda\} \tag{5.19}$$

$$= \pi(s_1)\mathcal{O}_{s_T}^{o_T} \prod_{t=1}^{T-1} \mathcal{O}_{s_t}^{o_t}\mathcal{P}_{s_t,s_{t+1}}^{a_t} \tag{5.20}$$

onde $\pi(s) \equiv \Pr\{s_1 = s \mid \lambda\}$. O termo a maximizar na eq. 5.14 fica, então,

$$\begin{aligned}
&\mathrm{E}_{\mathbf{S}|\mathbf{O},\lambda^{(\tau)}}\Big\{ \log \Pr\{\mathbf{O}, \mathbf{S} \mid \lambda\}\Big\} = \\
&\quad = \mathrm{E}_{\mathbf{S}|\mathbf{O},\lambda^{(\tau)}}\Big\{ \log \pi(s_1) + \sum_{t=1}^{T} \log \mathcal{O}_{s_t}^{o_t} + \sum_{t=1}^{T-1} \log \mathcal{P}_{s_t,s_{t+1}}^{a_t} \Big\} \\
&\quad = \sum_{s\in\mathcal{S}} \gamma_1^{(\tau)}(s) \log \pi(s) + \sum_{s\in\mathcal{S}} \sum_{t=1}^{T} \gamma_t^{(\tau)}(s) \log \mathcal{O}_{s_t}^{o_t} + \sum_{s\in\mathcal{S}} \sum_{s'\in\mathcal{S}} \sum_{t=1}^{T-1} \xi_t^{(\tau)}(s,s') \log \mathcal{P}_{ss'}^{a_t}
\end{aligned} \tag{5.21}$$

ou seja, considera-se que o valor da variável oculta, no caso os estados s_t, é conhecido, e calcula-se a esperança sobre as distribuições de probabilidade de estado $\gamma_{1:T}^{(\tau)}(s)$ e de transição $\xi_{1:T}^{(\tau)}(s,s')$, calculadas no passo de estimação.

Para maximizar a expressão da eq. 5.21 acima, utiliza-se o método do gradiente. Para parâmetros $\mathcal{O}_s^o$ e $\mathcal{P}_{ss'}^a$ discretos ou baseados em distribuições normais, é possível encontrar fórmulas analíticas para as soluções. No capítulo seguinte, será apresentada a parametrização utilizada no trabalho, e as fórmulas de cálculo dos parâmetros serão derivadas.

Método *forward-backward*

Este método, tradicionalmente utilizado em HMMs e aqui modificado para uso em POMDPs, é utilizado para obter a estimação da probabilidade *a posteriori* de estados $\gamma_t(s)$ e de transições $\xi_t(s,s')$, definidas nas eqs. 5.15 e 5.16. Este método utiliza dois processos recursivos de estimação: o primeiro, que utiliza observações e ações anteriores a t, é o processo *forward*. O segundo, que utiliza apenas informações futuras, é o processo *backward*. A combinação dos dois processos permite a estimação de $\gamma_t(s)$.

Processo *forward*

O processo *forward* é equivalente ao filtro de Bayes da seção 5.2.1, com a exceção de que é executado *offline*. O objetivo é estimar, para cada instante de tempo $t \in 1, 2, \ldots, T$ e de forma recursiva, a probabilidade $\alpha_t(s)$ de o sistema se encontrar em cada um dos possíveis estados $s \in \mathcal{S}$, condicionada às observações e ações anteriores a t. Tais probabilidades equivalem à definição de crença $b_t(s)$ das eqs. 5.3 e 5.4. Como subproduto do processo, é possível calcular a verossimilhança da sequência de observações, como será mostrado mais adiante no texto.

Definem-se, inicialmente,

$$\bar{\alpha}_t(s) \equiv \bar{b}_t(s) \equiv \Pr\{s_t = s \mid o_{1:t-1}, a_{1:t-1}, \lambda\}, \tag{5.22}$$

$$\alpha_t(s) \equiv b_t(s) \equiv \Pr\{s_t = s \mid o_{1:t}, a_{1:t-1}, \lambda\}. \tag{5.23}$$

Os valores $\alpha_t(s)$ são chamados de mensagem *forward*, pois consistem na informação que é passada de um instante de tempo ao próximo.

O cálculo das mensagens $\alpha_t(s)$ segue a eq. 5.8 para o passo de predição e a eq. 5.6 para o passo de atualização, para todo $t > 1$. No instante inicial $t = 1$, como não há crença no instante anterior, é necessário pré-determinar uma distribuição inicial de estados. Nesta versão do algoritmo, arbitra-se uma distribuição uniforme de probabilidades, maximizando a entropia, ou seja, minimizando a informação disponível sobre o estado inicial:

$$\bar{\alpha}_1(s) = \Pr\{s_1 = s \mid a_1, \lambda\} = \frac{1}{|\mathcal{S}|}, \quad \forall s \in \mathcal{S} \tag{5.24}$$

Para calcular a verossimilhança da observação $\Pr\{\mathbf{O} \mid \mathbf{A}, \lambda\}$, utiliza-se um subproduto do processo: as constantes de escalonamento do filtro de Bayes, definidas na eq. 5.9. Para derivar a fórmula, define-se:

$$P_t \equiv \Pr\{o_{1:t} \mid a_{1:t-1}, \lambda\}, \tag{5.25}$$

de forma que $P_T = \Pr\{\mathbf{O} \mid \mathbf{A}, \lambda\}$ seja a verossimilhança total desejada. Tem-se, então (omitindo a dependência em λ por clareza),

$$\begin{aligned} P_t &= \Pr\{o_{1:t} \mid a_{1:t-1}\} \\ &= \Pr\{o_t \mid o_{1:t-1}, a_{1:t-1}\}\Pr\{o_{1:t-1} \mid a_{1:t-1}\} \\ &= c_t P_{t-1}, \quad \forall t > 1 \end{aligned} \tag{5.26}$$

Além disso,

$$P_1 = \Pr\{o_1 \mid \lambda\} = \sum_{s \in \mathcal{S}} \Pr\{o_1 \mid s_1 = s\}\alpha_1(s). \tag{5.27}$$

Das eqs. 5.26, 5.27 e 5.24, tem-se:

$$\Pr\{\mathbf{O} \mid \mathbf{A}, \lambda\} = \prod_{t=1}^{T} c_t. \tag{5.28}$$

Dessa forma, o processo *forward* permite calcular a verossimilhança das observações. Na prática, para evitar estouro aritmético, em geral trabalha-se com o logaritmo da verossimilhança,

$$\log \Pr\{\mathbf{O} \mid \mathbf{A}, \lambda\} = \sum_{t=1}^{T} \log c_t. \tag{5.29}$$

Processo *backward*

O processo *backward*, utilizado em conjunto com o processo *forward*, tem por objetivo encontrar a estimação de estados $\gamma_s(t)$. Este processo utiliza apenas observações e ações futuras na estimação. Primeiramente, define-se a mensagem *backward* $\beta_t(s)$ como:

$$\beta_t(s) \equiv \Pr\{o_{t+1}, o_{t+2}, \ldots, o_T \mid s_t = s, a_{t:T}\lambda\}, \quad \forall t < T \tag{5.30}$$

Excepcionalmente, para o tempo final $t = T$, tem-se:

$$\beta_T(s) \equiv 1, \quad \forall s \in \mathcal{S} \tag{5.31}$$

Da forma como foi definida acima, esta mensagem corresponde à probabilidade conjunta de observação futura, dados o estado atual, as ações futuras e o modelo. O cálculo das mensagens $\beta_t(s)$ para todos os instantes de tempo é feita de forma recursiva, iniciando-se em $t = T - 1$ e iterando regressivamente até $t = 1$. Segue a fórmula de recursão:

$$\beta_t(s) = \sum_{s' \in \mathcal{S}}^{n} \mathcal{O}_{s'}^{o_{t+1}} \beta_{t+1}(s') \mathcal{P}_{ss'}^{a}(t), \quad \forall t < T \tag{5.32}$$

Na prática, para evitar estouro aritmético, define-se a mensagem $\hat{\beta}_t(s)$ escalonada. Para tanto utilizam-se novamente os coeficientes do filtro de Bayes definidos na eq. 5.9, e já calculados anteri-

ormente ao longo do processo *forward*. Faz-se

$$\hat{\beta}_t(i) = \frac{\beta_t(i)}{\prod_{w=t+1}^{T} c_w}. \tag{5.33}$$

Isso assegura que a mensagem $\hat{\beta}_t(s)$ mantenha a mesma ordem de grandeza para todos os instantes de tempo t (Rabiner, 1989). A fórmula de recursão para $\hat{\beta}_t(s)$, obtida a partir das eqs. 5.32 e 5.33, fica:

$$\beta_t(s) = c_{t+1}^{-1} \sum_{s' \in \mathcal{S}}^{n} \mathcal{O}_{s'}^{o_{t+1}} \beta_{t+1}(s') \mathcal{P}_{ss'}^{a}(t), \quad \forall t < T \tag{5.34}$$

Finalmente, é possível calcular de forma simples a estimação de estados $\gamma_t(s)$ a partir das mensagens *forward* $\alpha_t(s)$ e *backward* escalonada $\hat{\beta}_t(s)$:

$$\gamma_t(s) = \alpha_t(s)\hat{\beta}_t(s). \tag{5.35}$$

O mesmo para estimação de transição de estados $\xi_t(s, s')$:

$$\xi_t(s, s') = \frac{\alpha_t(s)\mathcal{P}_{ss'}^{a_t}\mathcal{O}_{s'}^{o_{t+1}}\hat{\beta}_{t+1}(s')}{\sum_{w \in \mathcal{S}} \sum_{w' \in \mathcal{S}} \alpha_t(w)\mathcal{P}_{ww'}^{a_t}\mathcal{O}_{w'}^{o_{t+1}}\hat{\beta}_{t+1}(w')}. \tag{5.36}$$

Até este ponto, foi derivado o passo de estimação do algoritmo Baum-Welch, culminando nas eqs. 5.35 e 5.36 acima. O capítulo seguinte especificará em detalhes a arquitetura POMDP utilizada, permitindo assim completar a dedução do passo de maximização. Antes disso, porém, cabe estabelecer relações entre o formalismo apresentado neste capítulo e o conteúdo dos capítulos anteriores.

5.4 POMDPs vistos como redes classificadoras cognitivas

Após a apresentação geral dos processos de decisão de Markov parcialmente observáveis, é tempo de tecer a relação entre tais modelos e as redes classificadoras apresentadas na seção 3.3. De tal comparação poderá decorrer uma aproximação significativa e um diálogo entre pesquisadores no campo da inteligência computacional biologicamente inspirada e de pesquisadores nas áreas de aprendizagem de máquina, controle e pesquisa operacional, que já utilizam o formalismo de POMDP.

Além disso, mostra-se que um agente baseado em POMDP pode se enquadrar de maneira satisfatória no modelo de inteligência proposto por Albus (2002, 2010). Finalmente, ao fim do capítulo, os POMDPs são contextualizados quanto a tentativas de modelagem do neocórtex através de redes bayesianas hierárquicas.

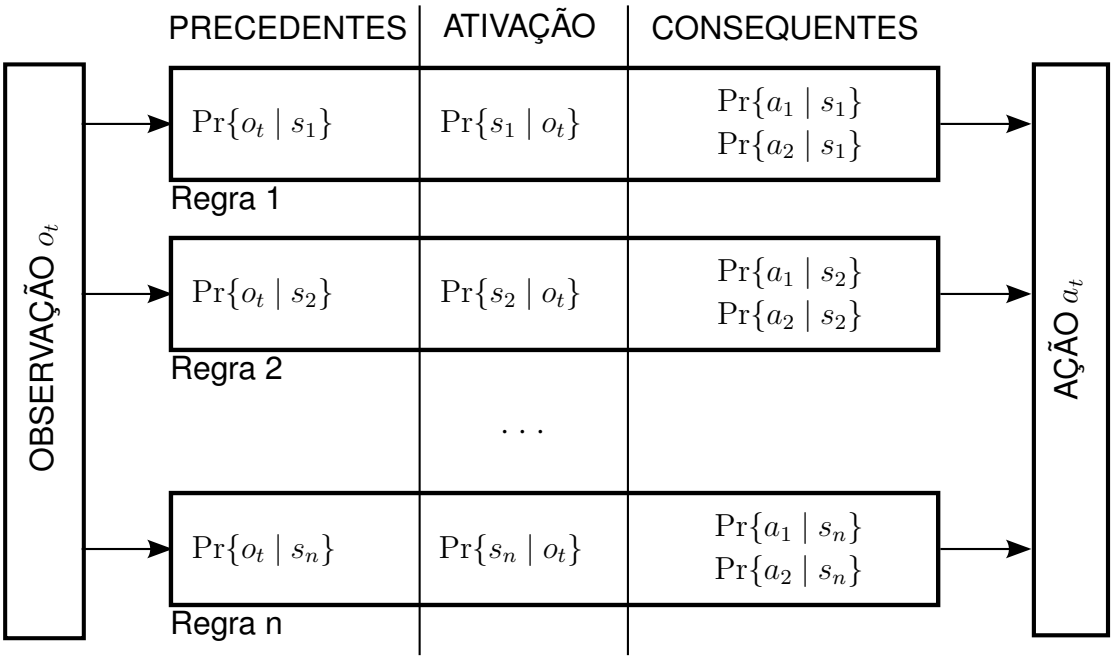


Fig. 5.5: Elementos de um POMDP que compõem um sistema de regras.

5.4.1 POMDPs como redes classificadoras

Pode-se estabelecer uma equivalência muito próxima entre POMDPs e as redes imunlógicas classificadoras, ao ponto que algoritmos desenhados para a síntese em tais redes possam ser aplicados sem muitas modificações na síntese de POMDPs. Primeiramente, mostra-se que alguns dos elementos do POMDP correspondem a um sistema de regras. Em seguida, incluindo-se as transições de probabilidade do POMDP, são estabelecidas conexões entre as regras, e o POMDP pode ser comparado a uma rede classificadora.

Modelo de observação como um sistema de regras

Procura-se primeiramente mostrar que, para um número finito de estados e para um conjunto contínuo de observações, o conjunto de probabilidades condicionais de observação $\mathcal{O}_s^o$, juntamente com a crença $b_t(s)$ e com a política $\pi(s, a)$ do agente, constitui um classificador bayesiano de máxima probabilidade *a posteriori* (MAP, do inglês *maximum* a posteriori), que pode ser considerado um sistema de regras, como apresentado no capítulo 3.

Como mostra a Fig. 5.5, em tal sistema, cada regra corresponde a um estado $s \in \mathcal{S}$. O antecedente de cada regra corresponde à probabilidade condicional de observação, dado o estado. Quando ocorre uma observação o_t, cada regra tem sua pré-ativação calculada através da definição dessa função de

probabilidade condicional. Tem-se então, para cada s, a ativação condicional:

$$\Pr\{o = o_t \mid s\}, \quad \forall s \in \mathcal{S} \tag{5.37}$$

Num sistema classificador bayesiano, é possível usar o teorema de Bayes para calcular a probabilidade *a posteriori* de cada regra, da seguinte forma:

$$\Pr\{s \mid o = o_t\} = \frac{1}{c}\Pr\{o = o_t \mid s\}\Pr\{s\}, \quad \forall s \in \mathcal{S} \tag{5.38}$$

onde $\frac{1}{c}$ é a constante de normalização. Porém, para tanto necessita-se o conhecimento de uma medida de probabilidade $\Pr\{s\}$, conhecida anteriormente à observação o_t. No caso do filtro bayesiano, tal medida corresponde à predição do estado atual, $\bar{b}_t(s)$, e a eq. 5.38 corresponde à equação de atualização, eq. 5.6.

Assim, pode-se dizer que há duas ativações para a regra: uma pré-ativação, que corresponde à probabilidade condicional de observação, e uma pós-ativação, que leva em consideração a predição atual de estados.

O consequente do conjunto de regras corresponde à política do agente. Cada regra possui um vetor que contém a probabilidade de execução de cada uma das ações dadas. Como em qualquer sistema de regras, a seleção da ação a tomar é feita considerando-se a ativação de cada uma das regras. Há duas maneiras de realizar tal seleção: competitiva e colaborativa.

No método **competitivo**, será selecionado o consequente da regra que tiver maior ativação, ou, mais especificamente, maior pós-ativação. No caso do POMDP, isso corresponde a escolher a política do estado $\hat{s}$ cuja probabilidade *a posteriori* seja máxima:

$$\hat{s} \equiv \arg\max_{s \in \mathcal{S}} \Pr\{s \mid o_t\}. \tag{5.39}$$

De posse da seleção da regra (ou estado) $\hat{s}$ de ativação máxima, aplica-se o método da roleta sobre a distribuição de probabilidades

$$\Pr\{a \mid \hat{s}\}, \quad \forall a \in \mathcal{A} \tag{5.40}$$

e a ação selecionada pela roleta é executada.

No método **colaborativo**, todas as regras são executadas, proporcionalmente à sua ativação. Isso corresponde a calcular a probabilidade *a posteriori* de execução de cada uma das ações:

$$\Pr\{a \mid o_t\} = \sum_{s \in \mathcal{S}} \Pr\{a \mid s\}\Pr\{s \mid o_t\}, \quad \forall a \in \mathcal{A} \tag{5.41}$$

Para a seleção da ação a executar, o método da roleta é usado sobre a distribuição $\Pr\{a \mid o_t\}$ definida

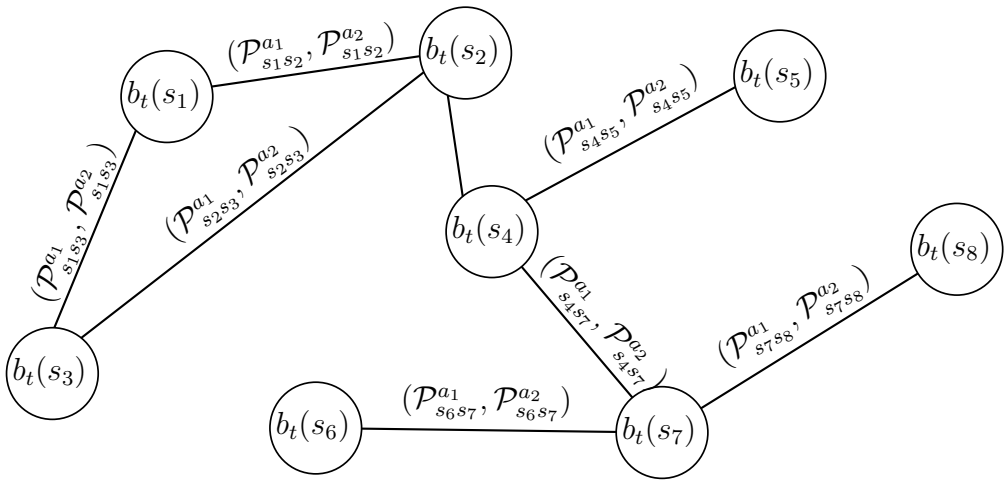


Fig. 5.6: POMDP representado como um grafo, ou uma rede de conexões entre estados.

acima.

A principal diferença entre o sistema de regras explicado acima e aqueles apresentados no capítulo 3 é a presença de consequentes probabilísticos. Isso permite que diferentes ações sejam tomadas mesmo se uma só regra é selecionada. A principal vantagem dessa abordagem é a possibilidade de tomada de ações exploratórias. É possível, dessa forma, implementar algoritmos de aprendizagem de política que, iniciando de uma política altamente aleatória, convirjam para políticas determinísticas numa taxa adequada, resolvendo o conflito entre exploração e explotação.

Além disso, num contexto mais amplo, há vezes em que uma política ótima é probabilística. Por exemplo, pode haver interesse em dificultar a entes externos a predição do comportamento do agente. Tal é o caso em jogos como o par ou ímpar, cuja política ótima consiste em escolher um número par ou ímpar com igual probabilidade.

POMDP como um grafo

Apresenta-se aqui um POMDP de número finito de estados como uma rede, ou, mais formalmente, um grafo, cujos principais elementos são ilustrados na Fig. 5.6. O conjunto $\mathcal{S}$ de estados corresponde ao conjunto de vértices do grafo, e as probabilidades de transição $\mathcal{P}^a_{ss'}$ correspondem a pesos de arestas. Além disso, como mostrado na figura, a cada vértice s é associada a crença $b_t(s)$ no instante de tempo t.

Em geral, as probabilidades de transição são diferentes dependendo da ordem dos estados, ou seja $\mathcal{P}^a_{ss'} \neq \mathcal{P}^a_{s's}$. Assim, o grafo é digirido. Além disso, entre dois estados, em geral, há mais de uma aresta em cada sentido, pois pode haver mais de uma ação que leve de um estado a outro. Vista dessa forma, a rede em questão trata-se de um multi-grafo. Alternativamente, pode-se considerar pesos

RIC	POMDP	Símbolo
Antígeno	Observação	$o \in \Omega$
Anticorpo/Regra	Estado	$s \in \mathcal{S}$
Consequente	Ação	$a \in \mathcal{A}$
Antecedente	Prob. Cond. de Observação	$\mathcal{O}_s^o = \Pr\{o \mid s\}$
Sistema de regras	Política	$\pi(a, s) = \Pr\{a \mid s\}$
Ativação das regras	Crença	$b_t(s) = \Pr\{s_t \mid o_{1:t}, \lambda\}$
Conexão entre regras	Prob. de Transição de Estados	$\mathcal{P}_{ss'}^a = \Pr\{s' \mid s, a\}$
Fitness	Retorno acumulado	$\mathcal{R}_{ss'}^a = \mathrm{E}\{r_t \mid s, s', a\}$

Tab. 5.1: Analogia entre POMDPs e as redes imuno-classificadoras de Cazangi (2008).

vetoriais para cada uma das arestas, como representado na Fig. 5.6.

A princípio, o grafo é completo, ou seja, todos os estados estão ligados entre si, porque não há restrições nos valores de probabilidade de transição do POMDP. Porém, dependendo do ambiente, há pares de estados cuja probabilidade de transição é nula, qualquer que seja a ação tomada. Nesses casos, é conveniente considerar que não há aresta conectando tais pares de estados.

Também associadas às arestas, mas não mostradas na Fig. 5.6, estão as recompensas $\mathcal{R}_{ss'}^a$ esperadas para cada transição. E a cada vértice ou estado correspondem as probabilidades de observação $\mathcal{O}_s^o$ para cada uma das possíveis observações $o \in \Omega$.

POMDP como uma rede classificadora

Finalmente, juntando as interpretações realizadas acima, um POMDP pode ser comparado às redes classificadoras apresentadas na seção 3.3. A Tabela 5.1 apresenta de forma completa a analogia. A seguir, são apresentados aspectos comuns e diferenças entre as duas abordagens.

Primeiramente, deve-se frisar que, enquanto na abordagem de Cazangi os valores de conexão e de ativação eram ilimitados, no POMDP tais valores são distribuições de probabilidade, e portanto são restritos ao intervalo $[0, 1]$ e devem possuir soma unitária. Outra diferença importante é, naquela abordagem, as conexões entre nós são únicas e bilaterais. Assim, para cada par de regras, as redes classificadoras apresentavam no máximo um coeficiente de conexão. Nos POMDPs, cada par de estados possui vários coeficientes: dois para cada ação, já que o sentido da conexão é importante.

Apesar dessas diferenças, é interessante notar que há vários aspectos comuns entre ambos. Primeiramente, em ambas as abordagens cada uma das regras tem uma concentração $a_i(t)$, ou crença $b_t(s)$, que se propaga e é modificada ao longo do tempo, e que influencia a ativação de regras vizinhas em instantes futuros. No sistema de Cazangi, a evolução da concentração é determinada por uma equação diferencial (eq. 3.2), enquanto que num POMDP, a evolução da crença se dá por um filtro de Bayes (eqs. 5.5 e 5.6). Em um caso ou outro, crença ou concentração correspondem à memória

de curto prazo, ou memória de trabalho (Kellogg, 2007, capítulo 4) do agente, que permite que ele expresse comportamentos não-reativos.

Outro aspecto em comum é a forma com que a observação influencia a concentração de cada regra. Na abordagem de Cazangi, é calculada uma similaridade, definida na eq. 3.1, entre a observação realizada e o protótipo de cada regra. Num POMDP com observações contínuas, é possível modelar as propabilidades condicionais de observação através de funções de base radial. Assim, a cada estado corresponderá um protótipo, e a probabilidade de observação será tão maior quanto mais próxima a observação estiver do protótipo. Essa abordagem será explicada com mais detalhes no capítulo seguinte.

Finalmente, a ação executada, em ambos os sistemas, depende da concentração de cada regra após a realização da observação. O sistema de Cazangi executa a ação dada pela regra de maior concentração. Isso é essencialmente o mesmo que faz a eq. 5.39, selecionando o estado com maior probabilidade *a posteriori*.

Com tais analogias tecidas, fica claro que é facilmente possível adaptar os algoritmos descritos por Cazangi para a utilização na síntese de POMDPs. Ao longo da pesquisa, porém, optou-se por utilizar algoritmos diferentes, que exploram o caráter probabilístico de tais redes. Esta abordagem foi escolhida porque permite uma maior formalização dos conceitos envolvidos, com garantia de convergência para um ótimo local. Em trabalhos futuros, será possível adotar uma abordagem híbrida, que utilize algoritmos de maximização de verossimilhança para busca local e algoritmos genéticos para busca global.

5.4.2 POMDPs como sistemas cognitivos

Dentre as poucas teorias que buscam modelar a inteligência em todos os seus aspectos, encontra-se a teoria de Albus (2002). Esta teoria modela a inteligência essencialmente como um controlador hierárquico, em que a cada camada da hierarquia seria designado o controle do agente em uma escala específica de espaço-tempo. Cada camada da hierarquia seria composta por inúmeras unidades controladoras, especializadas em alguma tarefa limitada espaço-temporalmente pela escala designada à camada em questão. Cada unidade controladora teria capacidade de comunicar-se com as unidades vizinhas, tanto horizontalmente (na mesma camada) quanto verticalmente (em camadas diferentes), de forma comparável à arquitetura de subsunção de Brooks (1986), ou então à Sociedade da Mente, de Minsky (1988).

Albus propõe que o sistema inteligente como um todo, assim como cada uma de suas unidades formadoras, seja composto por quatro elementos:

- **Processamento sensorial**: “compara observações sensoriais com expectativas geradas por um

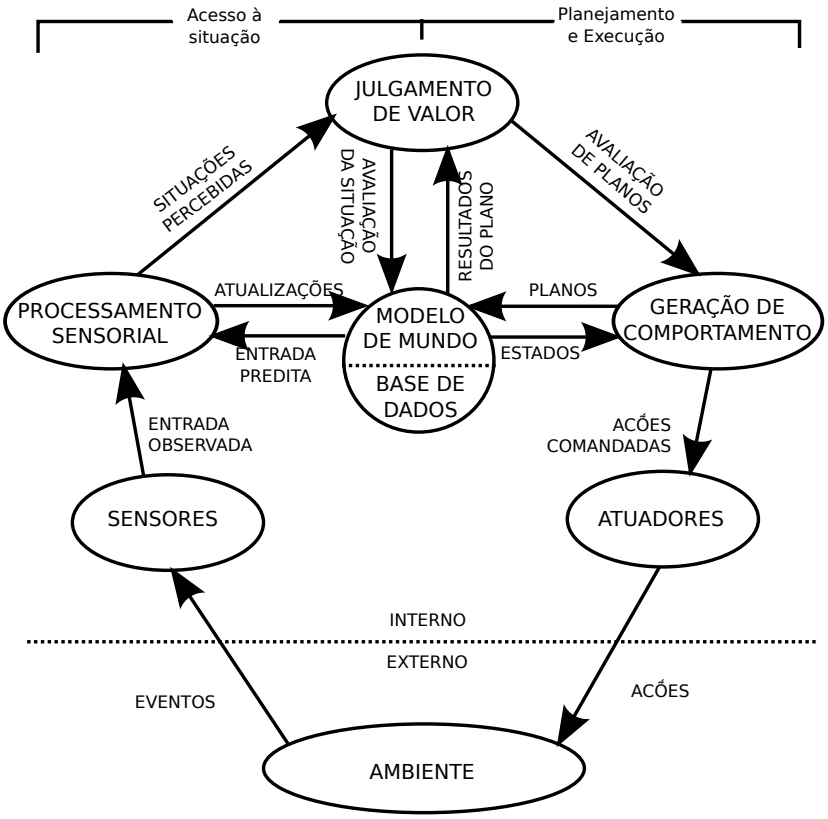


Fig. 5.7: Elementos da inteligência e relações funcionais entre eles (Albus, 2002).

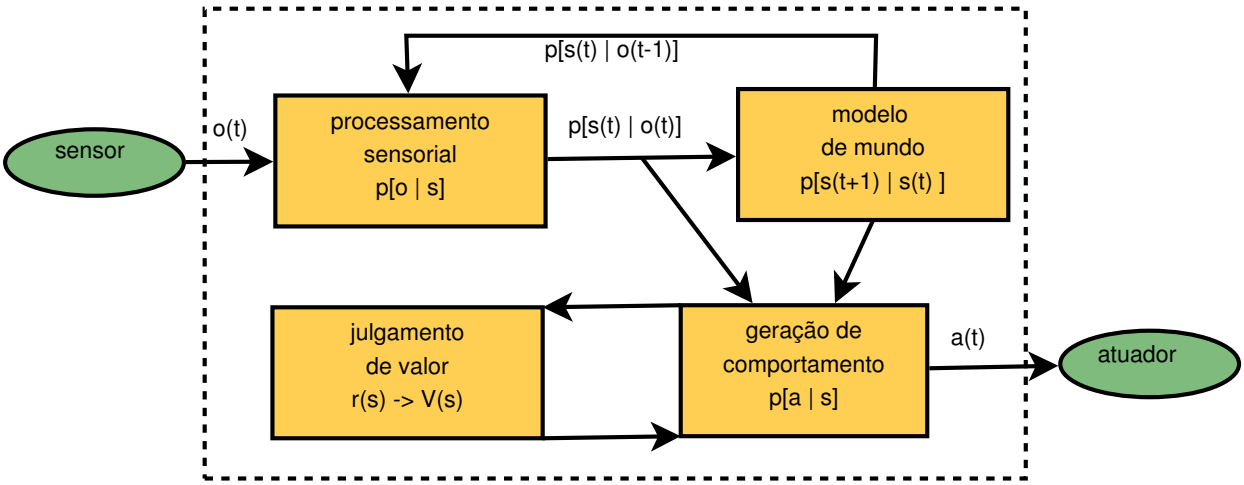


Fig. 5.8: Elementos do POMDP que o enquadram no modelo de inteligência de Albus (2002).

modelo interno de mundo. Algoritmos de processamento sensorial integram similaridades e diferenças entre observações e expectativas ao longo do tempo e espaço para detectar eventos e reconhecer características, objetos e relações no mundo. Dados sensoriais de ampla gama de sensores ao longo do tempo são fundidos em uma percepção unificada do estado do mundo."

- **Modelo de mundo**: "é a melhor estimativa que o sistema inteligente possui sobre o estado do mundo. Inclui um banco de dados de conhecimento sobre o mundo, e um sistema de gerenciamento que armazena e recupera informações. O modelo de mundo também é capaz de simular e de gerar expectativas e predições. Assim, o modelo de mundo pode fornecer respostas a requisições de informação sobre o presente, passado e possíveis estados futuros do mundo." O modelo de mundo é constantemente atualizado em função da experiência, ou seja, há aprendizado e adaptação.

- **Julgamento de valor**: "determina o que é bom e ruim, o que traz recompensa e punição, o importante e o trivial, o certo e o improvável. O sistema avalia tanto o estado observado do mundo quanto o resultado predito de planos hipotéticos. Computa riscos, custos e benefícios (...). Também atribui atratividade, repulsa a objetos, eventos, regiões do espaço e outras criaturas."

- **Geração de comportamento**: "seleciona objetivos, planeja e executa tarefas. (...) O sistema de geração de comportamento hipotetiza planos, o modelo de mundo prediz os resultados de tais planos, e o elemento de julgamento de valor avalia tais resultados."

A Fig. 5.7 apresenta os quatro elementos de um sistema inteligente, além de fornecer um esboço da interação de uns elementos com os outros. Mais amplamente, a figura exibe o laço completo de controle, do qual fazem parte também o ambiente, e os sensores e atuadores do agente.

Pretende-se mostrar aqui que o formalismo de POMDP, aliado a um algoritmo de aprendizagem de parâmetros e a uma cuidadosa definição das recompensas imediatas, satisfaz esta definição de sistema inteligente. Para isso, mostra-se como o POMDP implementa as funções de cada um dos quatro elementos apresentados. A Fig. 5.8 dá uma visão geral da ligação entre o modelo de Albus e o proposto neste trabalho.

Primeiramente, a forma como o elemento de **modelo de mundo** é implementada decorre diretamente do fato que um POMDP é um modelo generativo. A "melhor estimativa que o sistema inteligente possui sobre o estado do mundo" corresponde à crença $b_t(s)$, definida na eq. 5.3. O "banco de dados" referido na descrição acima armazena os parâmetros $\lambda = (\mathcal{O}^o_s, \mathcal{P}^a_{ss'})$ do modelo. O preditor e simulador do sistema é implementado através do passo de predição do filtro de Bayes, definido pela eq. 5.5. Tal equação pode ser usada iterativamente para prever o estado do sistema k passos adiante, resultando na predição $\bar{b}_{t+k}(s)$.

A implementação do **processamento sensorial** decorre também da definição de modelo generativo: as unidades processadoras correspondem a mecanismos que implementam a função de probabilidade de observação $\mathcal{O}_s^o$ para cada sensor. Ou seja: os sinais de entrada sensorial o_t específicos a cada um dos sensores são convertidos em vetores de probabilidades $\Pr\{o_t \mid s\}$ sobre os estados, de forma a representar uniformemente as leituras de todos os sensores.

Com todas as leituras expressas em um mesmo tipo de sinal, no caso uma distribuição de probabilidades, é possível compará-las entre si e, principalmente, compará-las com a predição de estado $\bar{b}_t(s)$. Tal função é executada pelo passo de atualização do filtro de Bayes, dado pela eq. 5.6. É nesse passo que ocorre a fusão sensorial: a equação é aplicada em sequência para uma diferente gama de sensores, cada um contendo uma determinada quantidade de informação quanto ao estado. A referida representação "consistente e unificada sobre o estado do mundo" corresponde então à crença $b_t(s)$, após a aplicação da equação de atualização.

O **julgamento de valor**, em um POMDP, consiste em duas partes: no modelo $\mathcal{R}_{ss'}^a$ de recompensas imediatas, e da função-valor $V(b)$ da crença. A função-valor $V(b)$ é obtida através do algoritmo de planejamento em POMDPs, e a aprendizagem do modelo $\mathcal{R}_{ss'}^a$ de recompensas imediatas pode ser realizada pelo algoritmo de aprendizagem a partir dos valores de recompensa r_t obtidos ao longo do tempo.

As recompensas imediatas r_t recebidas pelo sistema cognitivo constituem o conjunto de valores intrísecos do agente, que não pode ser modificado ao longo de sua existência, e que deve depender apenas de sinais sensoriais imediatos. Na teoria de Albus, tais recompensas imediatas correspondem ao valor das variáveis por ele denomidadas "variáveis de estado interno de baixo-nível". A dor física e a fome, são exemplo de valores negativos de tais variáveis. O *framework* de aprendizagem por reforço assume que as recompensas r_t são dadas pelo ambiente. Porém, um agente completo deve possuir sua própria definição de recompensas imediatas que, quando expostas ao ambiente, definem seu comportamento.

Outras variáveis de estado, citadas e classificadas por Albus como sendo de médio e alto nível, estão diretamente relacionadas a valores $V(b)$ da crença, ou seja, levam em consideração o resultado do algoritmo de planejamento no POMDP. Assim, esperança corresponderia a um $V(b)$ positivo para a crença $b_t(s)$ atual, ou seja, a expectativa de recebimento de boas recompensas futuras. Desespero corresponderia a um $V(b)$ altamente negativo, ou seja, incapacidade do algoritmo de planejamento de encontrar ações que levem a boas recompensas futuras.

Há também variáveis de estado que podem ser relacionadas diretamente ao algoritmo de aprendizagem. Alegria (*joy*), no sentido de surpresa positiva, corresponderia por exemplo a um valor altamente positivo de diferença temporal (eq. 4.7), ou seja, o recebimento de uma recompensa r_t maior que a esperada.

Finalmente, algumas variáveis de estado podem ser obtidas diretamente a partir do cálculo da entropia de Shannon da crença, $\mathrm{H}_{\mathcal{S}}\{b_t(s)\}$: *confiança* corresponde a valores baixos de entropia, enquanto que *incerteza* corresponde a valores altos. Uma propriedade interessante dos POMDPs é que a função-valor $V_t(b)$ é sempre convexa. Isso significa que crenças formadas pela combinação de estados, ou seja, crenças com alta entropia, possuem valores $V(b)$ menores ou iguais ao de crenças puras, ou seja, crenças de entropia nula. Assim, por uma propriedade intrínseca dos POMDPs, uma confiança elevada está ligada a uma expectativa boa, e uma forte incerteza, a uma expectativa ruim.

Note-se que os termos usados aqui para explicar o julgamento de valor (dor, desespero, alegria, etc) correspondem àqueles citados por Albus, e tenta-se exclusivamente tecer uma relação entre POMDPs e o modelo por ele proposto. Não há razão para afirmar que agentes dotados de tais arquiteturas realmente *sofram* as experiências subjetivas em geral denotadas por tais termos. Tal afirmação recairia num problema filosófico, o problema dos *qualia*, cuja discussão é, pelo menos por enquanto, puramente especulativa (Nagel, 1974).

Finalmente, a **geração de comportamento** consiste no algoritmo de planejamento do POMDP, já mencionado acima, e num seletor de ações. A princípio, a ação a executar é simplesmente aquela que leva mais provavelmente a uma crença de maior valor $V(b)$. Para tanto, são utilizadas as probabilidades de transição de estado $\mathcal{P}^a_{ss'}$, fornecidas pelo modelo de mundo. Um algoritmo de planejamento completo para POMDP é capaz inclusive de gerar ações visando à diminuição de incerteza. Isso ocorre porque a função-valor da crença é convexa: crenças puras, com menos entropia, possuem valor mais elevado, e por isso as ações que levam a tais crenças são priorizadas.

As comparações tecidas acima tornam razoável a classificação de um POMDP, acrescido de um algoritmo de aprendizagem e de uma definição da função de recompensas, como um sistema cognitivo. O nível de cognição atingido, a princípio, depende do tamanho e parametrização do POMDP.

Além disso, tanto as entradas sensoriais quanto as ações a executar a princípio podem ser contínuas ou discretas, de baixo ou alto nível. Assim, a arquitetura POMDP pode ser utilizada para implementar individualmente cada uma das unidades de processamento no interior de cada camada proposta por Albus. Numa arquitetura formada por unidades probabilísticas, as mensagens trocadas, tanto entre unidades do mesmo nível hierárquico quanto entre nívis diferentes, seriam implementadas como medidas de probabilidade, e o resultado seria uma rede bayesiana hierárquica. A seguir mostra-se como tais modelos vêm sendo utilizados para modelar o funcionamento do cérebro de forma biologicamente plausível.

Modelos POMDP hierárquicos e neocórtex

Apesar de tal fato não ser explorado nesse projeto, cabe mostrar como modelos POMDP podem ser prontamente estendidos para modelos hierárquicos, levando a uma arquitetura biologicamente

plausível para o funcionamento do neocórtex, e de acordo com o modelo hierárquico de cognição proposto por Albus.

O modelo de POMDP hierárquico, ou H-POMDP (Theocharous et al., 2001), vem sendo aplicado satisfatoriamente para resolver de forma unificada problemas de navegação robótica (Mahadevan et al., 2005). É interessante que tal formalismo possa resolver de forma completa um problema que, como mostrou-se no cap. 2, tradicionalmente necessita de diferentes tipos de algoritmos para a resolução de cada um de seus sub-problemas.

Além disso, H-POMDPs podem ser considerados como um caso especial de rede bayesiana dinâmica (**?**). Nos últimos anos, a abordagem bayesiana hierárquica, além de sua aplicação na resolução de problemas de estimação e inferência, vem sendo usada para modelar o funcionamento do neocórtex dos animais superiores. Como mostrado na seção anterior, os POMDPs enquadram-se satisfatoriamente como um elemento de cognição na arquitetura cognitiva de Albus. A união de POMDPs em diferentes níveis hierárquicos, formando um H-POMDP, poderia então implementar tal arquitetura por completo, considerando toda a hierarquia de escalas de espaço-tempo proposta.

No seu livro, Hawkins & Blakeslee (2005) propõem uma arquitetura bayesiana hierárquica para o neocórtex cerebral, seguindo o modelo de colunas corticais (Mountcastle, 1978) vistas como unidades genéricas de computação. O neocórtex é apresentado como um computador probabístico de uso geral, capaz de realizar predição e correção de crença através de uma estrutura hierárquica, partindo de níveis mais baixos diretamente ligados à percepção até níveis superiores mais abstratos. Ressalta-se que arquiteturas de rede bayesiana podem ser interpretadas como redes neurais probabilísticas, nas quais as conexões entre neurônios transmitem medidas de probabilidade.

Nesse sentido, a abordagem tomada nesse trabalho e, mais genericamente, em redes bayesianas, pode ser considerada como uma abordagem conexionista, na qual o processamento ocorre de forma paralela em cada nó da rede. A ativação de cada nó é um número entre 0 e 1 que corresponde a uma medida de probabilidade, e as conexões entre nós correspondem a dependências entre as medidas de probabilidade de cada nó. O sistema cognitivo como um todo é visto como uma rede na qual a representação do conhecimento é distribuída e sub-simbólica, e o processamento é efetuado de forma vetorial.

Dean (2005) mostra que a atual capacidade de computação, com o fortalecimento da computação paralela, já permite a simulação de um neocórtex artificial baseado em redes bayesianas hierárquicas. A missão a longo prazo seria, então, "modelar o cérebro em termos de um modelo generativo no qual a realimentação é usada para resolver ambiguidades".

Há também crescente interesse pela formulação modelos bayesianos hierárquicos em áreas mais específicas da cognição, como atenção e visão (George & Hawkins, 2005; Chikkerur et al., 2009). Uma das vantagens de se modelar aspectos partirculares da cognição sobre um *framework* comum,

no caso a metodologia bayesiana, é a maior facilidade de posterior integração.

É possível também estabelecer uma íntima relação entre gramáticas probabilísticas livres de contexto (Charniak, 1996) e POMDP hierárquicos (Theocharous et al., 2004). Dessa forma, a aprendizagem de uma linguagem pode ser generalizada para a aprendizagem de estados do ambiente em diferentes modalidades sensoriais. Tal gramática modelaria objetos e eventos de forma hierárquica, iniciando em níveis inferiores da hierarquia com pixels e frequências sonoras, e subindo ao longo da hierarquia numa escala espaço-temporal cada vez maior, modelando texturas e sons, passando por objetos e fonemas, e indo até a descrição de cenas completas em níveis mais altos da hierarquia, realizando fusão sensorial ao longo da subida hierárquica. Além disso, a cada estado de cada nível hierárquico seria associado um valor $V(s)$, possibilitando planejamento e controle.

Em resumo, tais trabalhos apontam para a modelagem do cérebro humano, ou pelo menos da cognição humana, como um controlador hierárquico, probabilístico e adaptativo, estimando e aprendendo através de inferência bayesiana. Modelagem esta que se encontra cada vez menos distante de uma viabilidade prática, devido ao grande aumento da capacidade de processamento ao longo das últimas quatro décadas, e ao crescente desenvolvimento de arquiteturas de processamento paralelo nos últimos anos.

Assim, mesmo o presente projeto tratando de POMDPs apenas na sua forma mais simples, é possível enquadrá-lo, conceitualmente e formalmente, em um programa de pesquisa muito mais amplo e que encontra crescentes adeptos de diferentes áreas do conhecimento relacionadas a ciência cognitiva e robótica.

5.5 Resumo

Este capítulo partiu de algumas limitações impostas pelo modelo de MDP para, a partir delas, introduzir os POMDPs. É apresentado o formalismo, são dados exemplos de POMDP, apresentam-se métodos de aprendizagem de POMDP e, finalmente, esboça-se um paralelo entre POMDPs e as redes imuno-classificadoras de Cazangi (2008).

- Uma severa limitação dos MDPs é não possuir a noção de observação. Assim, em um MDP, confunde-se estado e observação e considera-se que o agente possui acesso completo ao estado do sistema. Em robótica móvel, isso não é razoável, devido a dois fenômenos: o *sensor-aliasing*, em que estados diferentes levam à mesma leitura sensorial, e o ruído dos sensores, em que um mesmo estado leva a leituras sensoriais diferentes.

- O POMDP é um modelo que distingue o conceito de estado do conceito de observação. Num POMDP, considera-se que as observações são geradas por uma distribuição de probabilidades

condicionada ao estado do sistema. Assim, o modelo incorpora tanto ruído sensorial quanto *sensor-aliasing*.

- A resolução do problema de planejamento no POMDP é mais complexa que em um MDP, pois deve-se trabalhar com informação incompleta: a política do agente é função não do estado, mas da probabilidade sobre os estados (crença), que consiste num domínio de dimensionalidade infinitamente maior. Para lidar com essa explosão de dimensionalidade, introduzem-se aproximações ao POMDP, como por exemplo o QMDP.

- A aprendizagem de um POMDP consiste na estimação dos parâmetros de transição de estados e dos parâmetros das distribuições de probabilidade condicionais de observação. Para tanto, o capítulo apresenta o método Baum-Welch de treinamento, um método *offline* que busca atingir um máximo de verossimilhança através da técnica de Expectation-Maximization.

- Uma vez apresentados métodos de resolução e aprendizagem de POMDPs, o capítulo traça paralelos entre um POMDP e as redes imuno-classificadoras de Cazangi (2008). Cada anticorpo corresponde a um estado do POMDP, as conexões entre anticorpos correspondem às probabilidades de transição de estados, e a concentração dos anticorpos corresponde à crença do agente.

- Finalmente, enquadra-se o POMDP, enquanto modelo bayesiano, como peça de um ousado programa de pesquisa cujo objetivo último é modelar funcionalmente o neo-córtex dos animais superiores.

Capítulo 6

Síntese e aprendizagem em POMDPs para navegação

A problemática geral da navegação em robôs autônomos foi apresentada no cap. 2. Em seguida, no cap. 3, apresentaram-se algumas abordagens específicas para a resolução de tal problema. No caps. 4 e 5, foram introduzidos formalismos e metodologias de aprendizagem e de resolução de problemas de tomada de decisão em geral. Neste capítulo, todos esses elementos serão utilizados em conjunto na proposta de uma nova forma de se resolver o problema de navegação.

Primeiramente, o POMDP, que foi explicado de forma genérica no capítulo anterior, será especificado em mais detalhes. As parametrizações utilizadas serão explicitadas, permitindo que o projeto do algoritmo de aprendizagem, especificamente a parte de reestimação de parâmetros, seja concluído. A parametrização utilizada é fortemente inspirada nas redes imuno-classificadoras da seção 3.3.

Em seguida, procede-se à explicação dos algoritmos e processos utilizados e de como os POMDPs e métodos de aprendizagem por reforço são conjuntamente inseridos no ciclo de controle do robô a fim de resolver o problema de navegação. Além disso, procede-se à especificação do algoritmo de gradiente de política com uma parametrização da política específica para o trabalho em questão.

6.1 Trabalhos relacionados

Antes de proceder à aplicação dos métodos de POMDP e aprendizado à robótica, cabe fazer uma breve revisão bibliográfica.

A aplicação de métodos de aprendizado por reforço ao problema de navegação em robótica é um assunto clássico. Trabalhos como Smart et al. (2002); Bakker et al. (2006); Lin (1992) utilizam algoritmos baseados na maximização da função-valor, empregando parametrização.

O uso de POMDP para resolver problemas de decisão em robótica móvel vem sendo cada vez

maior. Além disso, tem-se dado ênfase à difícil tarefa de aprendizado em POMDPs. Em vários trabalhos, como por exemplo Chrisman (1992); Shatkay & Kaelbling (1997); Theocharous et al. (2001), diferentes variantes do algoritmo Baum-Welch são utilizadas para o treinamento de POMDPs baseados em HMM. Porém, tais trabalhos usam um conjunto de observações finito e discreto. Nikovski (2002) realiza alguns experimentos com parametrização de uma observação unidimensional, especificamente para um sensor do tipo bússola.

Quanto à POMDP aliado ao algoritmo de aprendizado por gradiente de política, Aberdeen (2003), em sua tese de doutorado, explora profundamente formas de aprendizado de POMDP utilizando-se um gradiente de política. Porém, a área de aplicação voltou-se principalmente a reconhecimento de fala. Os poucos experimentos de robótica apresentados na tese focaram na capacidade de planejamento com informação incompleta, e possuem um conjunto discreto de observações num ambiente discretizado formando um quadriculado.

Algoritmos de subida no gradiente natural aliados ao aprendizado por reforço surgiram recentemente (Peters et al., 2003; Morimura et al., 2005; Peters & Schaal, 2008a), e há poucos algoritmos propostos no assunto. A maioria é semi-offline e baseada em métodos TD. Bhatnagar et al. (2008) apresentam algumas abordagens *online*. Chega-se a uma proposta semi-offline baseada em Monte Carlo, para a qual não foi encontrada descrição detalhada na literatura.

6.2 Parametrização do POMDP

No capítulo anterior, os POMDPs foram apresentados de maneira geral. Pouco foi especificado sobre a natureza dos conjuntos de observação Ω, de estados $\mathcal{S}$, de ações $\mathcal{A}$ e, por consequência, sobre a natureza das funções de probabilidade de transição $\mathcal{P}^a_{ss'}$ e de observação $\mathcal{O}^o_s$.

Como já foi mencionado, tais conjuntos podem ser limitados ou ilimitados, discretos ou contínuos. A escolha aqui efetuada procura aproximar o máximo possível o POMDP utilizado de uma rede imuno-classificadora. Ao mesmo tempo, busca-se englobar o máximo possível do problema de controle do robô no funcionamento do POMDP. Buscou-se também implementar o POMDP de forma compatível com modelos de cognição existentes. Finalmente, em alguns casos a escolha é feita de modo a simplificar a implementação. Dadas essas premissas, procede-se à escolha dos conjuntos de observação, estado e ação.

Para o conjunto Ω de observação, buscou-se a maior proximidade possível com os dados brutos do sensor. Dessa forma, mantém-se uma fidelidade com a proposta de modelar todas as etapas do processo em um POMDP. Além disso, segue-se a analogia com as redes classificadoras. Assim, define-se o conjunto observação:

$$\Omega \equiv [0, \mathrm{MAX}]^d \in \mathbb{R}^d_+, \tag{6.1}$$

ou seja, Ω é um conjunto contínuo, limitado e multi-dimensional. Cada uma das d dimensões corresponde à leitura escalar de um sensor. Uma observação $o_t \in \Omega$ corresponde então a uma tupla de leituras realizadas por d sensores no tempo t. O limite superior, MAX, corresponde ao valor máximo que pode ser lido pelos sensores. Convenciona-se que o valor mínimo de leitura é 0, sem perda de generalidade.

O conjunto de estados $\mathcal{S}$, por sua vez, é definido como discreto e finito, com $|\mathcal{S}| = n$ estados, da seguinte forma:

$$\mathcal{S} = \{x_1, x_2 \dots x_n\}. \tag{6.2}$$

Isso possibilita a implementação do algoritmo de aprendizagem da seção 5.3.3, além de permitir a interpretação de cada estado como uma regra em um sistema de regras (seção 5.4.1). Como será mostrado em detalhes, cada estado pode ser visto como um signo, cujo embasamento (*grounding*) é realizado através de protótipos definidos no conjunto de observação.

Finalmente, o conjunto $\mathcal{A}$ de ações é tomado como discreto e finito, com $|\mathcal{A}| = m$ ações não-ordenadas.

$$\mathcal{A} = \{u_1, u_2 \dots u_m\}. \tag{6.3}$$

Esta definição foi escolhida por simplicidade de implementação. Porém, apesar de simplificar a implementação, deve-se notar que esta suposição limita bastante a solução proposta, já que a maioria dos atuadores recebe comandos contínuos, como por exemplo sinais de potência ou velocidade. Uma implementação com ações contínuas é possível, mas será deixada para trabalhos futuros.

A partir da definição do conjunto de estados, tem-se a definição da função de transição de estados, $\mathcal{P}^a_{ss'}$. Pode-se representá-la como um conjunto de matrizes $A_w = \{a^w_{ij}\}_{n\times n}$ de transição de estados, uma matriz para cada ação u_w, $w \in \{1 \dots m\}$. Cada elemento a^w_{ij} da matriz A_w corresponde à probabilidade de transição para o estado x_j, dado o estado x_i e a ação u_w. Assim,

$$A_w \equiv \begin{pmatrix} a^w_{11} & a^w_{12} & \cdots & a^w_{1n} \\ a^w_{21} & a^w_{22} & \cdots & a^w_{2n} \\ & & \cdots & \\ a^w_{n1} & a^w_{n2} & \cdots & a^w_{nn} \end{pmatrix}, \tag{6.4}$$

onde

$$a^w_{ij} = \mathcal{P}^{u_w}_{x_i x_j} = \Pr\{s_{t+1} = x_j \mid s_t = x_i, a_t = u_w\}. \tag{6.5}$$

A representação das probabilidades $\mathcal{O}^a_s$ de observação não é tão simples. Como o conjunto Ω de observação é contínuo, tais probabilidades devem ser representadas na forma de funções de densidade de probabilidade (*pdf*), uma para cada estado $x_i \in \mathcal{S}$. Faz-se então,

$$f_i(o) \equiv \mathrm{p}(o_t = o \mid s_t = x_i). \tag{6.6}$$

Para poder aplicar algoritmos de aprendizagem às funções $f_i(o)$, é necessário parametrizá-las. A seguir, encontram-se as várias decisões de projeto que definem as famílias de função e os parâmetros de cada uma.

A primeira suposição está relacionada ao fato de que as observações são vetoriais, ou seja:

$$o_t = (o_t^1, o_t^2 \ldots o_t^d).$$

Deve-se estabelecer, então, qual a dependência probabilística entre cada elemento de um vetor de observação. Aqui, considera-se que cada um desses elementos é independente, ou seja, pode-se fatorar a densidade de probabilidade condicional de observação da seguinte maneira:

$$\mathrm{p}(o_t = o \mid s_t = x_i) = \prod_{k=1}^{d} \mathrm{p}(o_t^k = o^k \mid s_t = x_i). \tag{6.7}$$

Assim, definem-se as funções $f_i^k(o^k) \equiv \mathrm{p}(o_t^k = o^k \mid s_t = x_i)$, tal que:

$$f_i(o) = \prod_{k=1}^{d} f_i^k(o^k). \tag{6.8}$$

Note que em geral não é realista supor que cada um dos d sensores tem leituras independentes. Em sensores de distância próximos, as leitura são em geral fortemente correlacionadas. Ao não modelar tais correlações, a principal consequência para os algoritmos de estimação é a sobre-confiança (*overconfidence*) do robô (Thrun et al., 2006, seção 6.7), ou seja, a entropia da crença $b_t(s)$ decai fortemente, porque o robô acha que encontrou mais evidências de um estado do que caberia de fato. Porém, ainda assim é possível obter bons resultados, e a simplificação na implementação do algoritmo é grande o suficiente para compensar os problemas que surgiriam caso contrário.

Resta agora definir e parametrizar as funções $f_i^k(o^k)$, que modelam as probabilidades de cada dimensão k da observação, dado cada estado x_i. É aqui que entra a modelagem de estados como signos cujas observações são representadas por protótipos. A teoria de protótipos encontra fundamentos conceituais e empíricos nas ciências cognitivas (Gärdenfors, 2004). Além disso, protótipos são usados nos sistemas de regras do cap. 3, que serviram de ponto de partida para este projeto.

Protótipos são representantes de uma determinada classe escolhidos para compor a representação interna de tal classe. Nos sistemas de regras do cap. 3, cada regra constitui uma classe, e cada classe é representada por um protótipo de observação. Quando uma observação real chega ao sistema, é

possível compará-la com os protótipos de cada uma das classes.

Uma dificuldade importante que surge na comparação é a necessidade de uma medida de similaridade. Não há uma única forma clara e objetiva de como calcular a distância entre uma observação e um protótipo. Formalmente, este problema pode ser traduzido como o problema de encontrar a definição de uma função

$$d : \mathcal{O} \times \mathcal{O} \mapsto \mathbb{R}_+, \tag{6.9}$$

onde $d(o, o_t)$ é definida como a distância entre o protótipo o e a observação o_t.

Experiências realizadas com humanos na classificação de formas e cores de objetos mostram que em geral não há uma definição clara para $d(o, o_t)$. Em geral, não é possível nem mesmo considerar $d(o, o_t)$ como uma métrica: em alguns casos, as pessoas tendem até mesmo a desrespeitar a desigualdade triangular, ou seja, consideram objetos diferentes em apenas uma dimensão menos similares do que objetos diferentes em duas dimensões, como cor e comprimento, por exemplo. Também há evidências de que $d(o, o_t)$ não seria transitiva: em certos casos, as comparando a a b as indivíduos apresentam respostas diferentes que ao compararem b a a.

No entanto, apesar das dificuldades que aparecem quando se considera a teoria de protótipos aplicada à cognição humana, é possível sim desenvolver classificadores baseados em métrica. O capítulo 3 mostra que, nos sistemas de regras de Cazangi (2004) e na sua rede imuno-classificadora (Cazangi, 2008), por exemplo, foram obtidos bons resultados através o uso de uma métrica euclidiana ponderada, expressa pela eq. 3.1.

Neste projeto, a medida de distância, ou melhor, de similaridade, deve corresponder a uma medida de probabilidade condicional, como mostrado. Além disso, espera-se utilizar protótipos para representar as classes. Isto posto, a opção mais tradicional é utilizar uma distribuição normal multivariada. Dessa forma, a média μ da distribuição corresponderia ao protótipo, e a matriz de variância Σ definiria os coeficientes da métrica a utilizar. Anteriormente, assumiu-se a independência entre as diferentes dimensões de observação. Isso corresponde a limitar Σ a uma matriz diagonal.

Para se obter a distribuição normal multivariada para d dimensões independentes, pode-se partir de d distribuições normais de uma variável. Definem-se então, para cada estado x_i e cada dimensão k da variável de observação,

$$f_i^k(o^k) = \frac{1}{(2\pi)^{1/2}\sigma_i^k} \exp\left(\frac{-(o^k - \mu_i^k)^2}{2(\sigma_i^k)^2}\right), \tag{6.10}$$

onde os parâmetros da distribuição são μ_i^k e σ_i^k, respectivamente a média e o desvio padrão da k-ésima dimensão de observação do estado x_i. Substituindo-se a definição acima na eq. 6.8, obtém-se a probabilidade condicional considerando todas as dimensões de observação, que constitui uma normal

multivariada:

$$f_i(o) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp\Big(-\tfrac{1}{2}(o-\mu_i)'\Sigma_i^{-1}(o-\mu_i)\Big), \tag{6.11}$$

onde os parâmetros da distribuição são

$$\mu_i = (\mu_i^1, \mu_i^2 \ldots \mu_i^d) \text{ e}$$
$$\Sigma_i = \mathrm{diag}\Big((\sigma_i^1)^2, (\sigma_i^2)^2 \ldots (\sigma_i^d)^2\Big),$$

respectivamente o vetor média e a matriz de covariância do estado x_i.

Cabe observar que a quantidade no interior da exponencial, na distribuição multivariada, corresponde a uma métrica. Assim, definindo-se a distância entre protótipo e observação como

$$d(o, o_t) = \sqrt{-\tfrac{1}{2}(o-o_t)'\Sigma_i^{-1}(o-o_t)}, \tag{6.12}$$

que é conhecida como métrica de Mahalanobis, e que no caso de Σ diagonal aqui considerado, corresponde à métrica euclidiana normalizada:

$$d_i(o, o_t) = \sqrt{\sum_{k=1}^{d} \frac{(\mu_i^k - o_t^k)^2}{(\sigma_i^k)^2}}, \tag{6.13}$$

Assim, pode-se considerar que, utilizando uma distribuição normal multivariada, está-se, de certa forma, implementado uma extensão dos sistemas de regras do cap. 3

Há um detalhe, porém, que pode tornar o uso da distribuição normal inconveniente: em alguns ambientes, o sensor de distância frequentemente não encontra obstáculos, e neste caso o valor de distância máximo (MAX) é retornado. Pode ocorrer que alguns estados sejam formados unicamente por tais tipos de observação em certas dimensões. Por esse motivo, em vez de modelar a distribuição de cada dimensão como uma normal, como na eq. 6.10, adota-se um modelo de mistura formado por dois componentes: uma normal univariada e um delta de dirac sobre o valor máximo (MAX). Assim, tem-se:

$$f_i^k(o^k) = \eta_i^k \frac{1}{(2\pi)^{1/2}\sigma_i^k} \exp\left(\frac{-(o^k - \mu_i^k)^2}{2(\sigma_i^k)^2}\right) + (1-\eta_i^k)\delta(o_t^k - \mathrm{MAX}), \tag{6.14}$$

onde, além dos parâmetros μ_i^k e σ_i^k da normal, surge o parâmetro η_i^k, o coeficiente de mistura, que determina, para cada estado x_i e para cada dimensão k de observação, qual a probabilidade de uma leitura do sensor retornar um valor menor que MAX. A Fig. 6.1 mostra um exemplo de distribuição resultante. Para obter a distribuição conjunta para o vetor de observação, considerando-se todas as

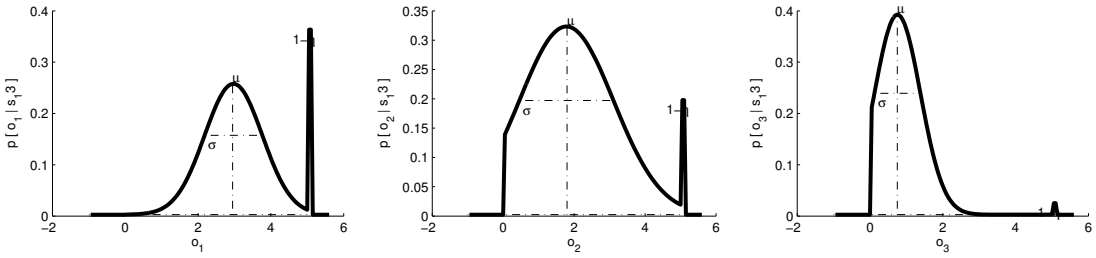


Fig. 6.1: Exemplo de probabilidades condicionais de observação $f_i^k(o^k)$ para um modelo com $d = 3$ dimensões, para um dado estado, pelo modelo de mistura definido na eq. 6.14.

dimensões de observação, aplica-se a eq. 6.8 sobre a eq. 6.14, e obtém-se:

$$f_i(o) = \prod_{k=1}^{d} \left\{ \eta_i^k \frac{1}{(2\pi)^{1/2}\sigma_i^k} \exp\left(\frac{-(o^k - \mu_i^k)^2}{2(\sigma_i^k)^2} \right) + (1 - \eta_i^k)\delta(o_t^k - \mathrm{MAX}) \right\}. \tag{6.15}$$

Deve-se atentar para o detalhe que tanto a distribuição normal da eq. 6.10 quanto a mistura definida acima (eq. 6.14) atribuem densidades de probabilidade positivas para valores fora do intervalo $[0, \mathrm{MAX}]$, o que não condiz com a especificação do sensor. Porém, supõe-se que esse fato não afetará muito fortemente a qualidade dos resultados, já que o que importa realmente é a comparação entre os valores de probabilidade obtidos para uma mesma observação, dados diferentes estados.

Em resumo, ao longo da seção foi especificado de forma completa o POMDP a ser utilizado no projeto. O conjunto de parâmetros λ a serem aprendidos é, então:

$$\lambda = \left(a_{ij}^w, \mu_i^k, \sigma_i^k, \eta_i^k \right), \quad w \in \{1 \dots m\}, \quad i, j \in \{1 \dots n\}, \quad k \in \{1 \dots d\} \tag{6.16}$$

6.2.1 Imprecisão dos sensores

Até o momento, não foi mencionada a imprecisão das leituras sensoriais. Porém, sabe-se que nenhum sensor é preciso a ponto de retornar sempre a mesma leitura, mesmo que não haja modificação no ambiente. Nesta seção, procura-se mostrar que um POMDP, parametrizado como nas seções anteriores, modela implicitamente tal incerteza.

A imprecisão na leitura sensorial decorre de dois fatores: primeiramente, fatores intrínsecos relacionados à tecnologia e construção do sensor. Por exemplo, a precisão de um sonar depende, entre outros fatores, da sensibilidade do detector de ondas sonoras. Assim, mesmo em um ambiente completamente isento de interferência sonora, o sensor dificilmente retornará o mesmo valor para duas medidas consecutivas. Em segundo lugar, há a imprecisão relacionada ao ambiente: no caso do sonar, quinas na parede podem gerar uma dupla reflexão, certas superfícies absorvem ou dispersam as ondas sonoras, e obstáculos móveis efêmeros podem passar diante do sensor.

Suponha-se agora que a localização dos obstáculos detectáveis por dado sensor de distância seja conhecida. O conjunto de tais obstáculos forma um mapa do ambiente, aqui denotado por m. Além disso, suponha-se que a pose (posição e orientação) do robô no instante t seja conhecida, e denotada por $\mathbf{x}_t$. Dados $\mathbf{x}_t$ e m, a distância exata entre o sensor e o obstáculo mais próximo é conhecida. Sob essas condições, a incerteza quanto à leitura o_t do sensor é totalmente devida à imprecisão do sensor, que pode ser modelada como:

$$\mathrm{p}(o_t^i \mid \mathbf{x}_t, m). \tag{6.17}$$

Uma possível parametrização para esta distribuição de probabilidades é apresentada em Thrun et al. (2006, seção 6.3). Lá, $\mathrm{p}(o_t^i \mid \mathbf{x}_t, m)$ é modelada como mistura de uma distribuição exponencial, uma normal, um delta de Dirac e uma uniforme. Note-se a semelhança com o modelo de observação definido na seção anterior, que constitui na mistura de uma normal e um delta de Dirac.

Argumenta-se a seguir que a parametrização realizada na seção precedente para o modelo de observação $\mathrm{p}(o_t^i \mid s, \lambda)$ incorpora, além da noção de protótipos explicada anteriormente, alguns aspectos da imprecisão do sensor. Para isso, considera-se conhecido, além dos parâmetros λ do POMDP, o mapa m do ambiente. Dessa forma, a probabilidade condicional de observação pode ser expressa como $\mathrm{p}(o_t^i \mid s, \lambda, m)$, e então fatorada da seguinte maneira:

$$\begin{aligned}
\mathrm{p}(o_t^i \mid s_t, \lambda, m) &= \int \mathrm{p}(o_t^i, \mathbf{x} \mid s_t, \lambda, m)\, d\mathbf{x} \\
&= \int \mathrm{p}(o_t^i \mid \mathbf{x}, s_t, \lambda, m)\, \mathrm{p}(\mathbf{x} \mid s_t, \lambda, m)\, d\mathbf{x} \\
&= \int \mathrm{p}(o_t^i \mid \mathbf{x}, m)\, \mathrm{p}(\mathbf{x} \mid s_t, \lambda, m)\, d\mathbf{x}.
\end{aligned}$$

Da fatoração acima, conclui-se que as probabilidades condicionais de observação, num POMDP parametrizado como nas seções anteriores, têm dupla função: por um lado, modelam as incertezas do sensor, $\mathrm{p}(o_t^i \mid \mathbf{x}, m)$. Por outro lado, efetuam também uma prototipação do espaço de poses do robô. Cada estado do POMDP é um protótipo que representa um conjunto de poses $\mathbf{x}$. Tais conjuntos podem ser vistos como conjuntos *fuzzy*, já que suas fronteiras não são bem delimitadas. Assim, pode-se considerar que a cada estado $x_i \in \mathcal{S}$ corresponde um conjunto *fuzzy* de poses, e que o grau de pertinência de cada pose $\mathbf{x}$ ao conjunto x_i é dado pela densidade de probabilidade $\mathrm{p}(\mathbf{x} \mid s = x_i, \lambda, m)$.

Apesar de o conhecimento do mapa permitir a realização da fatoração acima, o objetivo do presente trabalho é justamente buscar controladores que não possuam tal informação de forma explícita. Busca-se representar o modelo do sistema pelos parâmetros λ do POMDP, sem necessidade do conhecimento do mapa m. Assim, nesse trabalho, confia-se ao modelo de observação $\mathrm{p}(o_t^i \mid s, \lambda)$ a dupla função de prototipação e modelagem das imprecisões do sensores.

6.2.2 Equações de reestimação de parâmetros do POMDP

Como explicado na seção 5.3.3, o objetivo do passo de maximização do algoritmo Baum-Welch é a reestimação de parâmetros. Para isso, devem-se encontrar os parâmetros que levem à máxima verossimilhança de observação, considerando-se conhecida a probabilidade *a posteriori* sobre os estados do sistema em cada instante de tempo. Tais estados são estimados no passo de estimação do algoritmo. Abaixo, derivam-se as fórmulas de reestimação.

O problema de reestimação dos parâmetros é definido na eq. 5.14, e a expressão a maximizar no caso do algoritmo Baum-Welch é dada pela eq. 5.21. Agora, já tendo sido escolhidas as parametrizações para o POMDP, é possível substituí-las nestas equações para obter a reestimação ótima a cada iteração do algoritmo EM.

Reestimaçao dos parâmetros de transição

Inicia-se tratando do problema de reestimação das probabilidades $a_{ij}^{w} = \mathcal{P}_{x_i,x'_j}^{u_w}$ de transição de estados. Utilizam-se a seguir, apenas os termos dependentes de $\mathcal{P}_{ss'}^{a}$ na eq. 5.21. Os outros não influenciam na reestimação de tais parâmetros. Além disso, é necessário considerar a restrição de soma unitária, para que os parâmetros constituam distribuições de probabilidade. O problema a resolver é, então:

$$\begin{aligned} \underset{\mathcal{P}_{x_i,x_j}^{u_w}}{\text{maximizar}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{t=1}^{T-1} \xi_t^{(\tau)}(x_i,x_j)\log \mathcal{P}_{x_i,x'_j}^{a_t} \\ \text{s.a.} \quad & \sum_{j=1}^{n} \mathcal{P}_{x_i,x_j}^{u_w} - 1 = 0, \qquad \forall i \in \{1,\dots,n\}, \forall w \in \{1,\dots,m\} \end{aligned} \tag{6.18}$$

O lagrangiano do problema é,

$$\mathcal{L}\{\mathcal{P}_{x_i,x_j}^{u_w}, c_{iw}\} = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{t=1}^{T-1} \xi_t^{(\tau)}(x_i,x_j)\log \mathcal{P}_{x_i,x'_j}^{a_t} + \sum_{w=1}^{m}\sum_{i=1}^{n} c_{iw}\left(\sum_{j=1}^{n}\mathcal{P}_{x_i,x_j}^{u_w} - 1\right), \tag{6.19}$$

onde c_{iw} são os multiplicadores de Lagrange relacionados a cada uma das restrições do problema. Derivando-se o lagrangiano em relação a cada um dos parâmetros e dos multiplicadores e igualando-se a zero, é possível encontrar a expressão para a reestimação dos parâmetros de transição na τ-ésima iteração do algoritmo EM:

$$a_{ij}^{w\ (\tau+1)} = \mathcal{P}_{x_i,x_j}^{u_w\ (\tau+1)} = \frac{\sum_{t=1}^{T-1}\mathbb{I}\{a_t = u_w\}\xi_t^{(\tau)}(x_i,x_j)}{\sum_{t=1}^{T-1}\mathbb{I}\{a_t = u_w\}\gamma_t^{(\tau)}(x_i)}, \tag{6.20}$$

onde $\mathbb{I}$ é a função indicadora, definida como:

$$\mathbb{I}\{a_t = u_w\} \equiv \begin{cases} 1, & \text{se } a_t = u_w \\ 0, & \text{se } a_t \neq u_w \end{cases} \tag{6.21}$$

A eq. 6.20 de reestimação pode ser compreendida de forma intuitiva se os valores $\gamma_t(s)$ forem interpretados como um grau de confidência na veracidade da seguinte asserção: "o estado do sistema no tempo t era s". Se a certeza é absoluta, então tal expressão vale 1 caso a asserção seja verdadeira ou 0 caso seja falsa. Da mesma forma, os valores $\xi_t(s, s')$ podem ser interpretados como o grau de confidência de que "no tempo t ocorreu uma transição do estado s para o s'". Pode-se ainda interpretar $\gamma_t(s)$ e $\xi_t(s, s')$ como funções indicadoras $\mathbb{I}$ que assumem valores intermediários entre 0 e 1 quando há incerteza.

Assim, a fórmula de reestimação da eq. 6.20 pode ser vista de forma intuitiva como uma contagem relativa de quantas transições ocorreram do estado x_i ao estado x_j, na qual em vez de cada transição ser contada como 1 em caso de ocorrência ou 0 em caso de não-ocorrência, é contada como um valor intermediário entre 0 e 1, dependendo da confidência de que tal transição realmente tenha ocorrido.

Reestimaçao dos parâmetros de observação

O problema de reestimação dos parâmetros de observação consiste em reestimar os valores μ_i^k, σ_i^k e η_i^k, que definem as probabilidades condicionais de observação da forma como foram definidas na eq. 6.14.

Tais valores são aqueles que maximizam a expressão da eq. 5.21, com as probabilidades de observação $\mathcal{O}_s^o$ substituídas por sua parametrização, dada pela eq. 6.14. Considerando-se apenas os termos que dependem de $\mathcal{O}_s^o$, o problema a resolver fica:

$$\begin{aligned} \underset{\mu_i^k, \sigma_i^k, \eta_i^k}{\text{maximizar}} \quad & \sum_{i=1}^{n} \sum_{t=1}^{T} \gamma_t^{(\tau)}(s) \log \prod_{k=1}^{d} \Big\{ \\ & \mathbb{I}\{o_k^t < \text{MAX}\} \eta_i^k \frac{1}{(2\pi)^{1/2} \sigma_i^k} \exp\left(\frac{-(o_t^k - \mu_i^k)^2}{2(\sigma_i^k)^2} \right) + \\ & \mathbb{I}\{o_k^t = \text{MAX}\} \log(1 - \eta_i^k) \Big\}. \end{aligned} \tag{6.22}$$

Na equação acima, $\mathbb{I}\{o_t^k < \text{MAX}\} = 1$ se a observação o_t^k tiver sido originada a partir da distribuição normal, e $\mathbb{I}\{o_t^k = \text{MAX}\} = 1$ se o_t^k tiver sido originada a partir do delta de Dirac, ou seja, quando o sensor retornou o valor máximo. Nunca os dois valores são diferentes de zero ao mesmo tempo. Isso

permite passar o $\log$ para o interior do produtório, e o problema fica:

$$\begin{aligned}\underset{\mu_i^k,\sigma_i^k,\eta_i^k}{\text{maximizar}} \quad & \sum_{i=1}^{n}\sum_{t=1}^{T}\gamma_t^{(\tau)}(s)\sum_{k=1}^{d}\Big\{ \\ & \mathbb{I}\{o_k^t < \text{MAX}\}\left[\log\eta_i^k - \frac{1}{2}\log(2\pi) - \log\sigma_i^k - \frac{(o_t^k-\mu_i^k)^2}{2(\sigma_i^k)^2}\right] + \\ & \mathbb{I}\{o_k^t = \text{MAX}\}(1-\eta_i^k)\Big\}.\end{aligned} \tag{6.23}$$

Derivando-se em relação a η_i^k e igualando a 0, obtém-se a expressão para reestimação dos coeficientes de mistura na τ-ésima iteração do EM:

$$\eta_i^{k\ (\tau+1)} = \frac{\displaystyle\sum_{t=1}^{T}\mathbb{I}\{o_k^t < \text{MAX}\}\gamma_t^{(\tau)}(x_i)}{\displaystyle\sum_{t=1}^{T}\gamma_t^{(\tau)}(x_i)}. \tag{6.24}$$

Novamente, interpretando-se $\gamma_t(s)$ como o grau de pertinência do sistema ao estado s no tempo t, a equação acima pode ser considerada como a razão entre "o número de vezes em que o sistema esteve no estado x_i e o k-ésimo sensor não apresentou leitura $\mathbf{MAX}$" e "o número total de vezes em que o sistema esteve no estado x_i".

Derivando-se em relação a μ_i^k e obtém-se a reestimação das médias para a distribuição normal:

$$\mu_i^{k\ (\tau+1)} = \frac{\displaystyle\sum_{t=1}^{T}\mathbb{I}\{o_k^t < \text{MAX}\}\gamma_t^{(\tau)}(x_i)o_k^t}{\displaystyle\sum_{t=1}^{T}\mathbb{I}\{o_k^t < \text{MAX}\}\gamma_t^{(\tau)}(x_i)}. \tag{6.25}$$

Tal fórmula corresponde a uma média de todas as observações o_t^k ao longo do tempo, ponderadas pela pertinência $\gamma_t(x_i)$ do sistema ao estado x_i no tempo t, e desconsiderando observações em que o sensor registrou leitura $\mathbf{MAX}$.

Finalmente, os desvios-padrão são reestimados da seguinte forma:

$$\sigma_i^{k\ (\tau+1)} = \frac{\displaystyle\sum_{t=1}^{T}\mathbb{I}\{o_k^t < \text{MAX}\}\gamma_t^{(\tau)}(x_i)(o_k^t-\mu_i^k)^2}{\displaystyle\sum_{t=1}^{T}\mathbb{I}\{o_k^t < \text{MAX}\}\gamma_t^{(\tau)}(x_i)}. \tag{6.26}$$

Assim, foram obtidas fórmulas para a reestimação de todos os parâmetros do POMDP. Os parâmetros a_{ij}^w de transição são reestimados pela eq. 6.20, e os parâmetros de observação η_i^k, μ_i^k e σ_i^k são reestimados pelas eqs. 6.24, 6.25 e 6.26, respectivamente.

Finalmente, o algoritmo Baum-Welch completo utilizado neste trabalho é listado no Algoritmo. 1.

entrada: Sequência de observações e ações, $(\mathbf{O}, \mathbf{A})$
Número de estados n
Parâmetros iniciais $\lambda^{(1)}$
Número de iterações τ_f
saída : Parâmetros estimados $\lambda^{(\tau_f+1)} = (a_{ij}^w, \eta_i^k, \mu_i^k, \sigma_i^k)$

for $\tau \leftarrow 1$ **to** τ_f **do**
`// E-step: cálculo de` $\gamma_t^{(\tau)}(x_i)$ `e` $\xi_t^{(\tau)}(x_i)$ `a partir de` $\lambda^{(\tau)}$
`calcula` $f_i^{k\,(\tau)}(o_t^k)$ (eq. 6.14) ;
`calcula` $\alpha_t^{(\tau)}(s)$ (processo *forward*, seção 5.3.3) ;
`calcula` $\hat{\beta}_t^{(\tau)}(s)$ (processo *backward*, seção 5.3.3) ;
`calcula` $\gamma_t^{(\tau)}(s)$ (eq. 5.35) ;
`calcula` $\xi_t^{(\tau)}(s, s')$ (eq. 5.36) ;
`// M-step: cálculo de` $\lambda^{(\tau+1)}$ `a partir de` $\gamma_t^{(\tau)}(x_i)$ `e` $\xi_t^{(\tau)}(x_i)$
`calcula` $a_{ij}^{w\,(\tau+1)}$ (eq. 6.20) ;
`calcula` $\eta_i^{k\,(\tau+1)}$ (eq. 6.24) ;
`calcula` $\mu_i^{k\,(\tau+1)}$ (eq. 6.25) ;
`calcula` $\sigma_i^{k\,(\tau+1)}$ (eq. 6.26) ;
end

Algoritmo 1: Algoritmo Baum-Welch utilizado neste trabalho.

6.3 Aprendizagem de recompensas e função-valor no POMDP

De todos os parâmetros do POMDP $(\mathcal{S}, \mathcal{A}_s, \mathcal{P}_{ss'}^a, \mathcal{R}_{ss'}^a, \mathcal{O}_s^o, \Omega)$, o algoritmo *offline* apresentado nas seções anteriores é capaz, a princípio, de treinar apenas os parâmetros $\lambda = (\mathcal{P}_{ss'}^a, \mathcal{O}_s^o)$. Além disso, os conjuntos $\mathcal{S}$, $\mathcal{A}_s$ e Ω são pré-definidos. Resta então um método para a aprendizagem das recompensas imediatas, $\mathcal{R}_{ss'}^a$. Alternativamente, é possível aprender diretamente a função-valor $V(s)$, o que resolveria simultaneamente o problema de planejamento.

As recompensas imediatas são recebidas pelo agente a cada instante de tempo. Assim, ao longo de um episódio de duração T, o agente recebe uma sequência $\mathbf{R} = (r_1, r_2, \ldots, r_T)$. Em vista dos algoritmos de aprendizagem por reforço apresentados no capítulo anterior e do método de aprendizagem *offline* de POMDP, é possível delinear diferentes formas de aprendizagem dos parâmetros.

6.3.1 Aprendizagem *offline* da recompensa imediata

É possível efetuar o aprendizado das recompensas imediatas de forma *offline*. Para isso, basta considerar a recompensa r_t como se fosse uma leitura de sensor. O vetor de observação o_t fica, então:

$$o_t = (o_t^1, o_t^2, \ldots, o_t^d, r_t). \tag{6.27}$$

Isso feito, o treinamento pelo algoritmo Baum-Welch pode ser realizado. O resultado será que as recompensas imediatas serão modeladas por uma mistura de distribuição normal e Delta de Dirac, como especificado na seção precedente. É possível ignorar o delta se o valor MAX para a dimensão correspondente for escolhido de forma a nunca ser atingido. Assim, a recompensa será modelada como uma distribuição normal para cada estado x_i do sistema. Isso não é totalmente satisfatório considerando-se que na definição de MDPs e POMDPs as recompensas imediatas dependem não somente do estado s_t mas também de s_{t+1} e da ação a_t tomada. Seria então necessário modificar a fórmula de verossimilhança do POMDP para modelar tais parâmetros, o que é factível.

Outra opção seria simplesmente considerar que as recompensas recebidas dependem apenas do estado atual, ou seja, modelar as recompensas como $\mathcal{R}_s$. É possível representar qualquer POMDP original como um POMDP nessa nova formulação, mas o número de estados necessários seria maior, pois cada estado deve então ser quebrado em diferentes estados, um para cada valor da recompensa, segundo o estado seguinte e a ação tomada.

6.3.2 Aprendizagem *offline* da função-valor

Alternativamente à aprendizagem da recompensa imediata, é possível aprender *offline* diretamente a função-valor aproximada do QMDP (seção 5.2.4), exatamente como no método Monte Carlo apresentado na seção 4.2. Ao fim de um episódio de T intervalos de tempo, e conhecendo-se a sequência de recompensas imediatas, é possível calcular o retorno parcial a partir de cada instante de tempo, utilizando-se a eq. 4.2. Obtém-se então a sequência de retornos parciais, $(R_1, R_2, \ldots, R_T)$, que podem ser incluídos no vetor de observação. Assim,

$$o_t = (o_t^1, o_t^2, \ldots, o_t^d, R_t). \tag{6.28}$$

Como para o caso das recompensas imediatas, basta executar o algoritmo Baum-Welch como definido na seção anterior, e o resultado será uma distribuição normal de retornos para cada estado x_i do sistema. A função-valor $V(s)$ ou $V(x_i)$ corresponde à média da distribuição normal para cada estado, ou seja:

$$V(x_i) = \mu_i^{d+1}, \tag{6.29}$$

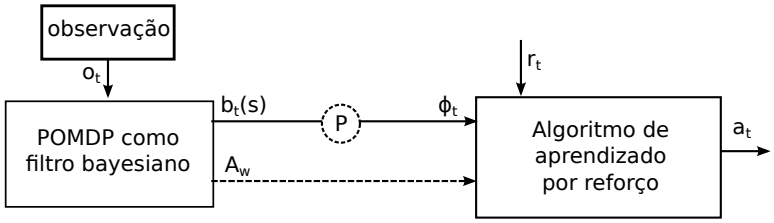


Fig. 6.2: Acoplamento proposto neste trabalho entre um POMDP e um algoritmo de aprendizagem por reforço. "P" representa um processamento opcional entre a saída $b_t(s)$ do POMDP e a entrada ϕ_t do algoritmo de aprendizado.

onde $d + 1$ é a dimensão da observação que corresponde ao retorno esperado, e μ é a média da distribuição condicional de observação, como definida na eq. 6.10.

A principal conveniência desse método é que não são necessárias modificações na definição do POMDP — diferentemente do método de aprendizagem de recompensas imediatas, abordado anteriormente —, nem no algoritmo de aprendizagem. Além disso, a resolução do POMDP, ou seja, o planejamento, é efetuada através do método de Monte Carlo, dispensando o uso posterior da programação dinâmica.

6.3.3 Aprendizagem *online*: acoplamento com algoritmos de aprendizado por reforço

Uma possível deficiência das abordagens apresentadas anteriormente é justamente o fato de serem *offline*. Como mostrado na seção 4.2, o principal trunfo da aprendizagem por reforço é o fato de ela se efetuar *online*, ou seja, ao longo da vida do agente. Nas abordagens a seguir, a aprendizagem por reforço é utilizada juntamente com POMDPs para se obter a aprendizagem *online*.

Na abordagem *online* proposta neste trabalho, faz-se um acoplamento de um POMDP, parametrizado como nas seções anteriores, com um algoritmo de aprendizagem por reforço, seja ele Sarsa, Q-learning ou gradiente de política, propostos na seção 4.2. O acoplamento se realiza como na Fig. 6.2. O POMDP é utilizado como um filtro Bayesiano, ou seja, recebe o vetor de observações o_t na entrada e, com base no modelo de mundo aprendido anteriormente, atualiza recursivamente o vetor de crenças $b_t(s)$. Este vetor de crenças é então passado ao algoritmo de aprendizado por reforço.

Como mostra a figura, pode haver algum pré-processamento, denotado por $\mathbf{P}$, entre a crença $b_t(s)$ e o vetor de entrada ϕ_t do algoritmo de aprendizado. Ao longo da seção, será apresentada uma justificativa para se fazer $b_t = \phi_t$, ou seja, não efetuar nenhum processamento. Na seção experimental, porém, serão testadas também outras alternativas para a entrada ϕ_t.

Note que o vetor de entrada ϕ_t do algoritmo por reforço corresponde ao estado s_t, conforme apresentado no capítulo 4. Essa troca de notação se faz porque o conjunto de estados $\mathcal{S}_{MDP}$ do

algoritmo de aprendizagem por reforço não corresponde necessariamente ao conjunto de estados do POMDP, principalmente quando há um processamento $\mathbf{P}$ entre as duas unidades. Em outras palavras, enquanto o POMDP tenta manter um conjunto de estados $\mathcal{S}_{POMDP}$ do ambiente, o algoritmo de aprendizagem mantém um conjunto de estados do POMDP, ou seja, o conjunto de possíveis crenças.

É possível que o algoritmo de aprendizagem por reforço utilize também as matrizes A_w de probabilidades de transição do POMDP como modelo da dinâmica do sistema, como mostra a conexão tracejada da Fig. 6.2. Nesse caso, a princípio não é necessário utilizar a função-valor de ação, $Q(s, a)$, como faz o algoritmo *Q-learning* da seção 4.2.1. A função $V(s)$ já é suficiente para determinar a política ótima do robô, como faz o algoritmo *Sarsa*. Porém, em testes preliminares, tal estratégia não apresentou bons resultados. Assim, optou-se por utilizar algoritmos livres de modelo, que estimam eles mesmos, implicitamente, a dinâmica do sistema.

Especificação das parametrizações

Pretende-se que a entrada ϕ_t do algoritmo de aprendizado assuma valores contínuos. Por essa razão, é necessário que o algoritmo de aprendizagem utilize parametrização. No caso do algoritmo *Sarsa*, deve-se parametrizar a função valor $V(s)$. No caso do *Q-learning*, a função-valor de ação $Q(s, a)$. Finalmente, para o algoritmo de gradiente de política apresentado na seção 4.2.4, a política $\pi_\theta(a|s)$ e a função valor $V(s)$ devem ser parametrizadas.

Propõe-se inicialmente a parametrização para $V(s)$ e $Q(s, a)$. Utiliza-se, por motivos que ficarão claros posteriormente nessa seção, uma parametrização linear:

$$\tilde{V}_\theta(\phi_t) = \phi_t \cdot \theta \qquad \tilde{Q}_\theta(\phi_t, u_i) = \phi_t \cdot \theta_i \tag{6.30}$$

onde $\phi_t \in \mathbb{R}^d$ é a entrada do algoritmo de aprendizado no tempo t, como retratado na Fig. 6.2. No caso da função $\tilde{V}_\theta$, o parâmetro θ é um vetor de d elementos. No caso da função $\tilde{Q}_\theta$, θ corresponde a uma matriz $d \times m$, onde m é o número de possíveis ações, da seguinte forma:

$$\theta = \left[\; \theta_1 \; \middle| \; \theta_2 \; \middle| \; \dots \; \middle| \; \theta_m \; \right].$$

O gradiente das parametrizações lineares apresentadas acima em relação aos parâmetros θ é facilmente calculado. Para a função $\tilde{V}_\theta(\phi)$, o gradiente equivale, simplesmente, ao vetor de entrada:

$$\nabla_\theta \tilde{V}_\theta(\phi_t) = \phi_t, \tag{6.31}$$

e, para a função-valor da ação, o gradiente corresponde a uma matriz $d \times m$, ou seja, de mesmo

tamanho que a matriz θ de parâmetros:

$$\nabla_\theta \tilde{Q}_\theta(\phi_t, a_t) = \left[\begin{array}{c|c|c|c|c|c} 0 & 0 & \dots & \phi_t & \dots & 0 \end{array} \right], \tag{6.32}$$

na qual a única coluna diferente de zero é a coluna j tal que $a_t = u_j$, ou seja, a coluna correspondente à ação a_t passada como argumento.

Apresenta-se a seguir uma justificativa para o uso das parametrizações lineares propostas, e além disso, o porquê de utilizar o vetor de crença $\mathbf{b}_t$ diretamente como entrada do algoritmo. Mostra-se que tal escolha de parâmetros resulta na aproximação QMDP, apresentada ao final da seção 5.2.4. Como já mencionado, é possível interpretar um POMDP como um MDP cujo conjunto de estados $\mathcal{S}_{MDP}$ é definido como:

$$\mathcal{S}_{MDP} \equiv \Delta(\mathcal{S}_{POMDP}), \tag{6.33}$$

onde $\Delta(\mathcal{S}_{POMDP})$ é o conjunto de crenças do POMDP original. Ou seja, um MDP onde cada estado corresponde a uma crença do POMDP original. Assim sendo, poderia-se aprender a função-valor $V(b)$ para cada possível crença $b \in \mathcal{S}_{MDP}$ através do método da diferença temporal. Infelizmente, essa abordagem leva à maldição da dimensionalidade.

Para se lidar com esse inconveniente, assume-se que a incerteza sobre os estados será resolvida no tempo $t+1$, obtendo-se a aproximação QMDP. Retomando-se a eq. 5.10, que caracteriza o QMDP:

$$V(b) \approx \sum_{s \in \mathcal{S}} b_t(s) V(s).$$

Considerando-se que o conjunto de estados é finito e definido como $\mathcal{S} = (x_1, x_2, \dots, x_n)$, tem-se,

$$\begin{aligned} V(b) &\approx \sum_{x=1}^{n} b_t(x_i) V(x_i) \\ &= \Big(b_t(x_1), b_t(x_2), \dots, b_t(x_n)\Big) \cdot \Big(V(x_1), V(x_2), \dots, V(x_n)\Big), \end{aligned}$$

onde $\cdot$ representa o produto interno de dois vetores. Definindo-se o vetor de crença $\mathbf{b}_t \in \mathbb{R}^n$ no tempo t como

$$\mathbf{b}_t \equiv \Big(b_t(x_1), b_t(x_2), \dots, b_t(x_n)\Big), \tag{6.34}$$

pode-se parametrizar $V(b)$ definindo-se os parâmetros $\theta \in \mathbb{R}^n$ como:

$$\theta \equiv \Big(V(x_1), V(x_2), \dots, V(x_n)\Big). \tag{6.35}$$

Assim, utilizando-se $\phi_t = \mathbf{b}_t$ na parametrização linear da eq. 6.30, obtém-se exatamente a apro-

ximação utilizada no algoritmo QMDP.

Nessa subseção, foram definidas parametrizações para as funções $V(\phi_t)$ e $Q(\phi_t, a_t)$, utilizadas respectivamente pelo Sarsa e *Q-learning*. É necessário ainda especificar a parametrização para o algoritmo de gradiente de política. Isso será feito posteriormente, na seção 6.4. A seguir, derivam-se as fórmulas de atualização para os parâmetros θ.

Especificação do TD(λ) parametrizado

De posse das parametrizações definidas acima, é possível chegar à fórmula de diferença temporal para a atualização dos parâmetros de $\tilde{V}_{\theta_t}(\phi_t)$, que consiste no algoritmo Sarsa. Substituindo-se na eq. 4.16, obtém-se uma fórmula para a atualização dos parâmetros θ no caso TD(0) (sem traço de eligibilidade):

$$\theta_{t+1} \leftarrow \theta_t + \alpha\delta_t\phi_t, \tag{6.36}$$

onde α é a taxa de aprendizagem e δ_t é a diferença temporal, calculada pela eq. 4.7, e o vetor ϕ_t é a entrada do algoritmo de aprendizado.

Para o caso TD(λ), o traço de eligibilidade τ é definido como um vetor no mesmo espaço vetorial dos parâmetros θ, ou seja, $\tau \in \mathbb{R}^d$. A fórmula de atualização para τ é obtida substituindo-se o gradiente da eq. 6.31 acima na eq. 4.17:

$$\tau_{t+1} = \lambda\gamma\tau_t + \phi_t, \tag{6.37}$$

onde γ é o desconto na definição de retorno (eq. 4.2). Finalmente, de posse de τ_{t+1}, atualizam-se os parâmetros θ através da eq. 4.18, aqui reproduzida:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\delta_t\tau_{t+1}.$$

A dedução das fórmulas de atualização de $\tilde{Q}_\theta(\phi_t)$, usadas no algoritmo *Q-learning*, são semelhantes, e por isso são omitidas. Como a parametrização é linear, a convergência de $\tilde{V}_\theta(\phi_t)$ ou $\tilde{Q}_\theta(\phi_t)$ para valores aproximados da função-valor é garantida, se a política for mantida fixa e se a taxa de aprendizagem α for decrescente ao longo do tempo e respeitar certas restrições (Bertsekas, 2007). Apesar disso, cabe dizer que há interesse em se atualizar a política a cada iteração, já que essa possibilidade é a principal vantagem da estratégia *online* sobre a *offline*. Infelizmente, nesse caso pode ocorrer divergência.

Finalmente, o uso de $\lambda = 1$, ou seja, TD(1), é semelhante à abordagem *offline* apresentada anteriormente. A única diferença é que, em vez de se utilizar a crença $\mathbf{b}_t$ como ϕ_t na eq. 6.36, naquele caso

utiliza-se a estimação de estados γ_t. Por ser calculada *offline*, γ_t leva em conta observações futuras e passadas, e consequentemente é a uma estimação com incerteza menor.

6.4 Especificação do algoritmo de gradiente de política

Procede-se aqui à especificação do algoritmo de gradiente de política, cujas linhas gerais foram apresentadas na seção 4.2.4. Para determinar completamente o algoritmo a ser usado neste trabalho, há três escolhas a serem feitas: a mais importante consiste na especificação da função parametrizadora $\pi_\theta(a_t|\phi_t)$ para a política. É necessário também escolher uma parametrização $\tilde{V}_v(\phi_t)$ para a função-valor e, finalmente, uma forma de estimar os parâmetros v da função-valor e w da função-vantagem.

Parametrização da função-valor

Utiliza-se, para a função-valor, a parametrização linear já proposta anteriormente. Porém, para evitar confusão quanto à notação, reserva-se agora a letra θ para os parâmetros da política. Os parâmetros da função-valor são expressos então pelo vetor $v \in \mathbf{R}^d$, onde d é a dimensão do vetor de entrada ϕ_t do algoritmo. Além disso, para evitar confusão com outro vetor de características apresentado posteriormente, chama-se a entrada ϕ_t do algoritmo de ϕ_t^{in}. Assim,

$$\tilde{V}_v(\phi_t^{in}) = \phi_t^{in} \cdot v. \tag{6.38}$$

Parametrização da política

Quanto à parametrização da política, utiliza-se um classificador logístico multinomial (Moore et al., 2009, cap. 14), que, a partir da entrada $\phi_t^{in} \in \mathbb{R}^d$, apresenta na saída as probabilidades $\Pr\{a_t = u_i \mid \phi_t^{in}\}$, para cada uma das possíveis ações $u_i \in \mathcal{A}$ de um conjunto finito de m ações, $\mathcal{A} = \{u_1, u_2, \ldots, u_m\}$. O modelo do classificador é:

$$\begin{aligned}
\pi_\theta(u_1|\phi_t^{in}) &\equiv \Pr\{a_t = u_1 \mid \phi_t^{in}, \theta\} = \frac{1}{1 + \sum_{j=2}^{m} \exp(\phi_t^{in} \cdot \theta_j)}, \\
\pi_\theta(u_i|\phi_t^{in}) &\equiv \Pr\{a_t = u_i \mid \phi_t^{in}, \theta\} = \frac{\exp(\phi_t^{in} \cdot \theta_i)}{1 + \sum_{j=2}^{m} \exp(\phi_t^{in} \cdot \theta_j)}, \quad \text{para } 2 \leq i \leq m
\end{aligned} \tag{6.39}$$

onde $\theta_i \in \mathbb{R}^d,\ i \in \{2, \ldots, m\}$ são os parâmetros do modelo. Para representar todos os parâmetros θ_i em uma única estrutura, define-se o parâmetro θ como uma matriz $d \times (m-1)$, na qual cada uma

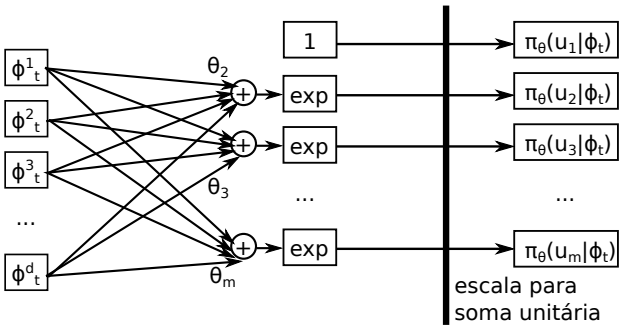


Fig. 6.3: Classificador logístico multinomial visto como uma rede neural com saída escalada.

das colunas representa um dos vetores θ_i:

$$\theta \equiv \left[\; \theta_2 \; \middle| \; \theta_3 \; \middle| \; \ldots \; \middle| \; \theta_m \; \right]. \tag{6.40}$$

Como mostra a Fig. 6.3, o classificador logístico pode ser visto como uma rede neural cujas entradas correspondem aos elementos do vetor ϕ_t^{in} e cujas saídas correspondem cada uma à probabilidade de executar uma das possíveis ações u_i. Diferentemente de uma rede neural usual, porém, o somatório das saídas deve ser igual a 1, para representar uma distribuição de probabilidades.

Em seguida, para se obter a parametrização compatível $f_w(\phi_t^{in}, a_t)$, expressa na eq. 4.26, deve-se obter o gradiente do logaritmo da política em relação aos parâmetros θ. Este pode ser calculando analiticamente a partir da eq. 6.39. O resultado é uma matriz $d \times (m-1)$, definida pelo seguinte produto matricial:

$$\nabla_\theta \ln \pi_\theta(u_i|\phi_t^{in}) = \begin{bmatrix} \phi_t^{in\ 1} \\ \phi_t^{in\ 2} \\ \cdots \\ \phi_t^{in\ d} \end{bmatrix} \begin{bmatrix} -\pi_\theta(u_2|\phi_t) \\ -\pi_\theta(u_3|\phi_t) \\ \cdots \\ 1 - \pi_\theta(u_i|\phi_t) \\ \cdots \\ -\pi_\theta(u_m|\phi_t) \end{bmatrix}^{\mathrm{T}}, \tag{6.41}$$

onde o termo 1 da segunda matriz aparece somente no termo correspondente à ação u_i para a qual o gradiente está sendo calculado. No caso $a = u_1$, o 1 não aparece em nenhum dos elementos da matriz.

Para poder se aplicar $\nabla_\theta \ln \pi_\theta(u_i|\phi_t^{in})$ na eq. 4.26 e obter $f_w(\phi_t^{in}, a_t)$, define-se, primeiramente,

$$\phi_t^{pol} = \mathbf{vec}\Big\{ \nabla_\theta \pi_\theta(u_i|\phi_t) \Big\} \tag{6.42}$$

ou seja, $\phi_t^{pol} \in \mathbb{R}^{d \times (m-1)}$ corresponde ao gradiente da política em forma de vetor. Considerando o

vetor de parâmetros como $w \in \mathbf{R}^{d\times(m-1)}$, redefine-se então $f_w(\phi_t^{in}, a_t)$ como:

$$f_w(\phi_t^{in}, a_t) = \phi_t^{pol} \cdot w. \tag{6.43}$$

Uma vez definidas as parametrizações $\tilde{V}_v(\phi_t^{in})$ da função-valor e $f_w(\phi_t^{in}, a_t)$ da função vantagem, a função-valor de ação $\tilde{Q}(\phi_t^{in}, a)$, pode ser expressa, a partir da eq. 4.34, como:

$$\begin{aligned} Q_{v,w}(\phi_t^{in}, a_t) &= \tilde{V}_v(\phi_t^{in}) + f_w(\phi_t^{in}, a_t) \\ &= \begin{bmatrix} \phi_t^{in} \\ \phi_t^{pol} \end{bmatrix} \cdot \begin{bmatrix} v \\ w \end{bmatrix}, \end{aligned} \tag{6.44}$$

onde $\cdot$ é o produto interno entre os dois vetores-coluna formados, cada um deles, pela concatenação de dois vetores-coluna.

Estimação dos parâmetros

Pela definição de função-valor da ação (eq. 4.4), tem-se que, fixada a política, $Q^\pi(s_t, a_t) = \mathrm{E}\{R_t \mid s_t, a_t\}$. Assim, parece razoável estimar os parâmetros v e w da função $\tilde{Q}_{(v,w)}(\phi_t^{in}, a_t)$, definida na eq. 6.44 acima, através do método dos quadrados mínimos efetuados sobre os retornos R_t das últimas τ iterações. Faz-se, então:

$$(v, w) = \arg\min_{v,w} \sum_{i=1}^{\tau} \left[R_{t-k} - Q_{v,w}(\phi_{t-k}^{in}, a_{t-k})\right]^2. \tag{6.45}$$

Além disso, como $\tilde{Q}_{(v,w)}(\phi_t^{in}, a_t)$ é linear nos parâmetros (v, w), pode-se usar uma regressão linear. Com base nessas constatações, deriva-se a seguir o algoritmo de gradiente de política usado neste trabalho.

Especificação do algoritmo

Diferentemente dos algoritmos Sarsa e *Q-learning*, que são implementados de forma inteiramente *online* na forma das equações de diferença temporal apresentadas anteriormente, o algoritmo de gradiente de política aqui proposto pode ser considerado como semi-online. A fim de realizar a regressão linear para encontrar os parâmetros v e w, o algoritmo acumula os vetores de entrada ϕ_t^{in}, assim como os vetores ϕ_t^{pol}, ao longo de uma sequência de τ iterações. Além disso, o algoritmo acumula a sequência de recompensas imediatas r_t. Ao fim de τ iterações, o algoritmo possui uma matriz $\mathbf{M}_{\tau\times dm}$ e um

vetor-coluna $\mathbf{r}_{1\times\tau}$, com dados acumulados nas últimas τ iterações, da seguinte forma:

$$\mathbf{M} = \begin{bmatrix} \phi_{t-\tau}^{in} & \phi_{t-\tau+1}^{in} & \cdots & \phi_{t-1}^{in} \\ \phi_{t-\tau}^{pol} & \phi_{t-\tau+1}^{pol} & \cdots & \phi_{t-1}^{pol} \end{bmatrix}^{\mathrm{T}}, \quad \mathbf{r} = \begin{bmatrix} r_{t-\tau} \\ r_{t-\tau+1} \\ \cdots \\ r_{t-1} \end{bmatrix}. \tag{6.46}$$

A partir do vetor coluna $\mathbf{r}$, o algoritmo calcula um outro vetor coluna $\mathbf{R}$, de mesmo número de linhas, que corresponde aos retornos aproximados das últimas τ iterações. Asim, a partir da definição de retorno da eq. 4.2,

$$\mathbf{R} = \begin{bmatrix} R_{t-\tau} \\ R_{t-\tau+1} \\ \cdots \\ R_{t-1} \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{\tau-1} \gamma^k r_{k+t-\tau} \\ \sum_{k=0}^{\tau-2} \gamma^k r_{k+t-\tau+1} \\ \cdots \\ \sum_{k=0}^{0} \gamma^k r_{k+t-1} \end{bmatrix}. \tag{6.47}$$

De posse da matriz de características $\mathbf{M}$ e do vetor de coluna de retornos $\mathbf{R}$, o algoritmo calcula os parâmetros w da função-vantagem e v da função-valor através de uma regressão linear:

$$\begin{bmatrix} v \\ w \end{bmatrix} = \mathbf{M}^{\dagger}\mathbf{R}, \tag{6.48}$$

onde $\mathbf{M}^{\dagger} = (\mathbf{M}^{\mathrm{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathrm{T}}$ é a pseudo-inversa da matriz $\mathbf{M}$, calculada utilizando-se o processo de decomposição em valores singulares (SVD).

Finalmente, os parâmetros w encontrados são utilizados para alterar os parâmetros θ da política, através do gradiente natural, conforme a eq. 4.32. Assim:

$$\theta_{k+1} = \theta_k + \alpha w. \tag{6.49}$$

O algoritmo 2, listado a seguir, representa as operações efetuadas por esse método em uma iteração do ciclo de controle. Em resumo, o algoritmo proposto é um algoritmo de gradiente natural de política *semi-online*, pois envolve acúmulo de observações e de retornos para se chegar à estimação do gradiente natural w. O algoritmo possui dois parâmetros numéricos:

- α, a taxa de aprendizagem. Quanto maior α, mais rápido o robô aprende, porém mais rápido decai a exploração. Assim, com um α grande demais, ruídos de observação e estocasticidade do ambiente podem impedir que ocorra um aprendizado satisfatório;

- τ, o intervalo de aprendizagem. Controla o quão *offline* é o algoritmo, já que a melhoria de política acontecerá somente em tempos t múltiplos de τ. Quanto maior τ, mais lenta a apren-

dizagem, porém maior a confiabilidade da estimação de w. É interessante que τ seja escolhido de forma a permitir que o ambiente seja suficientemente explorado pelo robô entre cada cálculo do gradiente.

entrada: Vetor de características de entrada: $\phi_t^{in} = \mathbf{b}_t$
Última ação executada: a_t
Recompensa recebida: r_t
Iteração atual: t

if $\mathrm{mod}\ (t, \tau) == 0$ **then**
  $\mathbf{R} \leftarrow$ `calcula_retorno(r)` ; `/* eq. 6.47 */`
  $\begin{bmatrix} v \\ w \end{bmatrix} \leftarrow \mathbf{M}^{\dagger}\mathbf{R}$; `/* eq. 6.48 */`
  $\theta \leftarrow \theta + \alpha w$; `/* eq. 6.49 */`
  $\mathbf{M} \leftarrow \emptyset$;
  $\mathbf{r} \leftarrow \emptyset$;
else
  $\phi_t^{pol} \leftarrow \nabla_\theta \ln \pi_\theta(a_t|\phi_t^{in})$; `/* eq. 6.41 */`
  $\mathbf{M} \leftarrow \mathbf{M} \cup [\phi_t^{in} \ \ \phi_t^{pol}]$;
  $\mathbf{r} \leftarrow \mathbf{r} \cup [r_t]$;
end

Algoritmo 2: Iteração do algoritmo proposto de gradiente natural de política.

6.5 Ciclo de controle do robô

Com todos os algoritmos de aprendizagem em mãos, resta estabelecer quais atualizações serão efetuadas de forma *online* e *offline*, e, para as atualizações *offline*, com que frequência e com quais dados serão executadas. Além disso, é necessário determinar qual algoritmo de aprendizado por reforço (Sarsa, *Q-learning* ou gradiente de política) será acoplado ao POMDP. Isto feito, pode-se apresentar o algoritmo completo de controle e aprendizagem do agente, a ser utilizado nos experimentos do capítulo seguinte.

O ciclo de controle *online* é dado pela Fig. 6.4, que baseia-se na Fig. 5.8. Cada passo é explicado a seguir:

1. **cálculo das probabilidades condicionais de observação**: corresponde ao passo de processamento sensorial. A eq. 6.14 é usada para calcular a probabilidade de cada uma das d leituras sensoriais para cada um dos estados $x_1, x_2, \ldots, x_n$. Em seguida, para cada estado x_i, a eq. 6.15 é utilizada para fundir as d leituras sensoriais em uma única probabilidade de observação por

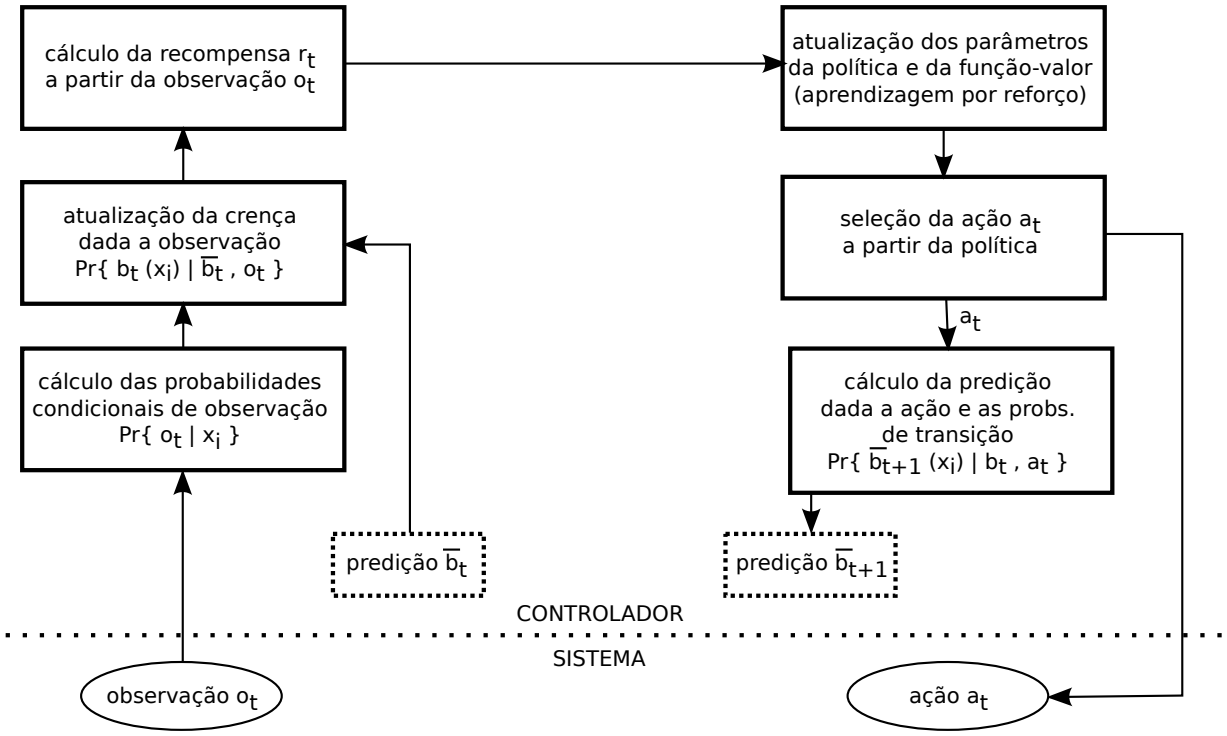


Fig. 6.4: Ciclo de controle *online*

estado, $\mathrm{p}(o_t \mid s),\ \ \forall s \in (x_1, x_2, \ldots, x_n)$. Note que, nesse passo, além das leituras sensoriais o_i^k recebidas dos sensores, é necessário acesso aos parâmetros de observação do POMDP, no caso μ_i^i, σ_i^k e η_i^k para cada estado x_i e para cada dimensão de observação k;

2. **atualização da crença dada a observação**: esse passo corresponde ao passo de correção do filtro de Bayes, expresso pelas eqs. 5.6 e 5.9, no qual a predição $\bar{b}_t(x_i)$ é atualizada utilizando-se os valores de probabilidade de observação $\mathrm{p}(o_t \mid s)$ calculados no passo anterior, obtendo-se assim a crença $b_t(x_i),\ \ \forall i \in \{1, 2, \ldots, n\}$;

3. **cálculo da recompensa r_t a partir da observação**: esse passo é implementado conforme a definição de recompensa para o agente em questão. A discussão de como se concebe a função de recompensa a partir do objetivo de alto nível do agente é feita na seção. 7.1.3;

4. **atualização dos parâmetros da política e da função-valor**: este papel é desempenhado pelo algoritmo de aprendizagem por reforço, seja ele Sarsa, *Qlearning* ou gradiente de política;

5. **seleção da ação a_t a partir da política**: tendo-se em mãos a política $\pi(a, x_i),\ \ \forall x_i$, e a crença b_t, ou seja, a probabilidade de se estar em cada estado x_i, é possível calcular a probabilidade de execução de cada ação, de maneira colaborativa ou competitiva. Estes dois métodos foram apresentados na seção 5.4.1;

6. **cálculo da predição $\bar{b}_{t+1}$ dada a ação**: uma vez que a ação a tomar a_t já tenha sido escolhida, é possível utilizar as probabilidades $\mathcal{P}_{ss'}^{a_t}$ e a crença atual b_t para calcular a predição $\bar{b}_{t+1}$. Esse passo corresponde ao passo de predição do filtro de Bayes, que segue a eq. 5.8.

O ciclo de controle *online* tendo sido especificado pela Fig. 6.4, e o algoritmo *offline*, que corresponde ao Baum-Welch tendo sido especificado no algoritmo 1, resta estabelecer a coordenação entre esses dois aspectos do controlador. O algoritmo 3 consiste numa visão geral do controlador.

Cabem algumas explicações quanto ao funcionamento de tal algoritmo. Como mostrado, o controle funciona alternando um ciclo *online* com um ciclo *offline*, por um período indeterminado. No algoritmo, cada iteração do laço é iniciada pelo ciclo *online*, executado em sincronia com o robô. Em seguida, há lugar o ciclo *offline*, executado assincronamente, que utiliza as observações recolhidas durante o ciclo *online* para estimar os parâmetros do POMDP.

Em geral, o ciclo *online* utiliza os parâmetros do POMDP estimados no ciclo *offline* imediatamente anterior. A exceção se dá na primeira iteração, quando nenhum ciclo *offline* foi executado anteriormente. Nesse caso, é executado um ciclo *online* alternativo, que não utiliza o POMDP. Em vez disso, a observação é apresentada diretamente à entrada do algoritmo de aprendizagem.

No próximo capítulo, o algoritmo 3 será utilizado para o aprendizado e o controle de um robô móvel autônomo em ambientes simulados.

entrada: Parâmetros iniciais do POMDP: $\lambda = (\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$, n, m
Função valor inicial: $V_1(b)$
Número de iterações do Baum-Welch: τ_f
Número de iterações online: T

while *True* **do**
  $\bar{b}_1 \leftarrow \frac{1}{n}(1, 1, \ldots, 1)$;
  $\mathbf{O} \leftarrow \emptyset$;
  $\mathbf{A} \leftarrow \emptyset$;
  for $t \leftarrow 1$ **to** T **do**
    $o_t \leftarrow$ `lê_sensores()` ;
    $(\bar{b}_{t+1}, a_t) \leftarrow$ `ciclo_online(`λ, $\bar{b}_t$, o_t`)` ; `/* Fig. 6.4 */`
    `executa_ação(`a_t`)` ;
    $\mathbf{O} \leftarrow \mathbf{O} \cup (o_t)$;
    $\mathbf{A} \leftarrow \mathbf{A} \cup (a_t)$;
  end
  $\lambda \leftarrow$ `baum_welch(` $\mathbf{O}$, $\mathbf{A}$, n, λ, τ_f`)` ; `/* Ciclo offline */`
end

Algoritmo 3: Visão geral do controlador, composto das fases *online* e *offline*.

6.6 Resumo

Neste capítulo, são propostas parametrizações para os algoritmos de aprendizado por reforço do cap. 4 e os POMDPs do cap. 5. Tais parametrizações são fortemente inspiradas pelas propostas de sistema de navegação autônomo (Cazangi, 2004) e de controladores baseados em redes imuno-inspiradas (Cazangi, 2008), ambas apresentadas no cap. 3. Dessa forma, pode-se dizer que este capítulo agrega as teorias expostas nos três capítulos anteriores, propondo assim um modelo para o problema de navegação em robótica autônoma.

- Inicialmente, é proposta uma parametrização para o POMDP. Em específico, propõe-se uma parametrização das distribuições condicionais de obsevação baseada em protótipos. Esta proposta é inspirada no sistema de navegação autônomo de Cazangi (2004), que propunha uma medida de similaridade entre um protótipo – o antecedente de uma regra – e a leitura sensorial. Com a parametrização do POMDP em mãos, o algoritmo Baum-Welch é especificado por completo.

- Na seção seguinte, o capítulo trata do aprendizado das recompensas no POMDP, que não é resolvido pelo algoritmo Baum-Welch. São propostas três formas de realizar tal aprendizado: as primeiras duas são efetuada de forma *offline*. Na primeira, aprendem-se as recompensas imediatas e, na segunda, aprende-se o retorno, ou seja, os valores de cada estado. Finalmente, a terceira proposta utiliza o método de diferença temporal parametrizado por um QMDP.

- Em seguida, como alternativa ao aprendizado de recompensas no POMDP, apresenta-se a especificação de um algoritmo de gradiente de política *semi-offline* cuja política parametrizada consiste em um classificador logístico multinomial. Mostra-se que tal política pode ter sua entrada acoplada à crença do POMDP.

- Finalmente, propõe-se o ciclo completo de controle do robô. Propõe-se um ciclo baseado em dois sub-ciclos: um ciclo *online*, no qual se dá o controle do robô em tempo real e são armazenados dados de leitura sensorial, e um ciclo *offline*, que utiliza os dados armazenados para treinar o POMDP.

Capítulo 7

Experimentos

No capítulo atual, os métodos propostos ao longo dos capítulos anteriores são aplicados na aprendizagem da política de um robô móvel autônomo navegando em um ambiente simulado. O propósito é avaliar a capacidade dos algoritmos propostos de gerar, no menor intervalo de tempo possível, controladores que atendam os objetivos da navegação. Primeiramente são escolhidos o agente, o ambiente e a função de recompensa. Em seguida, são discutidos os critérios de avaliação de performance. Finalmente, os resultados são discutidos e usados para guiar trabalhos futuros.

7.1 Problema Proposto

O problema a ser resolvido neste capítulo pode ser enquadrado no *framework* da concepção de agentes completos, descrito em Pfeifer & Scheier (1999). Segundo o autor, é impossível conceber um experimento de agentes autônomos sem que haja uma escolha interdependente de três fatores: a) as características físicas e lógicas do agente, b) o ambiente e c) a tarefa a ser executada. Isso significa que, mesmo que o objetivo seja apenas testar uma arquitetura de controlador, como é o caso aqui, não é possível negligenciar, ou mesmo isolar, os outros aspectos envolvidos no experimento.

Da mesma forma, é impossível avaliar isoladamente a atuação do controlador: um agente somente desempenha sua tarefa com sucesso se, além de possuir um controlador satisfatório, suas características físicas o permitirem, para um dado ambiente. Isso posto, é necessário justificar com cuidado a escolha de cada um dos aspectos envolvidos na concepção e avaliação de um projeto de agente completo.

Antes de proceder às escolhas específicas, é interessante estabelecer alguns princípios norteadores. Neste trabalho, tais princípios são os seguintes: a) as escolhas devem ser tomadas de forma a permitir o uso dos algoritmos de aprendizagem propostos e a comparação com outros algoritmos de aprendizagem por reforço contidos na literatura; b) os experimentos, à medida do possível, devem

Fig. 7.1: Robô Pioneer 3-DX, com sonares e sensores de colisão nos para-choques. *Fonte: http://www.conscious-robots.com/*

se assemelhar com outros já realizados na literatura; c) apesar de serem realizados em ambientes simulados, os experimentos devem ser adaptáveis da forma mais transparente possível a ambientes reais.

7.1.1 Escolha do agente: hardware

A escolha do agente envolve dois aspectos: o *hardware* e o *software*. Ambos influenciam na competência do robô para realizar uma dada tarefa. Nesta seção, serão apresentadas as escolhas de *hardware* e, na seção seguinte, as escolhas de *software*, ou seja, a arquitetura de controle.

Quanto à escolha do *hardware*, Pfeifer & Scheier (1999) dá exemplos de como uma escolha criteriosa da forma física do robô, das características e do posicionamento de sensores e atuadores pode facilitar muito a implementação de um agente autônomo, dependendo da tarefa e do ambiente. Por exemplo, rodas funcionam muito bem em ambientes com solo regular, mas patas, apesar de menos eficientes, são mais apropriadas para uma maior gama de terrenos. Da mesma forma, a visão pode ser muito adequada para ambientes complexos, mas pode apresentar falhas na detecção de alguns obstáculos que nunca se manifestariam em um sensor de distância, por exemplo.

Como entre os princípios norteadores da escolha do experimento encontra-se a fácil adaptabilidade a uma possível futura implementação real, é conveniente restringir a escolha da configuração física do agente a modelos de robôs reais. O robô tendo sido escolhido, é possível posteriormente escolher um ambiente e uma tarefa condizente com as capacidades do robô.

Neste projeto, escolhe-se como agente o robô-base *Pioneer 3-DX*, como mostra a Fig. 7.1. Tal robô possui um conjunto de 16 sonares dispostos em torno de sua carcaça, dos quais 4 serão utilizados nos experimentos. Tais sonares têm um alcance de 5 metros. Quando nenhum obstáculo é encontrado, o valor máximo “5 m” é indicado. Além disso, o robô possui sensores nos para-choques, capazes de detectar eventos de colisão com obstáculos. O robô possui dois atuadores, que atuam cada um sobre

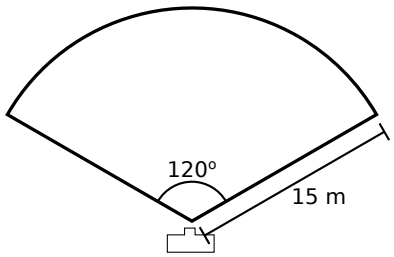


Fig. 7.2: Abertura angular da câmera utilizada nas simulações.

uma das duas rodas, gerando um par diferencial. As velocidades de cada uma das rodas determinam, em conjunto, as velocidades angular e linear do robô, como modelado pelas eqs. 2.4.

Além dos sensores embutidos no *P3DX*, é comum que seja acoplada, acima da sua base, uma câmera de vídeo. Dessa forma, através de algoritmos simples de processamento de imagem, é possível encontrar, na imagem capturada pela câmera, regiões contíguas de cores semelhantes. São os chamados *blobs* (borrões, em inglês), cujo tamanho, posição e cor são facilmente calculáveis. Considera-se, nesse trabalho, que o robô possui uma câmera com um detector de *blobs*, o *blobfinder*. Escolheu-se para tal câmera uma abertura angular de 120° e uma resolução de 160 pixels horizontais e 120 pixels verticais. O *blobfinder* detectará obstáculos que estiverem a até $15m$ da câmera, como mostra a Fig. 7.2.

A interface entre o robô e o seu controlador é feita através do *software open source Player*, que proporciona um protocolo comum de comunicação entre robôs e controladores. Dessa forma, é possível controlar qualquer robô que implemente a interface *Player*, seja ele real ou simulado, sem que haja necessidade de reprogramar o controlador.

Quanto ao *software* de simulação, optou-se pelo *Stage* (Vaughan, 2008), pelo fato de ser *open source*, simples, conter modelos prontos para os robôs *P3-DX*, assim como um suporte amplo à arquitetura *Player*. O *Stage* é considerado um simulador *2.5D* pois, apesar de a física do robô ser simulada em 2D, a visualização se dá em 3D, o que permite uma simulação mais fiel da câmera e do *blobfinder*.

A Fig. 7.3 mostra uma visualização gerada pelo simulador *Stage*, com um modelo do robô P3-DX em vermelho. Os quatro raios azuis partindo do robô representam a zona de detecção de obstáculos de cada um dos 4 sonares. À direita, a figura ilustra os *blobs* encontrados pela câmera de vídeo, posicionada acima do robô.

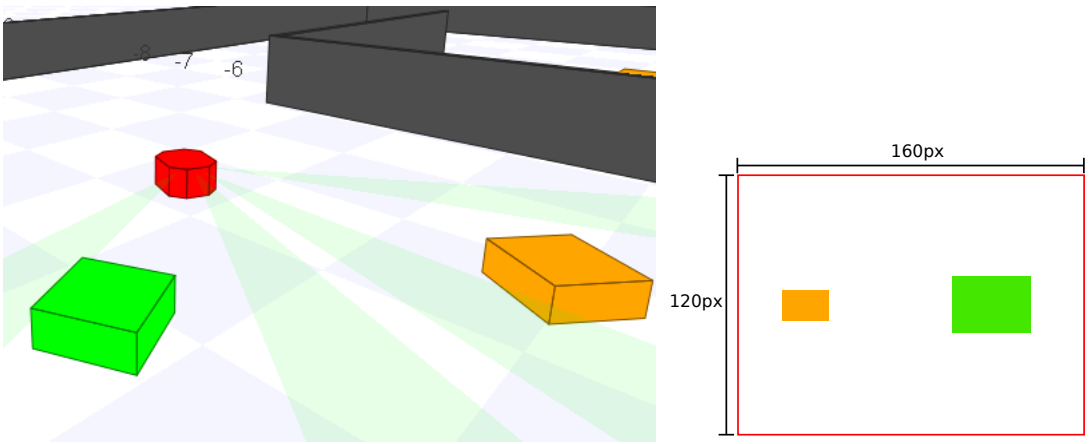


Fig. 7.3: Visualização gerada pelo simulador *Stage*, com modelo do robô P3-DX (em vermelho) e seus sensores. À direita, imagem gerada pelo *blobfinder* a partir da câmera instalada sobre o robô.

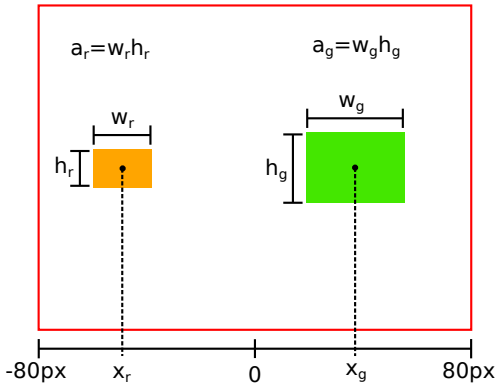


Fig. 7.4: Informação fornecida aos algoritmos a respeito dos *blobs*.

7.1.2 Escolha do agente: arquitetura de controle

Uma vez definidas as características físicas do agente e as interfaces de comunicação, deve-se definir o *software*, ou seja, o controlador do agente. Nesse ponto, pretende-se que seja possível utilizar os diferentes algoritmos de aprendizado por reforço apresentados ao longo do trabalho. Para isso, é necessário adaptar a interface do *Player* às entradas e saídas de tais algoritmos.

Definição do vetor de observação o_t

As entradas dos algoritmos apresentados correspondem às observações o_t, representadas por vetores de números reais de tamanho fixo, ou seja, $o_t \in \mathbb{R}^d,\ \forall t$. O *Player* representa a leitura dos sonares como um vetor real de tamanho fixo. Portanto, tais leituras já estão adaptadas aos algoritmos.

A detecção de *blobs*, porém, é mais complexa. O *Player* apresenta os *blobs* como uma lista de tamanho variável, com um elemento para cada *blob* presente na imagem da câmera a cada instante

de tempo. Para passar tal informação aos algoritmos de aprendizado, é preciso convertê-la em um vetor de tamanho fixo. Escolheu-se, então, passar apenas informação sobre no máximo 2 blobs a cada instante de tempo. A partir da lista, toma-se o blob de maior área para cada uma de duas cores pré-definidas (no caso, verde e laranja). As áreas a_g e a_r e a posição horizontal x_g e x_r de cada um desses dois *blobs*, como definidas na Fig. 7.4, são passadas então como parte do vetor de observação dos algoritmos de aprendizado.

Em resumo, o vetor de observação o_t apresentado como entrada aos algoritmos de aprendizado é composto da seguinte forma:

$$o_t = \Big(d_t^1,\ d_t^2,\ d_t^3,\ d_t^4,\ a_t^g,\ a_t^r,\ x_t^g,\ x_t^r\Big), \tag{7.1}$$

onde $d_t^1, \ldots, d_t^4$ são as leituras dos quatro sensores de distância (sonares) do robô no tempo t (em metros); a_t^g e x_t^g são, respectivamente, a área (em quantidade de pixels) e a coordenada horizontal do centro de massa (em pixels) do maior *blob* verde detectado no tempo t pela câmera; e a_t^r e x_t^r são a área e coordenada horizontal do centro de massa do maior *blob* laranja.

Os valores de distância ao obstáculo, d_t^i, são reais que excursionam de 0 a $5m$. Quanto ao *blobfinder*, a câmera utilizada possui resolução de 160×120 pixels. Como mostra a Fig. 7.4, os valores de x_t^g e x_t^r são inteiros excursionando entre -80 px e 80 px, com 0 para o centro da câmera ou quando nenhum blob da cor correspondente foi detectado. Além disso, a_t^g e a_t^r são inteiros que excursionam de 0 (nenhum blob daquela cor encontrado) a $160 \times 120 = 19200\ \mathrm{px}^2$ (para um blob ocupando toda a imagem da câmera).

Este pré-processamento das leituras sensoriais pode ser considerado como um mecanismo rudimentar de controle de atenção, pois, a partir de uma lista de tamanho variável de *blobs*, seleciona apenas aqueles que, a princípio, poderão ser mais úteis ao controle do robô. Mecanismos mais sofisticados de atenção deveriam levar em conta a utilidade de cada um dos dados fornecidos pelos sensores, ou seja, deveriam direcionar a atenção para a informação que leve potencialmente a recompensas maiores.

Definição das ações a_t

Para o comando do robô, comumente enviam-se comandos `setVel()` e `setVelAng()` à interface *Player* para alterar as velocidades linear e angular, respectivamente. Um controlador de baixo nível no interior do robô geralmente é responsável por calcular a cinemática reversa, ou seja, converter tais velocidades nas velocidades desejadas de cada uma das rodas, e, por sua vez, converter tais velocidades em valores de potência nos atuadores. Pode-se dizer, então, que o *Player* recebe, a cada

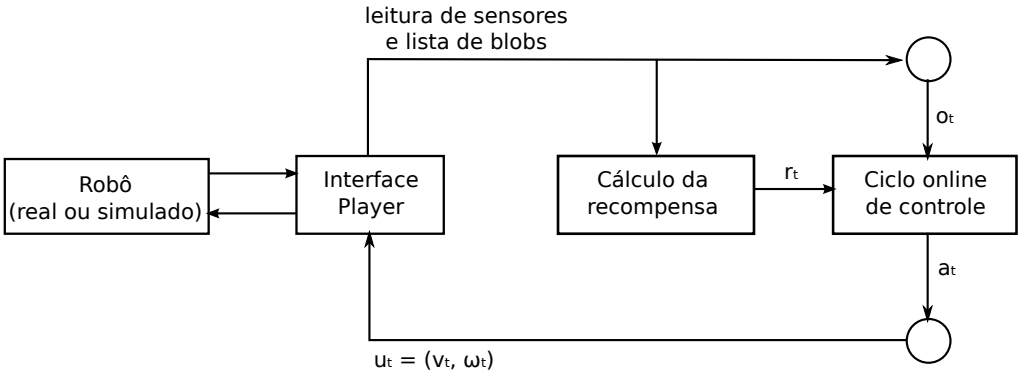


Fig. 7.5: Ciclo completo de controle, incluindo a interface com o *Player*.

intervalo de tempo t, um sinal de controle u_t, como segue:

$$u_t = \Big(v_t,\ \omega_t\Big), \tag{7.2}$$

onde v_t é a velocidade alvo (em metros por segundo) e ω_t é a velocidade angular alvo (em radianos por segundo), ambas no tempo t. Note que tais velocidades alvo não serão adotadas imediatamente pelo robô, devido a limitações físicas como inércia, potência máxima dos motores, necessidade de manutenção do atrito estático entre as rodas e o solo.

Ocorre, porém, que os algoritmos de aprendizado apresentados ao longo dos capítulos anteriores apresentam em sua saída uma ação a_t que assume valores discretos. Portanto, é necessário, de alguma forma, traduzir tais valores discretos em valores de u_t a serem fornecidos à interface *Player*. Ao longo dos experimentos, a menos quando apontado o contrário, utiliza-se o seguinte mapeamento:

a_t	u_t	**Descrição**
1	$(1,\ 0)$	Siga em frente a 1,0 m/s
2	$(0,\ -0.8)$	Faça curva à direita a 0,8 rad/s
3	$(0,\ +0.8)$	Faça curva à esquerda a 0,8 rad/s

O ciclo de controle completo do robô, incluindo a interface com o *Player*, é mostrado na Fig. 7.5. A cada ciclo de controle, representado por um instante de tempo t, a interface *Player* fornece um conjunto de dados advindo dos sonares e do *blobfinder*, que são convertidos num vetor de observações o_t, composto como na eq. 7.1. Tal vetor é apresentado então ao algoritmo de aprendizado, juntamente com um valor de recompensa r_t cujo cálculo será detalhado posteriormente. Em resposta, o algoritmo fornece uma ação a_t. A seguir, o mapeamento da tabela acima é usado para obter o sinal de controle u_t correspondente à ação a_t. O sinal de controle é então enviado ao *Player*.

É importante frisar que os algoritmos de aprendizado utilizados em nenhum momento têm acesso a esse mapeamento das ações 1, 2 e 3, ou seja, eles devem aprender por si sós o efeito de cada

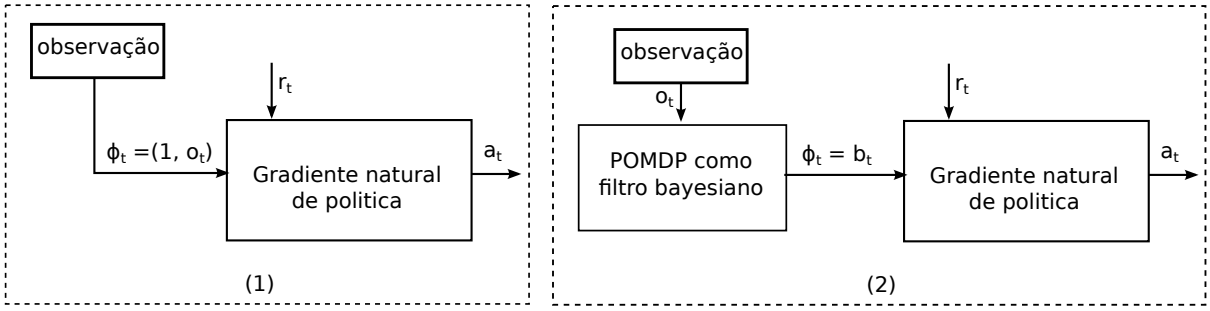


Fig. 7.6: As duas diferentes abordagens utilizadas para a parametrização das funções-valor e das políticas, nos algoritmos de aprendizado empregados.

uma das ações, seja em termos de recompensa resultante, seja em termos de transição de estados ou observações. Isso significa que, a princípio, é possível alterar o mapeamento acima sem que seja necessário alterar os algoritmos utilizados.

Escolha da entrada ϕ_t do algoritmo de aprendizado por reforço

Finalmente, é necessário especificar quais algoritmos de aprendizado serão utilizados e quais parametrizações serão escolhidas para as funções-valor e para a política, quando necessário. Em experimentos preliminares efetuados com três algoritmos distintos (algoritmo Monte Carlo offline, Q-learning, algoritmo baseado em gradiente natural de política), os algoritmos Monte Carlo offline e Q-learning não funcionaram. Resolveu-se então utilizar somente o algoritmo de gradiente natural de política, cuja apresentação inicial encontra-se na seção 4.2.4, e cujas escolhas específicas de parametrização e estimação são detalhadas na seção 6.4.

Como apresentado ao início da seção 6.3.3, na Fig. 6.2, a entrada ϕ_t do algoritmo de aprendizado por reforço não é necessariamente o vetor de crença $\mathbf{b}_t$ do controlador. Nos experimentos a serem efetuados, são utilizadas duas abordagens diferentes para ϕ_t, ilustradas na Fig. 7.6:

1. o vetor ϕ_t corresponde à entrada sensorial o_t, acrescida de um termo constante de polarização:

$$\phi_t \equiv \Big(1,\ o_t\Big); \tag{7.3}$$

2. o vetor ϕ_t corresponde ao vetor de crença $\mathbf{b}_t$, obtendo-se a aproximação QMDP:

$$\phi_t \equiv \mathbf{b}_t. \tag{7.4}$$

Assim, a abordagem (1) dará origem a controladores exclusivamente reativos. A abordagem (2), à qual foi dada ênfase na seção 6.3.3, permitirá o uso de observações passadas, advinda da propagação

da crença $b_t(s)$, como explicado anteriormente. Tais abordagens apresentam um compromisso entre simplicidade e flexibilidade. A abordagem (1) é mais simples, porém não tolera não-linearidades e comportamentos não reativos. Por essas razões, as duas abordagens serão testadas e será traçado um paralelo entre elas.

7.1.3 Tarefa

Numa arquitetura de aprendizado por reforço, o objetivo de um agente, ou seja, sua tarefa última, é "maximizar o retorno". Nesse trabalho, o retorno é definido pela eq. 4.2, em função das recompensas imediatas r_t recebidas a cada instante de tempo. Portanto, no contexto de aprendizagem por reforço, projetar um agente com determinada tarefa corresponde a projetar uma função-recompensa r_t, que pode depender de qualquer informação acessível ao agente no instante t. A Fig. 7.5 situa o módulo de cálculo da recompensa no ciclo de controle geral do robô.

As tarefas do robô foram definidas buscando-se um equilíbrio com a complexidade de *hardware* e *software* do robô, especificadas ao longo da seção anterior. Os objetivos do robô, para todos os experimentos executados, são, então, em ordem de prioridade:

- Evitar colisões com obstáculos;
- Afastar-se de objetos laranja (predadores);
- Aproximar-se de objetos verdes (presas);
- Andar o máximo possível em linha reta.

Isto posto, é necessário criar uma função recompensa que traduza matematicamente, em função das leituras sensoriais e da última ação do robô, as frases enunciadas acima. O caminho mais direto para isso é expressar r_t como uma soma ponderada de 4 termos, cada um relacionado com um dos objetivos acima. Assim, faz-se:

$$r_t = k_1 r_t^1 + k_2 r_t^2 + k_3 r_t^3 + k_4 r_t^4. \tag{7.5}$$

Primeiramente, deve-se definir as funções recompensa r_t^1, r_t^2, r_t^3 e r_t^4 para cada um dos 4 objetivos. Para a colisão, toma-se a seguinte função:

$$r_t^1 = \begin{cases} -1 & \text{se colidiu no tempo } t \\ 0 & \text{c.c.} \end{cases} \tag{7.6}$$

Esse tipo de recompensa é chamado de recompensa esparsa, pois é quase sempre 0. Além disso, é uma recompensa que assume apenas dois valores. Assim, o agente não recebe qualquer sinal de que

colidir com a parede é bom ou ruim até que pelo menos uma colisão tenha se efetivado. Este é um desafio importante para o algoritmo de aprendizado por reforço.

Para o objetivo de manter-se distante dos objetos de cor laranja, define-se r_t^2 como:

$$r_t^2 = -\sqrt{a_t^r}, \tag{7.7}$$

ou seja, a recompensa é tão mais negativa quanto maior a área do maior *blob* laranja detectado. Essa forma de função-recompensa, ao contrário da anterior, é densa. Consiste numa formulação que facilita o aprendizado, pois dá pistas precoces de que aproximar-se do objeto laranja é ruim. Escolheu-se uma função densa para este objetivo porque a frequência de colisão com objetos (laranja ou verdes) é algumas ordens de grandeza menor que a frequência de colisão com outros obstáculos, como paredes, para os ambientes dos experimentos efetuados. Se a recompensa fosse esparsa, a probabilidade de o robô colidir com um objeto laranja e adquirir a experiência da recompensa negativa seria muito baixa (Smart et al., 2002).

Além disso, escolheu-se tomar a raiz quadrada da área do *blob* por dois motivos: a) para que o crescimento se dê uniformemente conforme a aproximação do robô, para uma melhor visualização, b) para que a relação entre o elemento a_t^r do vetor de observação e a recompensa r_t seja não-linear, impondo uma dificuldade a mais ao algoritmo de aprendizagem.

Em seguida, para o objetivo de manter-se próximo a objetos verdes, define-se r_t^3 como:

$$r_t^3 = \sqrt{a_t^g}, \tag{7.8}$$

cuja justificativa é análoga ao caso anterior, mas empregando-se o raciocínio inverso.

Finalmente, para que o robô se mantenha em linha reta, define-se r_t^4 como:

$$r_t^4 = -|\omega_{t-1}|, \tag{7.9}$$

ou seja, pune-se o robô proporcionalmente à sua velocidade-alvo de rotação no instante anterior.

Escolha das constantes

Escolhidas as recompensas para cada um dos objetivos, surge a necessidade de ponderar cada um deles, ou seja, atribuir valores para as constantes k_1, k_2, k_3 e k_4. Os 4 objetivos enunciados acima de forma textual não fornecem informação suficiente para determinar tais constantes. Dão apenas uma pista, através da ordenação de prioridades. O ideal seria que nunca as prioridades fossem quebradas. Apesar de isso nem sempre ser possível, em alguns casos pode-se encontrar relações que garantam o respeito às prioridades.

Tome-se por exemplo a relação entre os objetivos "colisão com obstáculo" e "andar em linha reta". Para cada intervalo de tempo que o robô não anda em linha reta, uma recompensa constante (provavelmente negativa) $k_4 r^4$ é recebida. Assim, se n ações seguidas do robô envolvem curva, o retorno acumulado será $k_4 r^4 \sum_{i=1}^{n} \gamma^{i-1}$. Em contrapartida, a recompensa recebida por uma colisão é $k_1 r^1$. Assim, caso se deseja que o robô execute no máximo n_{max} curvas para evitar uma colisão, deve-se ter:

$$k_1 r^1 > k_4 r^4 \sum_{i=1}^{n} \gamma^{i-1}, \quad \text{sse } n \geq n_{max}$$

ou seja, em módulo, a recompensa recebida pela colisão deve ser maior do que a recompensa acumulada da curva efetuada. Assim,

$$\frac{k_1}{k_4} > \frac{r^4}{r^1} \sum_{i=1}^{n_{max}} \gamma^{i-1}.$$

No limite em que $n_{max} \to \infty$, tem-se:

$$\frac{k_1}{k_4} > \frac{r^4}{r^1}(1-\gamma)^{-1}. \tag{7.10}$$

Portanto, respeitando-se a relação acima, garante-se que nunca a punição por efetuar curvas supere a punição de uma colisão.

Para poder melhor comparar os diferentes métodos, todos os experimentos realizados utilizarão as mesmas constantes k_1, k_2, k_3 e k_4. A seguir, discorre-se brevemente sobre o procedimento tomado para definir os valores para cada uma delas.

Primeiramente, sabe-se que, para fins de maximização, as quatro constantes podem ser multiplicadas por um mesmo valor positivo sem que o resultado seja alterado. Assim, pode-se arbitrar uma das constantes e escolher as outras em razão desta primeira. Tomou-se, então,

$$k_1 = 100.$$

Em seguida, a constante k_4 foi escolhida de forma a respeitar a eq. 7.10 para $\gamma = 0,95$ e para uma curva realizada à velocidade angular $\omega = 1,0$ rad/s (nos experimentos, o robô faz curvas a $\omega = 0,8$ rad/s). Tem-se

$$\frac{k_1}{k_4} = \frac{r_4}{r_1}(1-0,95)^{-1} = \frac{-1,0}{-1} \times 20 = 20.$$

Então,

$$k_4 = \frac{1}{20} k_1 = 5.$$

Finalmente, escolheram-se as constantes k_3 e k_4 empiricamente. Procurou-se escolher constantes que proporcionassem um retorno que não compensasse a punição de uma colisão. Além disso, tentou-

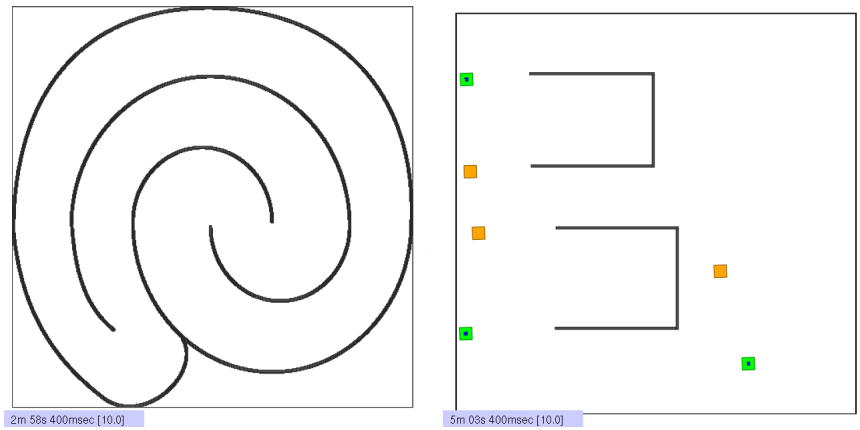


Fig. 7.7: Ambientes utilizados nos experimentos.

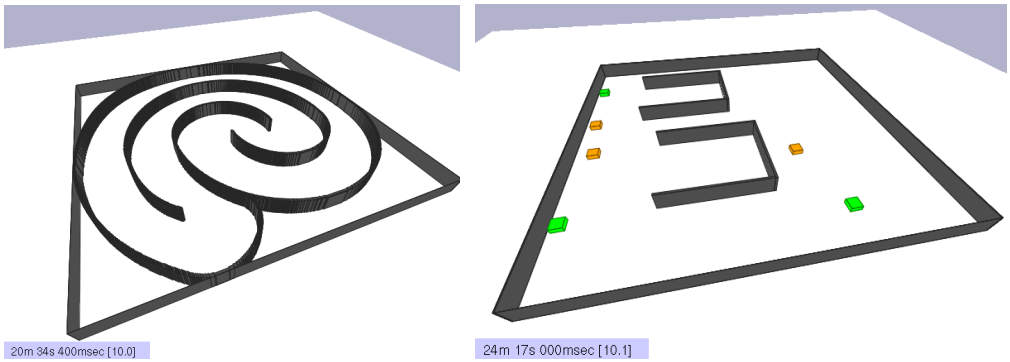


Fig. 7.8: Visão em perspectiva dos ambientes utilizados.

se verificar, em alguns testes, quais as constantes que faziam com que o robô se aproximasse mais dos alvos verdes. Chegou-se aos seguintes valores:

$$k_2 = 0,25 \text{ e } k_3 = 0,50.$$

Em suma, as constantes escolhidas foram:

$$k_1 = 100, \quad k_2 = 0,25, \quad k_3 = 0,50 \text{ e } k_4 = 5.$$

7.1.4 Ambiente

Os ambientes nos quais o robô se locomove são retangulares, constituídos de espaços livres e paredes, além de objetos verdes (presas) e laranjas (predadores). Ao longo dos experimentos, são utilizados dois ambientes diferentes: a) espiral dupla; b) ambiente em duplo "U". Os dois ambientes são mostrados na Fig. 7.7, e com uma visão em perspectiva tridimensional na Fig. 7.8.

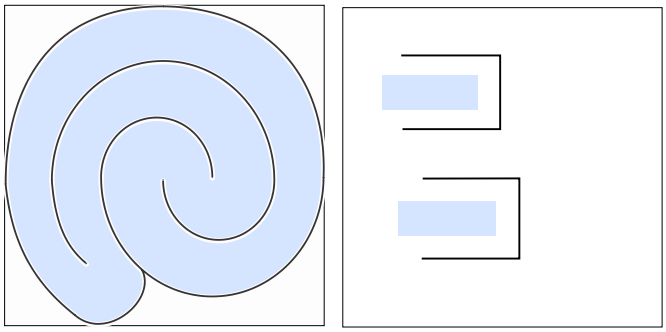


Fig. 7.9: Regiões onde o robô pode ser posicionado após colisão, em azul claro.

Cada vez que ocorre uma colisão do robô com uma parede, o robô é reposicionado em uma posição aleatória seguindo uma distribuição uniforme entre as permitidas pelo ambiente. Assim, além da definição do mapa e das posições e características dos alvos, o ambiente necessita de um mapa de possíveis posições onde o robô será reposicionado em caso de colisão. A Fig. 7.9 mostra, em azul, as posições onde é possível que o robô seja posicionado.

Como mostrado nas figuras, o ambiente em dupla espiral não apresenta alvos verdes ou laranja. Dessa forma, o robô não necessita, nesse ambiente, aprender a maximizar os termos r_2 e r_3 da recompensa, relacionados aos alvos. Já no ambiente em duplo "U", exitem 3 alvos verdes e 3 alvos laranja. No início de cada execução dos experimentos, os alvos são posicionados como mostra a Fig. 7.7, com um alvo verde e um laranja próximos à saída de cada um dos "U". Essa escolha é uma estratégia para que o robô tenha contato com tais alvos desde as primeiras execuções e possa aprender que observá-los proporciona recompensas.

Ao longo dos experimentos, sempre que o robô se aproxima suficientemente de um alvo, de forma que a área a_g ou a_r tome um valor maior que $1000\ \text{pixels}^2$, o alvo mais próximo do robô é reposicionado. Ou seja, considera-se que o robô atingiu o alvo, então deve-se reposicioná-lo. O alvo é reposicionado em uma posição aleatória seguindo uma distribuição uniforme, sempre no exterior dos "U". Assim, torna-se mais difícil para o robô atingi-los.

O reposicionamento dos alvos proporciona uma maior gama de experiência ao robô. Porém, o ambiente se torna menos determinístico. Logo, deve-se esperar que o retorno recebido pelo controlador apresente uma variância maior. Isso significa que pode ocorrer uma queda de performance significativa ao longo do aprendizado, mesmo que o controlador tenha melhorado.

7.1.5 Avaliação

A princípio, pode-se utilizar a recompensa média, ou então o retorno, como definido na eq.4.2, para medir a performance dos algoritmos. Porém, deve-se frisar que tal performance é fortemente

dependente da função recompensa escolhida. Assim, dada uma função recompensa ruim, é possível que mesmo que o algoritmo de aprendizado por reforço consiga maximizá-la, a tarefa desejada não seja executada.

Assim, além da maximização da recompensa, serão utilizados critérios de avaliação de performance específicos para cada um dos ambientes em questão. Para a espiral, será feita uma inspeção visual das trajetórias efetuadas pelo robô em torno das espirais. Para o labirinto em duplo U, será contado o número de vezes que o robô se aproxima dos alvos verde e laranja.

7.2 Resultados

Para obter os resultados a seguir, cada experimento foi executado 5 vezes, de forma independente, e com o controlador reiniciado a cada repetição, ou seja, a aprendizagem deve se realizar cada uma das vezes desde o início.

Cada rodada dos experimentos consistiu numa sequência de 20100 ciclos de controle, o que corresponde a um tempo simulado de exatamente 67 min (leituras de sensor são disparadas a cada $0,2$ s). Porém, os algoritmos utilizados puderam ser executados em uma velocidade $10\times$ mais rápida que a real. Assim, cada episódio foi simulado num tempo aproximado de 7 min.

Ao longo de cada execução, foram registradas as recompensas instantâneas, bem como cada um dos 4 termos da recompensa explicitados na eq. 7.5. Nas próximas seções, serão apresentados resultados para os dois diferentes ambientes. Além disso, alguns elementos do processo de aprendizagem serão analisados, bem como os parâmetros do controlador resultante.

7.2.1 Abordagem 1 com dupla espiral

Os resultados apresentados nos gráficos a seguir correspondem à seguinte configuração:

Ambiente	Dupla espiral
Algoritmo	Gradiente de política
Parametrização	Linear na entrada, eq. 7.3
Fator de desconto γ	0,95
Taxa de aprendizagem α	$7,5 \times 10^{-3}\ \mathrm{min}^{-1}$
Tempo entre atualizações de política τ	2000 ciclos (6,7 min)

Os parâmetros do algoritmo (taxa e período de aprendizagem) foram escolhidos empiricamente. Acredita-se que o período de aprendizagem esteja relacionado com o tempo que o agente leva para explorar de forma satisfatória os estados do ambiente, de forma a explorar sua ergodicidade. A taxa

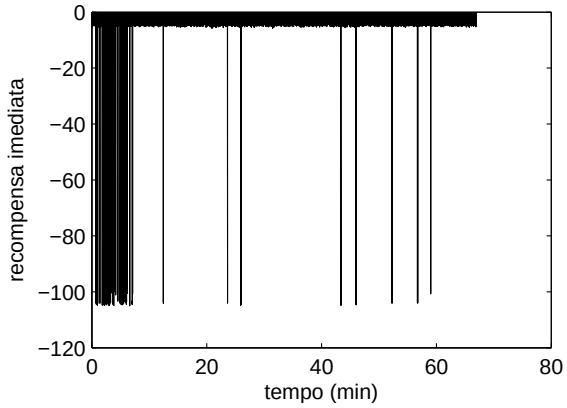


Fig. 7.10: Abordagem 1, espiral: Recompensa imediata recebida ao longo da execução 1.

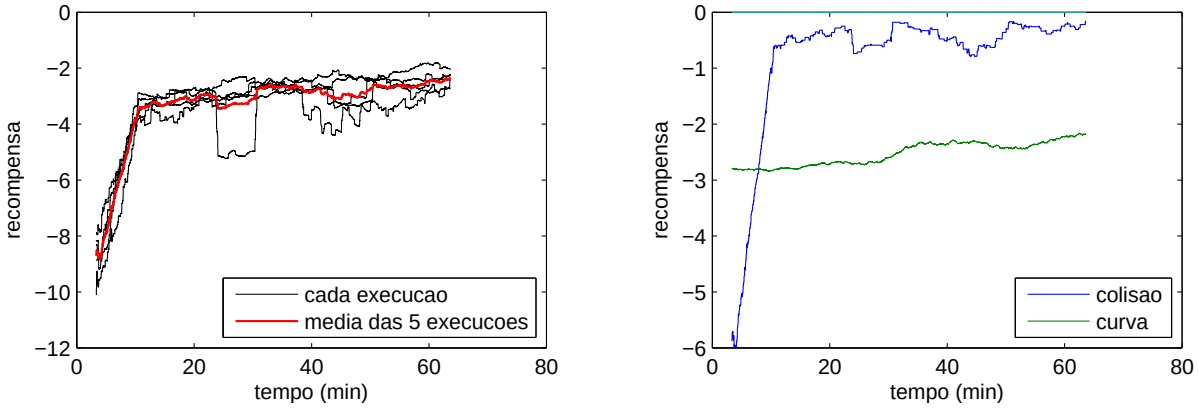


Fig. 7.11: Abordagem 1, espiral: Média móvel da recompensa imediata com *lag* de $6,6$ min (2k ciclos de controle). (a) Recompensa total de cada uma das execuções e média; (b) Recompensa média das 5 execuções, decomposta em cada um dos termos da eq. 7.5.

de aprendizagem, por sua vez, parece estar relacionada à aleatoriedade do sistema. Quanto menos determinísticas as transições de estado e as recompensas, menor deve ser a taxa de aprendizagem.

Recompensas recebidas

A Fig. 7.10 mostra as recompensas imediatas recebidas ao longo do tempo pelo agente na execução 1 do experimento. Os picos negativos da função correspondem a colisões com uma parede. O ruído, de intensidade bem menor, corresponde à recompensa negativa recebida cada vez que o agente faz uma curva. Apesar de ser possível notar que a frequência de colisões diminui ao longo do tempo, é necessário uma forma melhor de visualizar a recompensa a longo prazo.

Uma forma de fazê-lo é utilizando a média móvel. Define-se a média móvel $\bar{x}_t$ com *lag* L de uma

sequência x_t, $1 < t < N$ como:

$$\bar{x}_t = \frac{1}{L} \sum_{\tau=-L/2}^{L/2-1} x_{t+\tau}, \quad L/2 < t < N - L/2 \tag{7.11}$$

A média móvel revela tendências de longo prazo de uma sequência temporal. O significado exato de *longo prazo* é definido pelo parâmetro *lag* L, uma janela sobre a qual é efetuada a média.

A Fig. 7.11 mostra os resultados da aplicação da média móvel sobre as recompensas imediatas, com lag $L = 6,6$ min (2000 ciclos de controle). Pela parte (a) da figura, percebe-se que, nos primeiros 10 min, há uma forte tendência de crescimento da recompensa. A partir disso, continua havendo crescimento, mas de forma mais discreta. Nota-se, que as 5 execuções apresentam a mesma tendência.

A Fig. 7.11-b mostra a decomposição da recompensa nos seus quatro termos. Na verdade, como nesse ambiente não há nem presa nem predador, os dois últimos termos da recompensa são sempre nulos. Tal decomposição evidencia, porém, uma aprendizagem em dois tempos: nos primeiros 10 min, o robô aprende a desviar de obstáculos. Em seguida, ocorre a aprendizagem de "percurso em linha reta". É possível ver que, até o final do experimento, há uma tendência de diminuição da punição por curva efetuada.

Finalmente, é marcante que o aprendizado tenha sido, em média, monotônico do início ao fim dos experimentos, ou seja: não ocorre esquecimento, e a função objetivo sempre aumenta. De fato, teoricamente, algoritmos de gradiente de política garantem convergência para um máximo local de retorno. Porém, essas garantias dependem de algumas suposições que nem sempre são válidas, entre elas: o agente deve seguir uma trajetória ergódica, ou seja, deve poder percorrer todos os estados do sistema, sem ficar preso a um conjunto de sub-estados. Além disso, a função da política do agente deve ser derivável. É perfeitamente possível que o robô assuma uma trajetória não-ergódica (se ficar preso em algum beco, por exemplo), ou que a política se torne não derivável (quando há convergência para uma política determinística). Nas 5 execuções desse experimento, nenhuma dessas condições excepcionais foi atingida.

Trajetórias do robô

A Fig. 7.12 mostra 5 excertos de trajetórias efetuadas pelo robô ao longo da primeira execução do experimento. Cada um dos 5 quadros mostra o percurso do robô ao longo de 7 min (2100 ciclos de controle). Como pode se ver, no primeiro quadro, o robô apresenta um percurso completamente aleatório. Isso decorre do fato de que o controlador apresenta como saída as probabilidades $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ para cada uma das ações disponíveis, para qualquer leitura de sensor. Assim, o robô tem a mesma probabilidade de ir em frente, dobrar à esquerda ou à direita. Dado o ambiente, é inevitável que

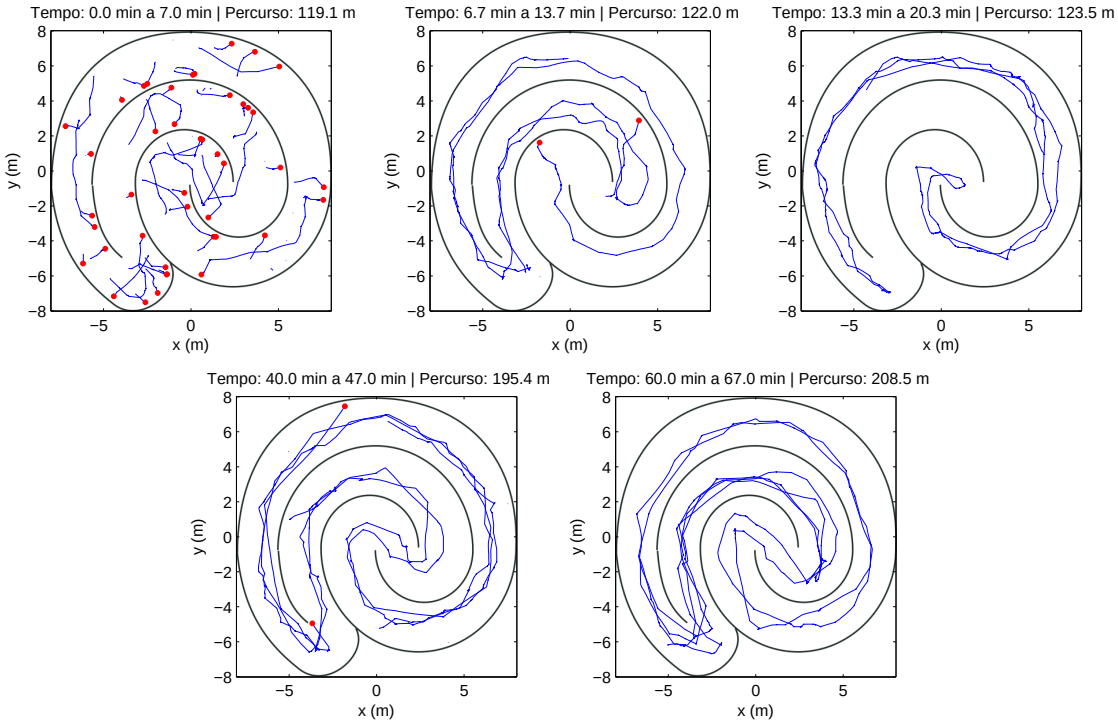


Fig. 7.12: Abordagem 1, espiral: Excertos de trajetórias ao longo da execução 1. Os círculos vermelhos são colisões.

ocorram colisões com as paredes. São 43 colisões, ou seja, uma média de 6 colisões por minuto. Mesmo assim, o robô pôde efetuar um trajeto de 119 m, ou seja, uma média de $0,28$ m/s.

No segundo e terceiro quadros, é possível notar uma grande melhoria no percurso do robô: praticamente não há mais colisões. Além disso, há um aumento da distância total percorrida. Em ambos os quadros, o robô se desloca a uma velocidade média de $0,29$ m/s, um pouco maior que no início.

Finalmente, o quinto quadro, que retrata os 7 min finais do experimento, revelam um robô que se desloca a $0,50$ m/s, quase o dobro da velocidade média dos quadros anteriores. Observando as trajetórias do robô retratadas no quadro, percebe-se que o mesmo desloca-se em linha reta por mais tempo, quando se encontra longe da parede. Além disso, o robô percorre o labirinto completamente, o que decorre do fato que ele efetua menos curvas (não dá meia-volta). Nesse percurso final, o robô não colide com obstáculos.

Através da inspeção acima, é possível verificar que o algoritmo de aprendizagem utilizado, unido à definição das recompensas, apresentada na seção anterior, foi capaz de levar o robô ao comportamento desejado, no ambiente em questão.

Variação da entropia

No início da aprendizagem, espera-se que o controlador tome ações completamente exploratórias, independente de qual a leitura de sensor. Por isso, o controlador é inicializado de forma que produza uma probabilidade igual de executar cada uma das ações, não importando qual a entrada do sensor. Ao longo do tempo, o comportamento exploratório deve dar lugar ao comportamento de explotação, ou seja, ações que levam a recompensas maiores devem ter preferência. A redução da exploração deve implicar uma redução, ao longo do tempo, da entropia condicional $H(a|o)$ do controlador. Procura-se aqui analisar a evolução de $H(a|o)$ ao longo da aprendizagem, e se possível estimar a variação $\frac{d}{dt}H(a|o)$.

A entropia condicional $H(a|o)$ é uma medida da incerteza sobre a quando se conhece o. Como o pode assumir diferentes valores, toma-se a esperança em relação à distribuição de o. Então,

$$H(a|o) \equiv \int_{o \in \mathbf{O}} H(a \mid \mathbf{o} = o) p(o) do, \tag{7.12}$$

onde

$$H(a|\mathbf{o} = o) = -\sum_{a \in \mathbf{A}} \Pr\{a \mid \mathbf{o} = o\} \log \Pr\{a \mid \mathbf{o} = o\}. \tag{7.13}$$

Com base na sequência de observações o_t e da sequência de probabilidades de ação,

$$\Big(\pi(a_1|o_t),\ \pi(a_2|o_t),\ \pi(a_3|o_t)\Big), \tag{7.14}$$

que são produzidas pelo controlador do robô a cada instante de tempo em função da entrada, no caso o_t, é possível estimar o valor de $H(a|o)$. Primeiramente, calcula-se $H(a_t|\mathbf{o} = o_t)$ para cada tempo t diretamente pela eq. 7.13, da seguinte forma:

$$H(a_t \mid \mathbf{o} = o_t) = -\sum_{i=1}^{3} \pi(\mathbf{a}_t = a_i \mid o_t) \log \pi(\mathbf{a}_t \mid o_t). \tag{7.15}$$

A eq. 7.12, por sua vez, envolve uma integral no espaço de observações, cuja distribuição de probabilidades não é conhecida. Porém, considerando-se que o sistema é ergódico, é possível substituir a integral no espaço de observações por uma média no tempo. Assim, considera-se que a média móvel de $H(a_t|\mathbf{o} = o_t)$ pode ser uma boa representação de $H(a|o)$, para uma escolha adequada do *lag*.

A Fig. 7.13 mostra a média móvel da entropia $H(a_t|\mathbf{o} = o_t)$ obtida pela média das 5 execuções do experimento. Pode-se notar que há uma queda ao longo do tempo, conforme esperado. De fato, o valor inicial corresponde à entropia de uma distribuição trinomial uniforme ($log_2 3 = 1,58$ bit), que atribui a cada uma das três ações uma igual probabilidade. Ao longo do tempo, a entropia cai

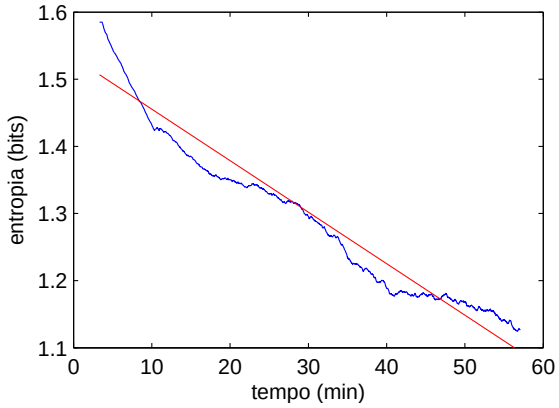


Fig. 7.13: Abordagem 1, espiral: Média móvel (*lag* de $6,6$ min) da entropia $H(a_t|\mathbf{o} = o_t)$ na saída do controlador, média das 5 execuções. Em vermelho, regressão linear.

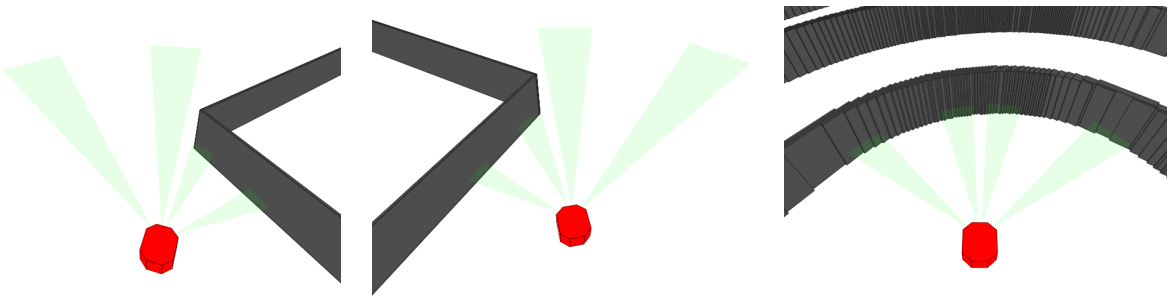
monotonicamente, chegando quase a 1 bit.

Aplicando-se uma regressão linear sobre $H(a_t|\mathbf{o} = o_t)$ ao longo do tempo, é possível obter uma aproximação para a taxa de variação da entropia. A regressão é mostrada por uma reta vermelha na Fig. 7.13. O coeficiente angular $\bar{b}$ obtido deve ser uma boa estimação para $\frac{d}{dt}H(a|o)$. Obtém-se

$$\bar{b} = -7,7 \times 10^{-3} \text{ bits/min.} \tag{7.16}$$

Controlador resultante

Como especificado na seção 6.4, o controlador cujos parâmetros foram ajustados pelo algoritmo de aprendizado consiste em um classificador logístico. Ao início do aprendizado, todos os parâmetros θ do classificador têm o valor 0, de forma a se obter probabilidade igual de execução de cada uma das ações, independente da entrada ϕ_t. Ao longo do tempo, o gradiente de política é responsável por modificar tais parâmetros. Nesta seção, são apresentados os parâmetros e algumas das características do controlador obtido nas iterações finais da primeira execução do experimento.



(a) (b) (c)

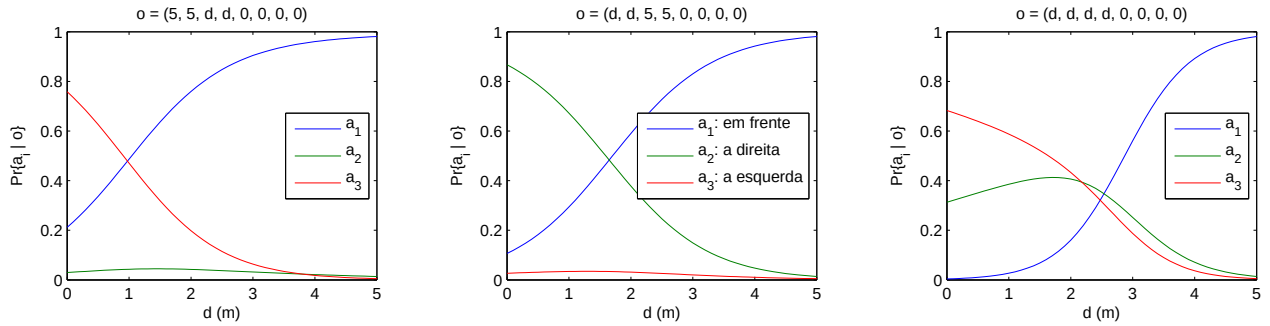
Fig. 7.14: Abordagem 1, espiral: Exemplos de situações que podem ser encontradas pelo robô. As situações são gerais, não se referindo especificamente ao ambiente de dupla espiral.

Os parâmetros θ obtidos ao fim da última iteração são:

$$\theta = \begin{bmatrix} -5,168 & -0,780 \\ 0,195 & -0,281 \\ 0,583 & -0,212 \\ 0,954 & 0,496 \\ 0,356 & 0,355 \\ 0,000 & 0,000 \\ 0,000 & 0,000 \\ 0,000 & 0,000 \\ 0,000 & 0,000 \end{bmatrix}$$

Uma vez definidos os parâmetros, as observações o_t, através da eq. 7.3 são fornecidas ao classificador da eq. 6.39 a fim de obter as probabilidades de execução de cada uma das ações. Percebe-se que, como nesse experimento não há alvos, nem verdes nem laranja, os parâmetros do controlador relacionados às entradas sensoriais dos alvos não são aprendidos, e mantêm-se em 0. A seguir, encontram-se as respostas fornecidas pelo controlador para algumas situações específicas encontradas pelo robô.

A Fig. 7.14 mostra diferentes situações com as quais o robô pode se deparar ao longo do experimento. Nas três situações, dependendo de quão perto o robô se encontra do obstáculo, ocorre risco de choque com a parede. Logo, o robô é obrigado a tomar uma ação de curva. Como fazer curva leva a recompensas negativas, o robô deve girar o mínimo possível para desviar do obstáculo. Nas situações (a) e (b), percebe-se claramente que o melhor é girar, respectivamente, à esquerda e à direita. Na situação (c) não há um lado ótimo, mas o robô deve decidir girar para um dos lados. Se for atribuída probabilidade igual para ambos os lados, o robô ficará preso. Além disso, quando o robô se encontra suficientemente longe dos obstáculos, é vantajoso seguir reto, para evitar as recompensas negativa causadas por ações de giro.



(a) (b) (c)

Fig. 7.15: Abordagem 1, espiral: Probabilidades $\Pr\{a \mid o\}$ de ação produzidas pelo controlador para as diferentes situações representadas na Fig. 7.14

A Fig. 7.15 mostra as probabilidades de cada uma das ações fornecidas pelo controlador, quando se fazem variar alguns dos valores do vetor de observação o. O controlador usado é aquele obtido ao final da execução 1 do experimento. Cada um dos gráficos da figura corresponde a uma das situações retratadas na Fig. 7.14, como mostrado a seguir:

Na situação (a), o robô detecta obstáculos nos seus dois sonares à direita. Suponha que tais obstáculos encontrem-se ambos a uma distância d do robô. Então, $d^3 = d^4 = d$. Os dois sonares à esquerda não detectam obstáculos, ou seja, retornam leituras $d^1 = d^2 = 5$. Além disso, não há nenhum *blob*, nem verde nem laranja, visível na câmera do robô. Assim, $a^r = a^g = x^r = x^g = 0$. O vetor o de observação é então:

$$o = \Big(d,\ d,\ 5,\ 5,\ 0,\ 0,\ 0,\ 0\Big). \tag{7.17}$$

Fornecendo este vetor o de observação à entrada do controlador, e fazendo-se variar d de 0 a 5, obtêm-se as probabilidades de ação para cada uma das ações a_1, a_2 e a_3 em função de d. O resultado é mostrado na Fig. 7.15-a. Percebe-se que, como esperado, para distâncias pequenas ($d < 1$ m), o controlador atribui probabilidade maior à ação a_3 (vire à esquerda). Para distâncias maiores, a ação mais provável é a_1 (siga em frente). A ação a_2 (vire à direita) tem probabilidade quase nula, independente de d.

Na situação (b), por sua vez, não há obstáculos à direita, portanto, $d^3 = d^4 = 5$. Há, porém, obstáculos à esquerda, a uma distância d do robô. Assim, $d^1 = d^2 = d$ e vetor de observações é, então:

$$o = \Big(5,\ 5,\ d,\ d,\ 0,\ 0,\ 0,\ 0\Big). \tag{7.18}$$

Para tal vetor de observação, o controlador gera as probabilidades da Fig. 7.15-b. Para distâncias $d < 2$, o controlador atribui maior probabilidade à ação a_2 (vire à esquerda) e, para distâncias maiores, dá maior probabilidade a a_1 (siga em frente).

Finalmente, na situação (c), os quatro sonares registram um obstáculo a uma distância d do sensor.

Tem-se, então,

$$o = \Big(d,\ d,\ d,\ d,\ 0,\ 0,\ 0,\ 0\Big). \tag{7.19}$$

As probabilidades geradas por tal vetor de observação são retratadas na Fig. 7.15-c. Como discutido anteriormente, apesar dessa situação trazer retornos equivalentes independente da ação (a_1 ou a_2) escolhida, é importante que o controlador dê mais probabilidade a uma delas, para que o robô não fique preso. No caso, devido a ruídos durante o treinamento ou a características particulares do ambiente, o controlador dá uma probabilidade maior de virar à esquerda (ação a_3). Se a distância for maior que um limiar ($d > 2,5$ m), novamente a probabilidade de seguir reto é maior.

Nota-se que, por alguma razão, os limiares de decisão entre as ações de curva a_2 e a_3 e de seguir reto a_1 são diferentes para cada uma das situações. Novamente, tal diferenciação gerada pelo algoritmo de aprendizado pode vir simplesmente de ruído de treinamento ou então reflete alguma característica peculiar ao ambiente em questão. Analisando-se os resultados das outras execuções do experimento, os limiares são diferentes a cada vez, portanto não foi possível chegar a uma conclusão definitiva quanto a essa questão.

De qualquer forma, a análise acima explica como o controlador resultante do algoritmo de aprendizado é capaz de evitar obstáculos percorrendo o máximo possível do caminho em linha reta.

7.2.2 Abordagem 1 com labirinto em U

As seguintes rodadas de experimento foram efetuadas com a mesma configuração daqueles da seção anterior. Novamente, foram executadas 5 repetições, cada uma com 20100 ciclos de controle. A única diferença é que o ambiente da simulação é outro. A seguir, as especificações:

Ambiente	Labirinto em U
Algoritmo	Gradiente de política
Parametrização	Linear na entrada, eq. 7.3
Fator de desconto γ	0,95
Taxa de aprendizagem α	$7,5 \times 10^{-3}\ \mathrm{min}^{-1}$
Tempo entre atualizações de política τ	2000 ciclos (6,7 min)

Recompensas recebidas

A Fig. 7.16 mostra a média móvel da recompensa recebida ao longo das 5 execuções do experimento com ambiente em U. Percebe-se que, apesar de haver uma tendência de aumento da recompensa ao longo do tempo, há uma queda brusca de recompensa. Analisando-se a figura, nota-se que essa queda ocorre na terceira execução, e que é causada pela redução no termo da recompensa relacionado ao alvo verde, especificado na eq. 7.8.

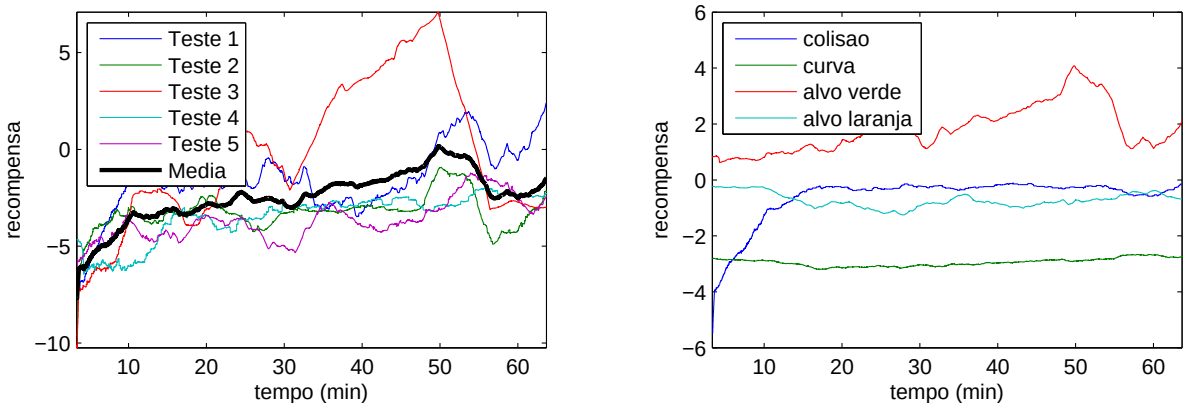


Fig. 7.16: Abordagem 1, duplo U: Média móvel da recompensa imediata com *lag* de 6.6 min (2k ciclos de controle). (a) Recompensa total de cada uma das execuções e média; (b) Recompensa média das 5 execuções, decomposta em cada um dos termos da eq. 7.5.

À exceção dessa ocasião excepcional, que ocorre apenas uma vez e que será analisada posteriormente, é possível notar que houve uma melhoria, principalmente nos termos de recompensa relacionados à desvio de obstáculos (eq. 7.8) e ao alvo verde. Quanto ao alvo laranja e à curva, não houve muita melhoria. Nota-se que o objetivo de reduzir a quantidade de curvas (eq. 7.9) é conflitante com os outros objetivos e, portanto, não se deve esperar melhora expressiva deste termo.

Trajetórias do robô

Algumas sequências de trajetória, efetuadas pelos robô durante a execução 1, são retratados na Fig. 7.17. Na figura, estão retratados os alvos, em verde e laranja. Deve-se entender que, a cada instante de tempo, há no ambiente apenas 3 alvos verdes e 3 laranja. Algumas figuras retratam mais alvos porque, quando o robô se aproxima de um alvo, este é reposicionado e, por isso, eles aparecem mais de uma vez na figura. Os pontos pretos da figura correspondem às posições onde o robô estava quando o alvo ficou próximo o suficiente para ser reposicionado.

Nas primeiras iterações, o robô não possuía capacidade de desviar de obstáculos, por isso não saía nem mesmo dos "U". Após 7 min, esta habilidade foi desenvolvida. Porém, a habilidade de seguir alvos verdes e fugir de alvos laranja só se manifesta mais tarde, e de forma menos perceptível. A última sequência (sequência 6) da Fig. 7.17 mostra 15 alvos verdes. Isso significa que, no mínimo, o robô conseguiu encontrar 12 desses alvos e reposicioná-los. Isso é muito mais do que nas sequências anteriores.

Além disso, a sequência 6 da figura mostra que o robô se aproximou também de pelo menos 5 alvos laranja. Isso é um pouco mais do que nas sequências anteriores, e confirma a piora no termo

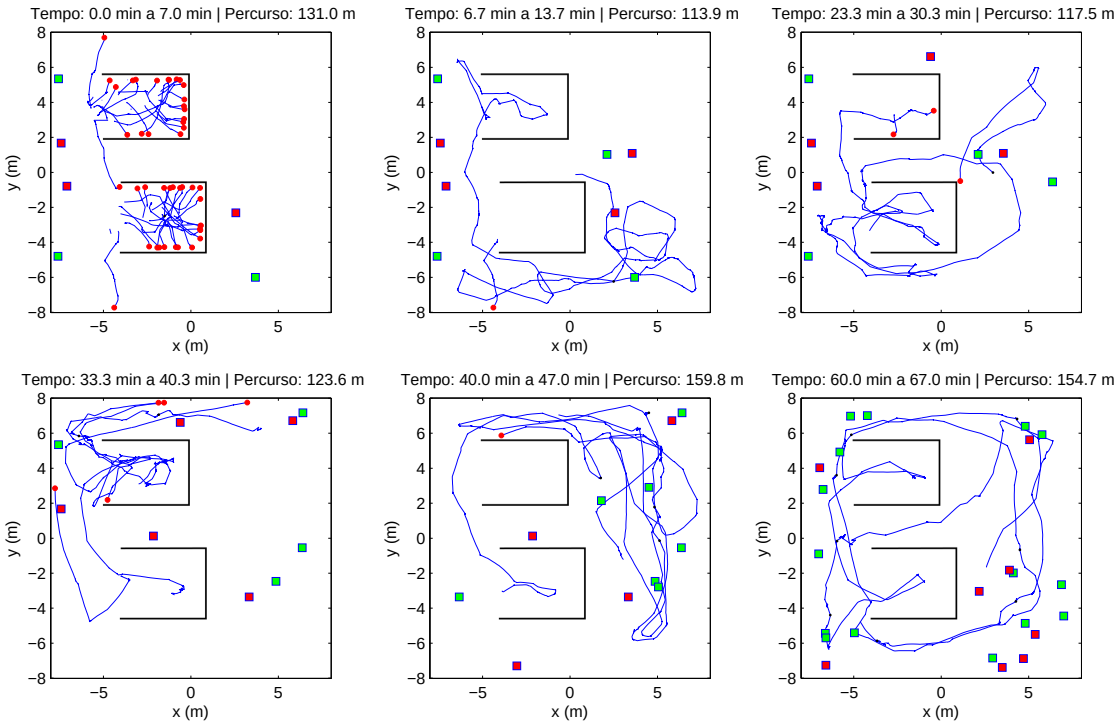


Fig. 7.17: Abordagem 1, duplo U: Excertos de trajetórias ao longo da execução 1. Os círculos vermelhos são colisões, os quadrados são os alvos. Os pontos pretos são momentos nos quais considera-se que o robô atingiu a proximidade de um alvo (verde ou laranja).

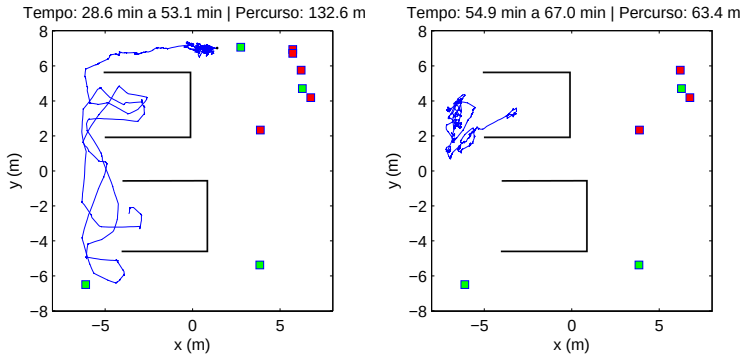


Fig. 7.18: Abordagem 1, duplo U: Exemplo de sobre-aprendizagem na execução 3. O robô recebe grande recompensa aproveitando-se de uma deficiência na arquitetura da função-recompensa, mas por isso acaba desaprendendo.

correspondente de recompensa, retratado na Fig. 7.16-b. É possível, porém, que a piora nesse termo da recompensa tenha proporcionado um aumento maior nos outros termos.

Finalmente, resta encontrar uma explicação para a piora repentina no termo relacionado ao alvo verde, na execução 3. A Fig. 7.18 mostra a trajetória do robô antes e depois da piora. Nota-se que, de alguma forma, o algoritmo explorou uma deficiência na arquitetura da função-recompensa, e conseguiu acumular um grande retorno, mantendo-se próximo ao alvo verde sem atingi-lo. Isso explica o pico de recompensa visível na Fig. 7.16-a aos 50 min.

Após isso, porém, o algoritmo de aprendizado faz um cálculo de gradiente fortemente polarizado pela alta recompensa obtida no período anterior. Provavelmente, o algoritmo associa a parede ao lado esquerdo do alvo com uma alta recompensa. Dessa forma, na sequência seguinte, retratada também na Fig. 7.18, o robô, ao se deparar com uma parede à sua esquerda, busca executar as mesmas ações que o mantinham próximo ao alvo verde, e acaba ficando preso próximo à parede.

Cabe um esclarecimento quanto ao sucesso de um controlador reativo em sair de um beco em U já que, originalmente, tal experimento havia sido proposto justamente para que robôs puramente reativos ficassem presos no beco. Isso ocorre porque, no experiento original, o robô é capaz de detectar o alvo por detrás do muro. No experimento aqui apresentado, porém, o robô não é capaz de ver os objetos verdes além das paredes. Por isso, não é atraído para eles e não fica preso.

Variação de entropia

Como para o ambiente anterior, uma estimativa da entropia condicional média ao longo do tempo encontra-se na Fig. 7.19.

Apesar de uma evidente não-linearidade presente na figura, faz-se uma regressão linear, como para o outro ambiente. O coeficiente angular $\bar{b}$ consiste em uma estimativa para a taxa média de

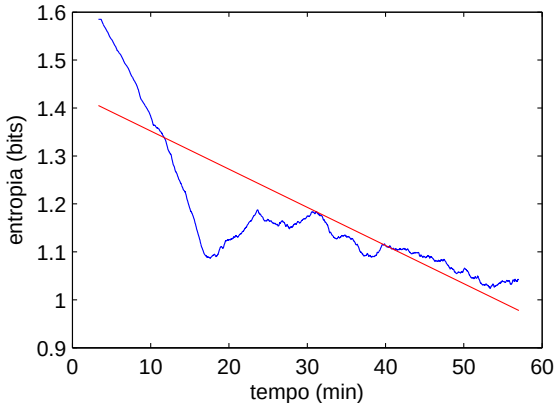


Fig. 7.19: Abordagem 1, duplo U: Média móvel (*lag* de $6,6$ min) da entropia $H(a_t|\mathbf{o} = o_t)$ na saída do controlador, média das 5 execuções. Em vermelho, regressão linear.

variação da entropia condicional, $\frac{d}{dt}H(a|o)$. Obtém-se

$$\bar{b} = -8,0 \times 10^{-3} \text{ bits/min}. \tag{7.20}$$

Os valores encontrados são muito próximos aos do ambiente em dupla espiral.

Controlador resultante

A seguir encontram-se os valores resultantes encontrados para o controlador ao fim da primeira execução do experimento.

$$\theta = \begin{bmatrix} -3,479 & 0,251 \\ -0,278 & -0,349 \\ 0,354 & -0,044 \\ 0,931 & 0,058 \\ 0,238 & 0,472 \\ -0,006 & 0,000 \\ -0,005 & -0,001 \\ 0,024 & 0,041 \\ 0,006 & -0,015 \end{bmatrix}$$

Nota-se que, diferentemente do ambiente em dupla espiral, agora os coeficientes relacionados aos alvos verde e laranja são diferentes de zero, devido à experiência obtida a respeito dos mesmos.

O aprendizado de desvio de obstáculos foi realizado da mesma forma que para o ambiente em dupla espiral, ou seja, para as três situações de proximidade de obstáculo retratadas na Fig. 7.14, o

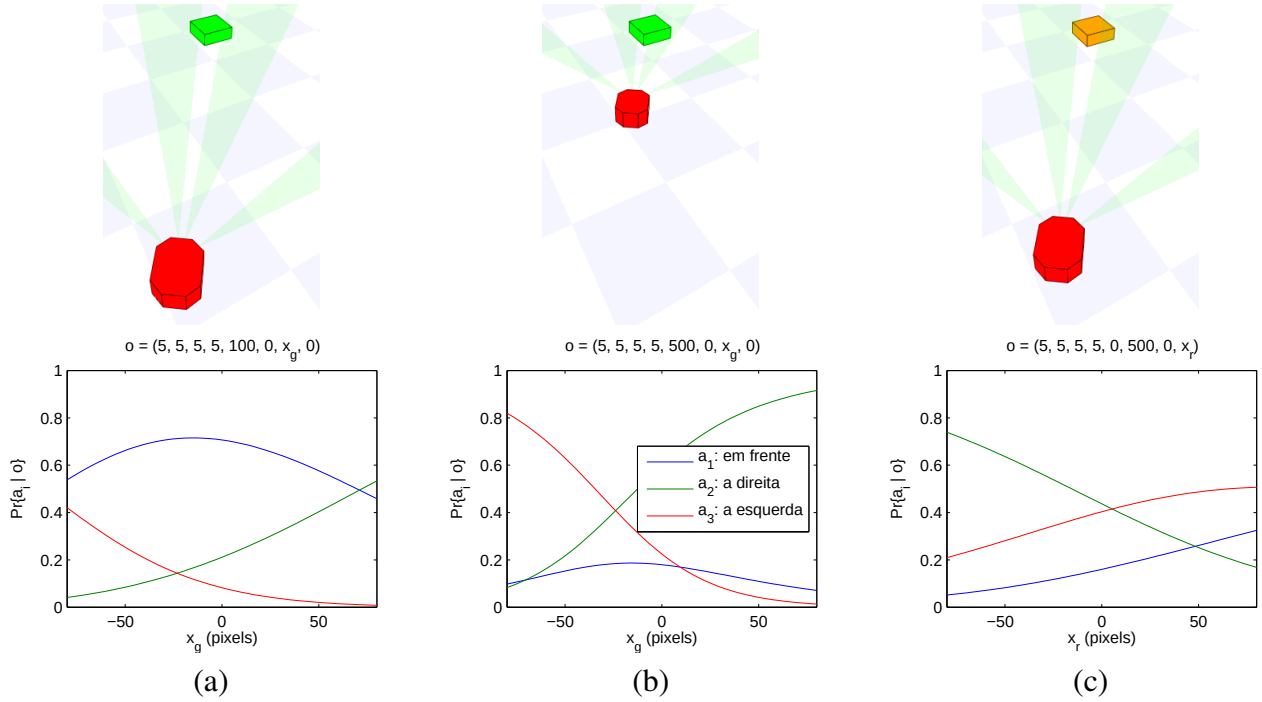


Fig. 7.20: Abordagem 1, duplo U: Exemplos de situações que podem ser encontradas pelo robô e probabilidades $\Pr\{a \mid o\}$ de ação produzidas pelo controlador para cada uma delas.

robô responde aproximadamente como na Fig. 7.15, garantindo que o robô desvie de um obstáculo com o mínimo possível de curvas.

Resta interpretar o comportamento do controlador para situações envolvendo alvos. As três situações retratadas na Fig. 7.20 correspondem, respectivamente a, a) alvo verde distante visível, b) alvo verde próximo, c) alvo laranja distante visível.

Para cada uma dessas situações, faz-se variar a posição horizontal do blob na câmera, representada por x_g ou x_r, como especificado na Fig. 7.4. Assim, se x_g ou $x_r = 80$, o centro do respectivo alvo encontra-se à esquerda do robô, e se x_g ou $x_r = 80$ px, o centro do alvo se encontra à direita do robô.

Na Fig. 7.20-a, o robô encontra-se relativamente distante ($a_g = 100$) de um alvo verde. Como, pela eq. 7.7, a recompensa é mais positiva quão mais próximo o robô se encontrar do alvo, espera-se que o robô siga em frente. Isso é exatamente o que o controlador faz. Como mostra a figura, a probabilidade de executar tal ação é maior que as duas outras ações. Além disso, conforme o alvo se encontra desalinhado horizontalmente, o controlador tende a aumentar a probabilidade de executar curvas na direção correspondente. Assim, quanto mais o alvo encontra-se à esquerda do robô ($x_g < 0$), maior a probabilidade de efetuar curva à esquerda (a_3). Se o alvo encontra-se à direita ($x_g > 0$), maior a probabilidade de executar curva à direita (a_2).

Na Fig. 7.20-a, o robô encontra-se mais próximo ($a_g = 500$) do alvo verde. Como mostra a figura, o controlador diminui a probabilidade de seguir reto, e aumenta a probabilidade de curva, de forma a

alinhar o robô na direção do alvo.

Finalmente, na Fig. 7.20-c, com o alvo laranja, as probabilidades se invertem: se o alvo se encontra à direita, a probabilidade de virar à esquerda é maior, e vice-versa. Dessa forma, o alvo laranja é encarado como uma espécie de obstáculo do qual se deve desviar com o mínimo possível de curva. Isso condiz com o fato de que a recompensa do alvo laranja é negativa, segundo a eq. 7.8.

7.2.3 Abordagem 2 com dupla espiral

Nos experimentos efetuados anteriormente, as leituras do sensor haviam alimentado o algoritmo de forma direta, como na eq. 7.3. Nesta seção, é utilizada a abordagem 2, expressa pela eq. 7.4. Assim, é o vetor de crenças $\mathbf{b}_t$ do POMDP que é utilizado como entrada do algoritmo de aprendizado. O ambiente é o da dupla espiral.

No experimento em questão, como nos outros, são realizadas 5 repetições. Em cada uma delas, segue-se o seguinte procedimento:

1. Faz-se o robô navegar por $6,7$ min (2000 iterações) no ambiente, seguindo uma política uniforme, com mesma probabilidade para cada ação;

2. Executa-se o algoritmo Baum-Welch, introduzido na seção 5.3.3, para treinar o POMDP;

3. Faz-se o robô navegar por 67 min (20100 iterações), sujeito ao algoritmo de gradiente de política, usando a crença do POMDP como entrada.

O algoritmo Baum-Welch envolve 2 parâmetros principais. O mais importante é o número n de estados. Esta constante depende da complexidade do ambiente: quanto mais situações distintas forem encontradas pelo robô, mais estados o POMDP deve possuir. No presente teste, tal parâmetro foi escolhido manualmente: executaram-se alguns testes preliminares com um número n elevado. Quando, durante o treinamento do Baum-Welch, algum dos estados não explicava nenhuma observação[1], era removido, e n era decrescido de 1. Dessa forma, para o experimento em questão, chegou-se a um valor $n = 8$.

A segunda constante do Baum-Welch é o número de iterações. Em geral, pode-se controlar a variação do logaritmo da verossimilhança ao longo das iterações e parar o treinamento quando a variação é menor que um dado limiar. Escolheram-se 10 iterações de treinamento. A Fig. 7.21, mostra a evolução do logaritmo da verossimilhança, calculado com a eq. 5.29, ao longo das iterações do

[1]Um estado s não explica nenhuma observação quando a soma das probabilidades a posteriori para tal estado é nula, ou seja, $\sum_{t=1}^{T} \gamma_t(s) = 0$. Nesse caso, o denominador das eqs. 6.20, 6.24, 6.25 e 6.26 é nulo, e portanto a estimação dos parâmetros do POMDP para o estado s não pode ser realizada, e nem faz sentido. A forma mais natural de contornar tal problema é remover s do conjunto de estados.

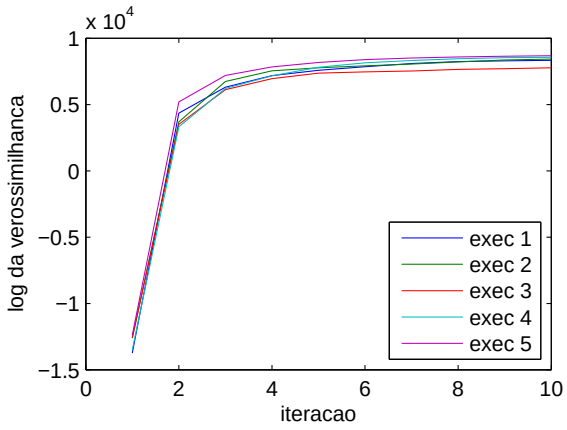


Fig. 7.21: Abordagem 2: Evolução do log da verossimilhança ao longo do Baum-Welch.

Baum-Welch para cada uma das 5 execuções do experimento. Como se pode notar, a verossimilhança aumenta mais fortemente nas primeiras 3 iterações, apos às quais tende a ficar estagnada. Isso sugere que o número de iterações poderia ser reduzido ainda mais.

Quanto à parametrização das distribuições condicionais de observação, no POMDP treinado pelo Baum-Welch, utiliza-se aquela da eq. 6.15, na qual cada dimensão da observação é representada pela mistura de uma normal e um delta de Dirac. O delta de Dirac é posicionado sobre 5 m, o valor máximo da leitura dos sonares. Em resumo, os parâmetros do POMDP a serem aprendidos pelo Baum-Welch são enunciados na eq. 6.16.

Quando aos parâmetros do gradiente de política, foram mantidos os mesmos valores usados nos experimentos anteriores. A tabela a seguir resume os parâmetros utilizados nesta abordagem:

Ambiente	Dupla espiral
Fator de desconto γ	0,95
Algoritmos	Baum-Welch + Grad. Pol.
Gradiente de política	
Parametrização	Crença na entrada, eq. 7.4
Taxa de aprendizagem α	$7,5 \times 10^{-3}$ min^{-1}
Tempo entre atualizações de política τ	2000 ciclos (6,7 min)
Baum-Welch	
Parametrizaçao do POMDP	Eq. 6.15
Dados para aprendizagem	2000 ciclos (6,7 min)
Número de iterações	10
Número n de estados	8

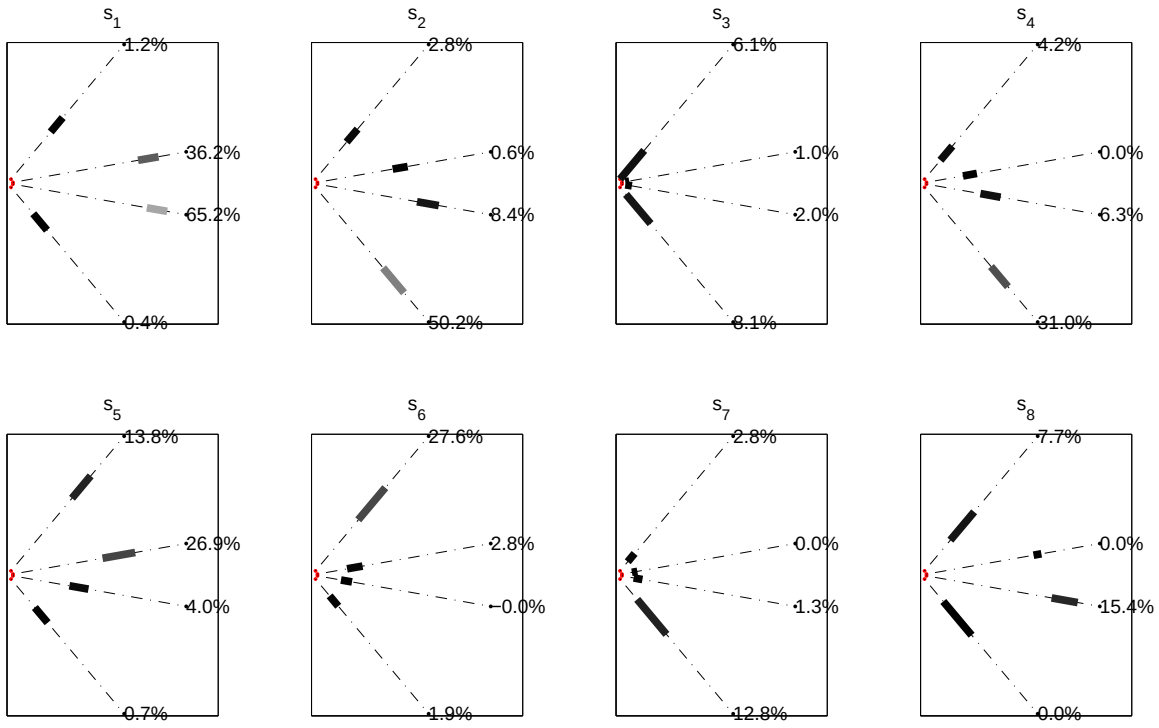


Fig. 7.22: Abordagem 2: Representação da distribuição condicional de observação, para cada um dos estados. Em cada estado, as barras sobre cada raia representam a média e o desvio-padrão da distância até o obstáculo, para cada sensor. A porcentagem corresponde à probabilidade de o sensor não encontrar nenhum obstáculo (leitura máxima) nesse estado.

Parâmetros de observação do POMDP

Logo após a primeira etapa de cada execução, o algoritmo Baum-Welch toma as trajetórias executadas pelo robô seguindo a política uniforme e treina os parâmetros do POMDP. Tais parâmetros consistem nos parâmetros das distribuições condicionais de observação e nos elementos das matrizes de transição. Nesta seção, apresentam-se os resultados obtidos na execução 1 do experimento.

Os parâmetros μ_i^k, σ_i^k e η_i^k atribuídos pelo algoritmo Baum-Welch às distribuições condicionais de observação da eq. 6.15, estão representados na Fig. 7.22. Para cada estado s_i, $i \in \{1, \ldots, 8\}$, a figura contém um quadro, no qual o robô é representado como um ponto vermelho. Do robô saem quatro raias, cada uma correspondendo ao k-ésimo sonar, $k \in \{1, \ldots, 4\}$. A média μ_i^k e o desvio padrão σ_i^k da distância a obstáculos, para o k-ésimo sensor e i-ésimo estado são representadas pela barra posicionada ao longo da raia correspondente. A cor da barra é tão mais escura quanto maior a probabilidade η_i^k de encontrar obstáculos. Já a porcentagem posicionada ao fim da raia corresponde à probabilidade de *não* se encontrar obstáculos, $1 - \eta_i^k$.

Analisando a Fig. 7.22, percebe-se que o algoritmo Baum-Welch foi capaz de separar os estados

de forma a representar diferentes situações que se apresentam ao robô. Assim, por exemplo:

- O estado s_1 representa situações no qual o robô tem uma alta probabilidade de não encontrar obstáculos à frente, mas com obstáculos dos dois lados. Corresponde, por exemplo, à situação em que o robô se encontra no centro do labirinto, com a frente direcionada de forma paralela às paredes;
- O estado s_3 representa situações em que o robô está prestes a colidir;
- O estado s_4 representa obstáculos à esquerda;
- O estado s_5 e s_6 representam obstáculos à direita.

Uma outra forma de visualizar como são distribuídos os estados a partir da observação é através da probabilidade $\Pr\{s_i \mid o\}$ de estado dada a observação. Para isso, aplica-se o teorema de Bayes sobre as probabilidades condicionais $\Pr\{o \mid s_i\}$ da eq. 6.15, da seguinte forma:

$$\Pr\{s_i \mid o\} = \frac{\Pr\{o \mid s_i\} \Pr\{s_i\}}{\sum_{j=1}^{n} \Pr\{o \mid s_j\}}, \tag{7.21}$$

utilizando-se um *a priori* não-informativo $\Pr\{s_i\} = \frac{1}{n}$. A Fig. 7.23 mostra o valor de $\Pr\{s_i \mid o\}$ para cada um dos 8 estados, em três situações distintas.

Como nas abordagens anteriores, para cada situação faz-se o parâmetro d variar, alterando a proximidade entre o robô e o obstáculo. Conforme muda o parâmetro, muda também a distribuição $\Pr\{s \mid o\}$ de estados. Pode-se ver que, em muitos casos casos, uma observação determina com probabilidade quase 1 o estado.

Pela Fig. 7.23, nota-se que o estado s_4 está mais presente quando há obstáculo próximo à esquerda, e os estados s_5 e s_6, quando há obstáculo próximo à direita. Isto condiz com a análise anterior feita na Fig. 7.22.

Finalmente, a Fig. 7.24 mostra cada posição do robô ao longo do experimento, na cor correspondente ao estado mais provável estimado pelo filtro de Bayes. O esquema de cores é o mesmo da Fig. 7.23. Percebe-se que o estado mais visível é o s_8, em cor azul, que, como comentado anteriormente, representa o robô próximo ao centro do labirinto e alinhado paralelamente a suas paredes.

Matrizes de transição do POMDP

O algoritmo Baum-Welch também treina as matrizes de transição de estados (eq. 6.4), A_1, A_2 e A_3. Os valores estimados para cada elemento dessa matriz, na execução 1 do experimento, encontram-se na Tabela 7.1. Percebe-se os valores de transição de um estado para ele mesmo (valores $a_i i$ na

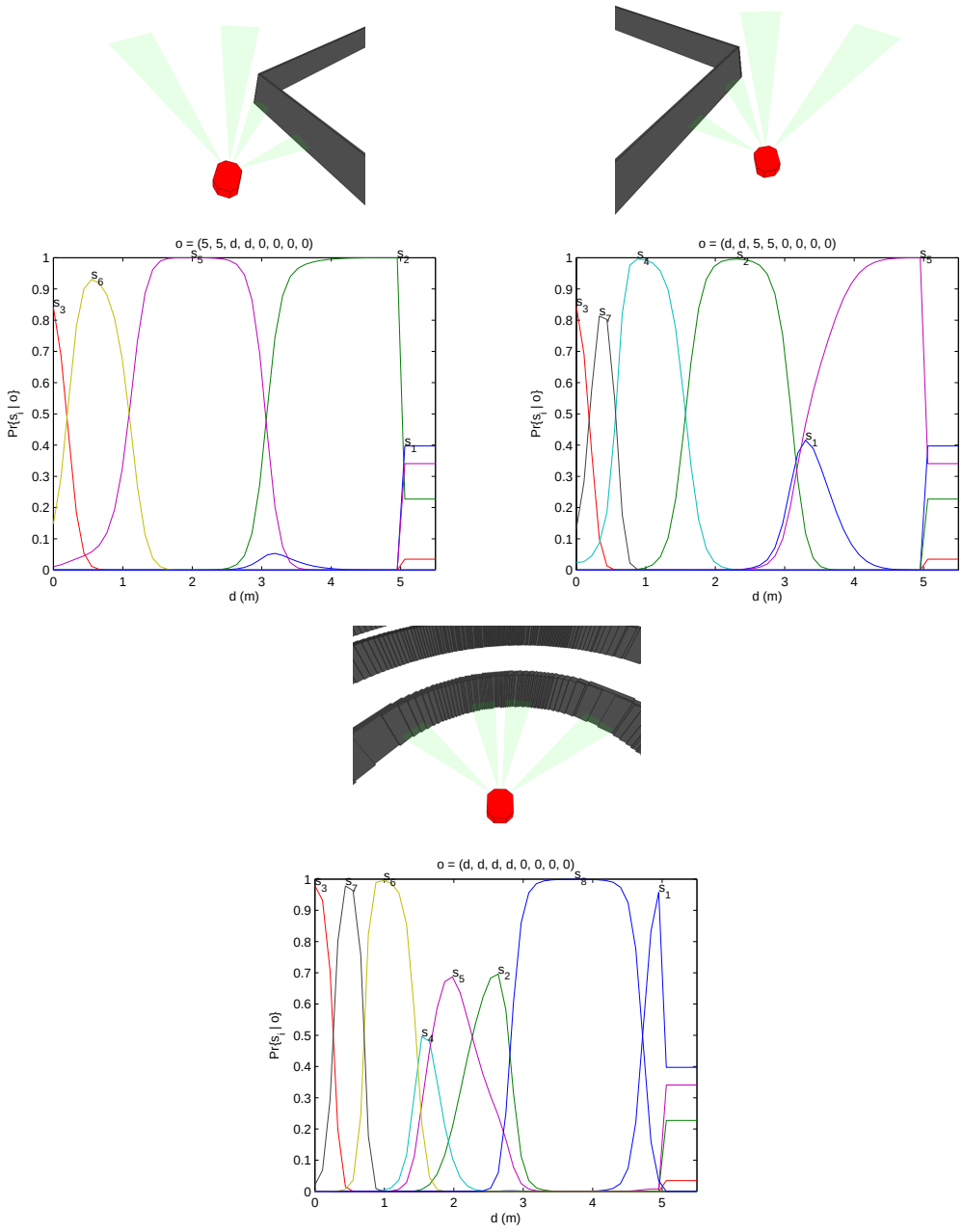


Fig. 7.23: Abordagem 2: Três situações encontradas pelo robô. Probabilidades a posteriori sobre os estados dada a observação. Cada estado é representado por uma cor.

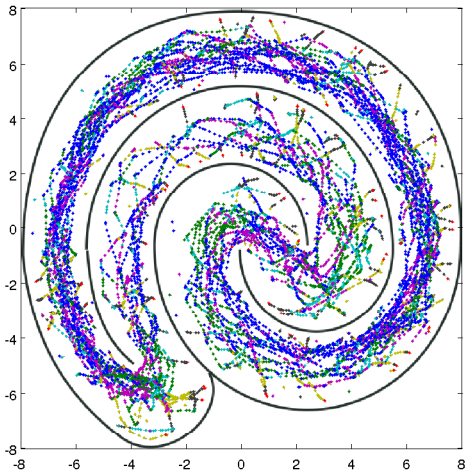


Fig. 7.24: Abordagem 2: As cores representam o estado mais provável estimado pelo filtro bayesiano ao longo da trajetória do robô. As cores de cada estado correspondem às usadas na Fig. 7.23.

diagonal) são muito superiores aos outros. Quanto maior esse valor, maior o tempo médio de permanência em um estado[2].

Constata-se então, pela tabela, que algumas ações mantêm o robô em um determinado estado por mais tempo que outras. Por exemplo, quando o robô está no estado s_5 (obstáculo à direita), tem-se: $a^1_{55} = 0.844$, $a^2_{55} = 0.813$ e $a^3 = 0.912$. Isso significa que ação 3 – virar à esquerda – mantém o robô por mais tempo nesse estado. A ação 2 – virar à direita, por sua vez, tende a levar o robô para o estado s_6 ($a^2_{56} = 0.107$), que representa um obstáculo mais acentuado à direita.

A situação simétrica ocorre no estado s_4, que corresponde à um obstáculo à esquerda. Tem-se que $a^2_{44} = 0.938$, e $a^3_{44} = 0.789$. Nesse caso, tomar a ação 2 – virar à direita – mantém o robô mais tempo nesse estado e tomar a ação 3 – virar à esquerda – tende a levar o robô ao estado s_7 ($a^3_{47} = 0.090$), que corresponde à um obstáculo mais acentuado à esquerda.

Recompensas recebidas

Após o treinamento do POMDP, ocorre o treinamento da política, através do gradiente de política. A Fig. 7.25 mostra o a evolução da recompensa média recebida ao longo do tempo para cada uma das 5 execuções, bem como a decomposição nos dois componentes da recompensa. Como na abordagem 1, apresentada anteriormente, houve uma rápida melhoria da recompensa nos primeiros 10 min. Após isso, ocorreu uma certa estagnação, apesar de ser possível notar, pelo quatro (b) da figura, que a

[2]O tempo de permanência τ_i num estado i é uma variável aleatória que, numa cadeia de Markov, segue distribuição geométrica com valor esperado $E\{\tau_i\} = \frac{1}{1-a_{ii}}$. Esse valor cresce monotonicamente com a_{ii}.

A_1	$\mathbf{s_t = 1}$	$\mathbf{s_t = 2}$	$\mathbf{s_t = 3}$	$\mathbf{s_t = 4}$	$\mathbf{s_t = 5}$	$\mathbf{s_t = 6}$	$\mathbf{s_t = 7}$	$\mathbf{s_t = 8}$
$\mathbf{s_{t+1} = 1}$	0.827	0.061	0.125	0.006	0.051	0.000	0.000	0.125
$\mathbf{s_{t+1} = 2}$	0.076	0.796	0.125	0.026	0.009	0.000	0.000	0.125
$\mathbf{s_{t+1} = 3}$	0.000	0.000	0.125	0.000	0.000	0.017	0.146	0.125
$\mathbf{s_{t+1} = 4}$	0.008	0.141	0.125	0.880	0.000	0.000	0.024	0.125
$\mathbf{s_{t+1} = 5}$	0.089	0.001	0.125	0.000	0.844	0.043	0.000	0.125
$\mathbf{s_{t+1} = 6}$	0.000	0.000	0.125	0.030	0.096	0.842	0.014	0.125
$\mathbf{s_{t+1} = 7}$	0.000	0.000	0.125	0.059	0.000	0.098	0.817	0.125
$\mathbf{s_{t+1} = 8}$	0.000	0.000	0.125	0.000	0.000	0.000	0.000	0.125

A_2	$\mathbf{s_t = 1}$	$\mathbf{s_t = 2}$	$\mathbf{s_t = 3}$	$\mathbf{s_t = 4}$	$\mathbf{s_t = 5}$	$\mathbf{s_t = 6}$	$\mathbf{s_t = 7}$	$\mathbf{s_t = 8}$
$\mathbf{s_{t+1} = 1}$	0.869	0.112	0.125	0.000	0.071	0.000	0.027	0.125
$\mathbf{s_{t+1} = 2}$	0.104	0.831	0.125	0.041	0.009	0.000	0.000	0.125
$\mathbf{s_{t+1} = 3}$	0.000	0.000	0.125	0.000	0.000	0.043	0.116	0.125
$\mathbf{s_{t+1} = 4}$	0.000	0.046	0.125	0.938	0.001	0.022	0.011	0.125
$\mathbf{s_{t+1} = 5}$	0.017	0.000	0.125	0.000	0.813	0.033	0.000	0.125
$\mathbf{s_{t+1} = 6}$	0.000	0.000	0.125	0.000	0.107	0.824	0.010	0.125
$\mathbf{s_{t+1} = 7}$	0.010	0.000	0.125	0.020	0.000	0.068	0.836	0.125
$\mathbf{s_{t+1} = 8}$	0.000	0.011	0.125	0.000	0.000	0.011	0.000	0.125

A_3	$\mathbf{s_t = 1}$	$\mathbf{s_t = 2}$	$\mathbf{s_t = 3}$	$\mathbf{s_t = 4}$	$\mathbf{s_t = 5}$	$\mathbf{s_t = 6}$	$\mathbf{s_t = 7}$	$\mathbf{s_t = 8}$
$\mathbf{s_{t+1} = 1}$	0.857	0.060	0.125	0.025	0.068	0.000	0.020	0.125
$\mathbf{s_{t+1} = 2}$	0.075	0.821	0.125	0.041	0.000	0.000	0.000	0.125
$\mathbf{s_{t+1} = 3}$	0.000	0.000	0.125	0.004	0.000	0.000	0.100	0.125
$\mathbf{s_{t+1} = 4}$	0.000	0.110	0.125	0.789	0.000	0.010	0.035	0.125
$\mathbf{s_{t+1} = 5}$	0.069	0.001	0.125	0.036	0.912	0.043	0.000	0.125
$\mathbf{s_{t+1} = 6}$	0.000	0.008	0.125	0.015	0.011	0.876	0.030	0.125
$\mathbf{s_{t+1} = 7}$	0.000	0.000	0.125	0.090	0.000	0.071	0.815	0.125
$\mathbf{s_{t+1} = 8}$	0.000	0.000	0.125	0.000	0.009	0.000	0.000	0.125

Tab. 7.1: Abordagem 2: As matrizes de transição para cada uma das ações. Cada célula contém o valor $\Pr\{s_{t+1} \mid s_t, a\}$ estimado pelo algoritmo Baum-Welch. Assim, cada coluna da matriz tem soma unitária.

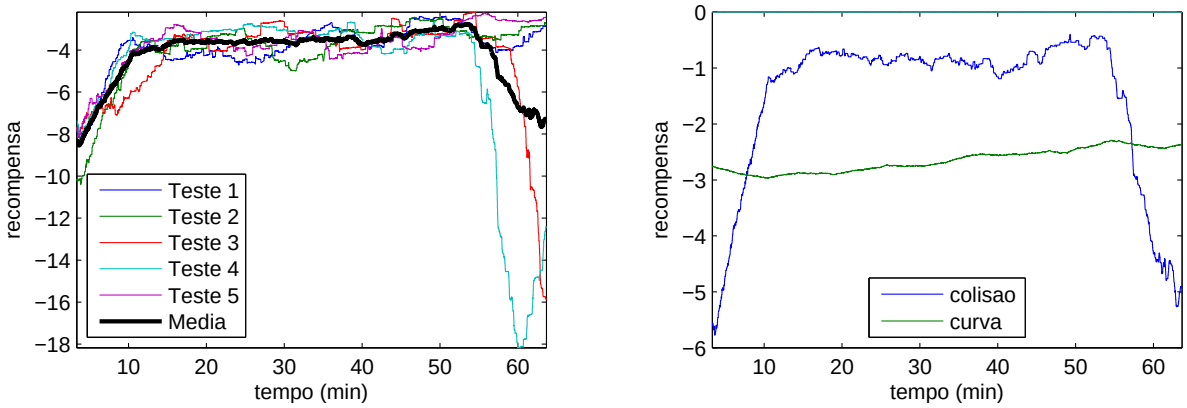


Fig. 7.25: Abordagem 2: Média móvel da recompensa imediata com *lag* de $6,6$ min (2k ciclos de controle). (a) Recompensa total de cada uma das execuções e média; (b) Recompensa média das 5 execuções, decomposta em cada um dos termos da eq. 7.5.

recompensa relacionada à andar em linha reta continua aumentando.

Primeiramente, percebe-se pela figura que, mesmo nos momentos de máxima recompensa média, não se atinge o patamar de recompensas que para a abordagem 1, sujeita ao mesmo ambiente. Isso pode significar que a redução da leitura sensorial em um número finito de estados pode retirar do algoritmo a capacidade de sintonia fina. Por outro lado, pode dar ao algoritmo uma maior capacidade de agir em problemas altamente não-lineares e também, como mencionado no início do capítulo cap. 5, pode permitir que o robô adquira não-reatividade. Por isso, este experimento, por poder ser resolvido de forma reativa, deve ser considerado apenas preliminar.

Em segundo lugar, percebe-se de forma gritante que nas repetições 3 e 4 do experimento, houve, ao final, uma queda brusca na performance relacionada a desvio de obstáculos. Não se detectou a causa exata dessa degradação, mas acredita-se que possa estar relacionada de certa forma a algum erro numérico ocasionado pela saturação do classificador logístico, ou seja, pelo crescimento acentuado dos parâmetros.

Trajetórias do robô

As trajetórias efetuadas pelo robô na execução 1, mostradas na Fig. 7.26, são muito similares às obtidas com a abordagem 1, porém, percebe-se que há uma instabilidade maior ao longo do tempo, com o robô ocasionalmente voltando a colidir com a parede.

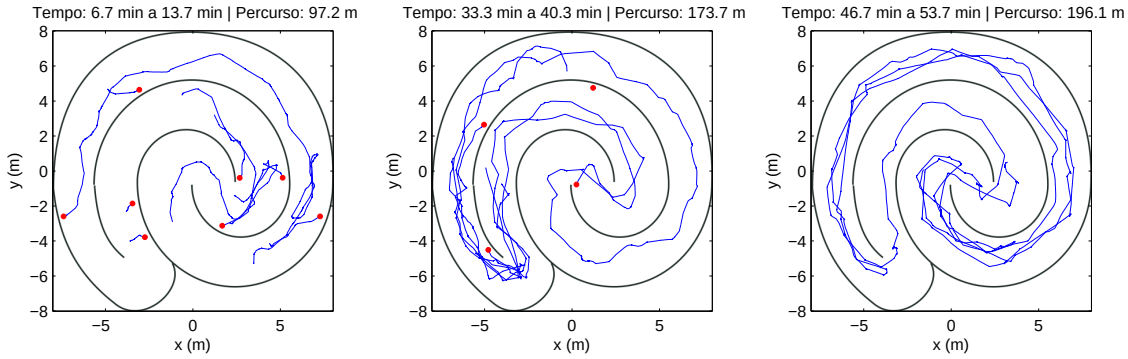


Fig. 7.26: Abordagem 2: Excertos de trajetórias ao longo da execução 1. Os círculos vermelhos são colisões.

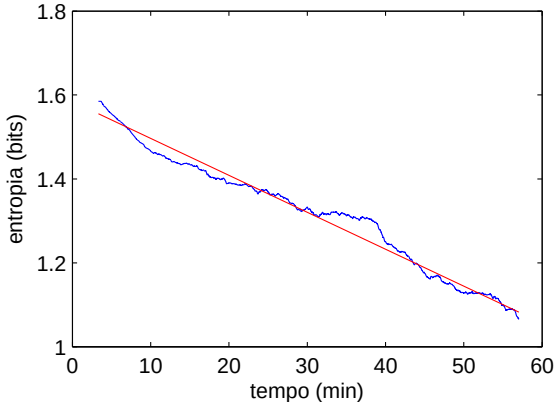


Fig. 7.27: Abordagem 2: Média móvel (*lag* de 6.6 min) da entropia $H(a_t|\mathbf{o} = o_t)$ na saída do controlador, média das 5 execuções. Em vermelho, regressão linear.

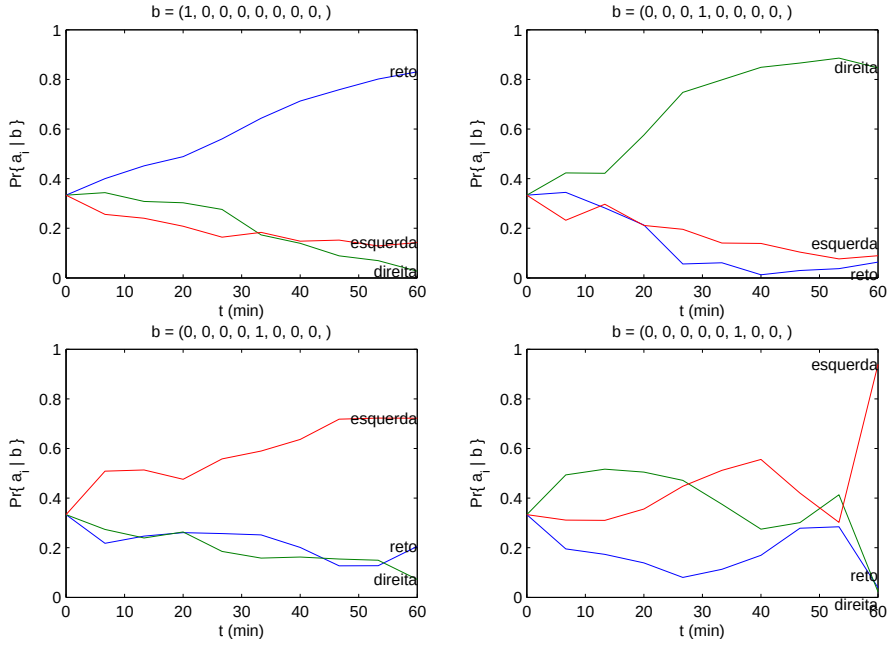


Fig. 7.28: Abordagem 2: Evolução da decisão do robô, para algumas crenças puras.

Variação de entropia

Como para os experimentos anteriores, estimada a variação de entropia ao longo do tempo, retratada na Fig. 7.27. O resultado da estimação para a variação média de entropia, para as 5 execuções do experimento, foi:

$$\bar{b} = -8,8 \times 10^{-3} \text{ bits/min.} \tag{7.22}$$

Estes valores se aproximam dos resultados obtidos para os outros experimentos.

Aprendizado da política

Resta, finalmente, apresentar os resultados do treinamento do algoritmo de gradiente de política. Apresentam-se, na Fig. 7.28 as probabilidade de ação $\pi_\theta(a_i|\mathbf{b}_t)$ ao longo do tempo, para algumas crenças $b_t(s)$ puras, ou seja, para crenças onde toda a probabilidade se concentra em um dos estados s_i do POMDP.

A figura mostra que, nas primeiras iterações do algoritmo, as probabilidades de ação são as mesmas para as três ações, independente da crença. Ao longo do tempo, começa a haver uma tendência de crescimento das probabilidades de uma das ações em relação a outra. Apenas para s_6 houve uma inversão de tendencia ao longo do tempo.

Pela figura, percebe-se que a política aprendida corresponde ao esperado para a tarefa de desvio de

obstáculos. Para a crença $\mathbf{b} = (1, 0, 0, 0, 0, 0, 0, 0)$, correspondente ao estado s_1, que, como discutido anteriormente, significa nenhum obstáculo à frente, a ação predominante ao longo do tempo é a_1 – seguir em frente. Para as crenças correspondentes ao estado s_4, que é ativo quando há obstáculo à esquerda, a ação predominante é virar à direita. O oposto ocorre para as crenças s_5 e s_6, que sinalizam obstáculo à direita.

7.3 Considerações sobre os experimentos

Este capítulo apresentou três experimentos nos quais a tarefa do robô era parecida. O que mudou foi a complexidade do ambiente (entre o experimento 1 e o 2) e a metodologia aplicada (entre o experimento 1 e o 3). Tecem-se aqui algumas comparações entre o desempenho do robô nos diferentes experimentos.

Claramente o desempenho do robô foi superior no experimento 1. Uma parametrização linear dos sensores, ligada diretamente à entrada do algoritmo de aprendizado, foi o suficiente para fazer com que o robô desviasse de obstáculos. O algoritmo de gradiente natural de política, baseado em uma técnica que garante convergência, além do fato de se utilizar estimações semi-offline do gradiente, certamente contribuíram para o sucesso do experimento. Tentativas efetuadas com métodos mais clássicos de parametrização não convergiam ou simplesmente eram incapazes de aprender, devido à grande aleatoriedade do sistema.

Ainda com a ligação direta entre a leitura sensorial e o algoritmo de gradiente de política, foi possível obter bons resultados em 4 dos 5 testes efetuados no segundo experimento. A convergência foi um pouco mais lenta, mas é justificada pela maior dificuldade imposta pelo problema: os alvos verde e laranja são altamente estocásticos, por mudarem constantemente de lugar, saírem e voltarem ao campo de visão do robô por uma simples rotação, e por não estarem sempre presentes. Assim, a variância no aprendizado é maior.

A brusca piora de desempenho ocorrida num dos testes do experimento 2 foi causada por uma falha na arquitetura da função-recompensa. Este resultado negativo aponta para uma futura melhoria no algoritmo, para assegurar que não haja esquecimento do que já havia sido aprendido.

Finalmente, o último experimento, além do gradiente de política, aplica todo o arcabouço teórico desenvolvido sobre POMDPs. Apesar de um resultado um pouco pior que no experimento 1, o experimento serve como um teste preliminar de viabilidade da abordagem. Em troca dessa pequena queda de performance, seria possível obter vantagem em experimentos não-reativos e altamente não-lineares, onde uma simples combinação linear das leituras do sensor não poderia dar conta das demandas da aplicação.

Apesar de preliminar, o experimento 3 permite compreender as vantagens e deficiências do POMDP

e dos métodos utilizados para treiná-lo. Com base nos resultados, são propostas, na conclusão, melhorias e sugestões de trabalhos futuros.

Capítulo 8

Conclusão

A importância do aprendizado em sistemas artificiais tem crescido bastante, à medida que aumenta o dinamismo da vida moderna. Este trabalho procurou oferecer uma contribuição no sentido de dotar sistemas de controle da capacidade de se adaptarem e aprenderem características do ambiente e do robô, a fim de realizar tarefas de navegação autônoma.

Inicialmente, foi apresentado um modelo de rede imunológica classificadora usada como controlador para resolver o problema de navegação em robótica autônoma. Posteriormente, foram apresentados os processos de Decisão de Markov Parcialmente Observáveis (POMDP) e um paralelo foi tecido entre redes imunológicas e POMDPs. Finalmente, foram apresentados métodos de aprendizado em POMDP e diversos métodos de aprendizado por reforço. Tais métodos foram aplicados no problema da navegação em robótica autônoma.

Uma das principais motivações ao longo do trabalho, diante das inúmeras propostas *ad-hoc* que surgem a todo momento para problemas como o de aprendizado de máquina, foi buscar uma justificativa o mais formal possível para cada etapa do trabalho. Buscou-se amparo então na teoria de probabilidades, e em alguns modelos de processos estocásticos.

Em um primeiro momento, a busca de uma justificativa matemática mais forte pode ter reduzido o alcance dos resultados experimentais, devido ao esforço despendido na compreensão dos conceitos. Acredita-se, porém, que a abordagem mais formal dotou o trabalho de uma maior perspectiva futura, por utilizar uma linguagem comum a áreas diversas como matemática, aprendizado de máquina, teoria de controle e filtragem e pesquisa operacional.

Os primeiros capítulos deste trabalho (capítulos 2 a 5) apresentaram principalmente teorias e métodos encontrados na literatura. A partir do final do capítulo 5, com o paralelo entre sistemas classificadores e POMDPs, iniciou-se a exposição das contribuições do trabalho. No capítulo 6, apresentaram-se as parametrizações e algoritmos idealizados especificamente para o problema em questão e, finalmente, no capítulo 7, tais algoritmos foram aplicados para resolver problemas de

navegação robótica.

Como contribuições principais do trabalho, destacam-se:

- uma visão geral, no capítulo 4, a respeito de algoritmos de aprendizado por reforço, incluindo os recentemente propostos algoritmos de gradiente natural de política;
- um estudo geral sobre POMDPs, no capítulo 5, focando principalmente nos aspectos de filtragem e de aprendizado de parâmetros;
- estabelecimento de um paralelo, na seção 5.4.1, entre redes imunológicas artificiais e POMDPs. Tal paralelo permite a aplicação de algoritmos probabilísticos em domínios onde as redes imunológicas artificiais são utilizadas e, em contrapartida, permite a aplicação de algoritmos de síntese de redes imunológicas artificiais na síntese de POMDPs;
- proposta de uma parametrização, na seção 6.2, para a distribuição condicional de observação do POMDP, voltada para espaços contínuos de observação;
- especificação detalhada, na seção 6.4, de um algoritmo de gradiente natural de política, com cálculo semi-offline do gradiente;
- finalmente, no capítulo 7, realização de experimentos simulados, com problemas de múltiplos objetivos, e comparação entre duas diferentes parametrizações na entrada do algoritmo de aprendizado.

Os resultados obtidos nos experimentos permitem direcionar trabalhos futuros:

- o último experimento efetuado mostra que as mais importantes melhorias a serem efetuadas dizem respeito ao treinamento do POMDP. É necessário automatizar a escolha do número n de estados. Uma ideia é iniciar com um conjunto de estados pequeno e incorporar novos estados enquanto houver melhoria de performance, como faz Nikovski (2002);
- É necessário também encontrar uma forma de melhorar o modelo do POMDP de forma *online*, à medida que a política do agente muda, e ao mesmo tempo evitar esquecer de estados antigos que não são mais atingidos mas que devem continuar sendo evitados. Métodos bayesianos variacionais (Robert, 2007) são uma possível forma de treinar modelos probabilísticos de forma *online*;
- O fato de que os POMDPs são baseados em modelos de Markov impõe uma forte restrição sobre o modelo do sistema que pode ser representado pelas matrizes de transição. Alternativas que poderiam trazer resultados seriam a) o uso de parametrizações mais sofisticadas nas matrizes de

transição e uma possível dependência da transição em relação ao vetor observação; b) utilização de modelos semi-markovianos (Lipsky, 2008, capítulo 8), onde modela-se, além da matriz de transição, a distribuição sobre os tempos de transição; c) redes bayesianas dinâmicas mais gerais, nas quais é possível haver fatorização de estados, evitando a explosão combinatorial;

- Seria interessante utilizar a recompensa no treino do POMDP. Assim, a separação entre estados seria feita prioritariamente pela diferença de recompensa a que uma observação ou outra incorre a longo prazo. Através da regra da cadeia, seria possível estender o algoritmo de gradiente de política para alterar também os parâmetros do POMDP;

- É necessário procurar alguma correção para o problema ocorrido no experimento 2, no qual uma trajetória especialmente polarizada do robô (recompensas artificialmente muito elevadas) levou a uma piora posterior de desempenho;

- Finalmente, nesse trabalho foi realizado um acoplamento entre os módulos POMDP e de aprendizagem por reforço. Pode-se enquadrar tal arquitetura no contexto da crescente área de pesquisa chamada *deep machine learning* (Arel et al., 2010), que trata do acoplamento hierárquico de métodos de aprendizagem de máquina e cujo objetivo final é modelar funcionalmente o neocórtex. Como apresentado na seção 5.4.2, um dos caminhos para tanto utiliza os conceitos de redes bayesianas, das quais os POMDPs são representantes.

Bibliografia

D. Aberdeen. *Policy-gradient algorithms for partially observable Markov decision processes*. PhD thesis, Australian National University, 2003.

J.S. Albus. Outline for a theory of intelligence. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(3):473–509, 2002.

J.S. Albus. A model of computation and representation in the brain. *Information Sciences*, 180(9): 1519–1554, 2010.

S. Amari & H. Nagaoka. *Methods of information geometry*. Amer Mathematical Society, 2007.

S.I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

P.J. Antsaklis & K.M. Passino. *An introduction to intelligent and autonomous control*. Kluwer Academic Publishers, 1993.

I. Arel, D.C. Rose & T.P. Karnowski. Deep Machine Learning: A New Frontier in Artificial Intelligence Research. *Computational Intelligence Magazine, IEEE*, 5(4):13–18, 2010.

R.C. Arkin. *Behavior-based robotics*. The MIT Press, 1998.

J.A. Bagnell & J. Schneider. Covariant policy search. In *International Joint Conference on Artificial Intelligence*, volume 18, pp. 1019–1024. Citeseer, 2003.

L. Baird & A.H. Klopf. Reinforcement learning with high-dimensional continuous actions. *US Air Force Technical Report WL-TR-93-1147, Wright Laboratory, Wright-Patterson Air Force Base, OH*, 1993.

B. Bakker, V. Zhumatiy, G. Gruener & J. Schmidhuber. Quasi-online reinforcement learning for robots. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2997–3002. IEEE, 2006.

L.E. Baum, T. Petrie, G. Soules & N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, 41(1): 164–171, 1970.

J. Baxter, P.L. Bartlett & L. Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15(1):351–381, 2001.

R.E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

Y. Bengio & P. Frasconi. An input output HMM architecture. In *Advances in neural information processing systems*, pp. 427–434. Citeseer, 1995.

D.P. Bertsekas. Dynamic programming and optimal control, vol. II. 2007.

D.P. Bertsekas & J.N. Tsitsiklis. *Neuro-dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

S. Bhatnagar, R. Sutton, M. Ghavamzadeh & M. Lee. Incremental natural actor-critic algorithms. *Advances in Neural Information Processing Systems*, 20:105–112, 2008.

R. Brooks. A robust layered control system for a mobile robot. *IEEE journal of robotics and automation*, 2(1):14–23, 1986.

R.R. Cazangi. *Uma Proposta Evolutiva para Controle Inteligente em Navegação Autônoma de Robôs*. Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 2004.

R.R. Cazangi. *Síntese de Controladores Autônomos em Robótica Móvel por meio de Computação Bio-inspirada*. Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 2008.

E. Charniak. *Statistical language learning*. The MIT Press, 1996.

S. Chikkerur, T. Serre & T. Poggio. A Bayesian inference theory of attention: neuroscience and algorithms. 2009.

L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 183–183. Citeseer, 1992.

L.N. de Castro & J. Timmis. *Artificial immune systems: a new computational intelligence approach*. Springer Verlag, 2002.

T. Dean. A computational model of the cerebral cortex. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 20, p. 938. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

A.P. Dempster, N.M. Laird, D.B. Rubin et al. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

D. Ernst, P. Geurts & L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(1):503, 2006.

D. George & J. Hawkins. A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 3, pp. 1812–1817. IEEE, 2005.

P.W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

R.V. Gulick. Consciousness. The Stanford Encyclopedia of Philosophy. 2004. URL `http://plato.stanford.edu/entries/consciousness/`.

P. Gärdenfors. *Conceptual spaces: The geometry of thought*. The MIT Press, 2004.

J. Hawkins & S. Blakeslee. *On intelligence*. Owl Books, 2005.

J.H. Holland. *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press, 1975.

J.H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell, editores, *Machine learning: An artificial intelligence approach (Vol. 2)*. Morgan Kaufmann, Los Altos, CA, 1986.

N.K. Jerne. Towards a network theory of the immune system. *Ann. Immunol*, 125C:373–389, 1974.

L.P. Kaelbling, M.L. Littman & A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

S. Kakade. Optimizing average reward using discounted rewards. In *Computational Learning Theory*, pp. 605–615. Springer, 2001.

R.T. Kellogg. *Fundamentals of cognitive psychology*. Sage Publications, Inc, 2007.

L.J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, 1992.

L.R. Lipsky. *Queuing theory: A linear algebraic approach*. Springer Verlag, 2008.

M.L. Littman, A.R. Cassandra & L.P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

M.L. Littman, R.S. Sutton & S. Singh. Predictive representations of state. *Advances in neural information processing systems*, 2:1555–1562, 2002.

W.S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

E.A. Ludvig, R.S. Sutton & E.J. Kehoe. Stimulus representation and the timing of reward-prediction errors in models of the dopamine system. *Neural computation*, 20(12):3034–3054, 2008.

P. Maes. Situated agents can have goals. *Robotics and autonomous systems*, 6(1-2):49–70, 1990.

S. Mahadevan, G. Theocharous & L.P. Kaelbling. Spatial and Temporal Abstractions in POMDPs Applied to Robot Navigation, 2005.

N. Metropolis. The beginning of the Monte Carlo method. *Los Alamos Science*, 15:125–130, 1987.

M. Minsky. *The society of mind*. Simon and Schuster, 1988.

G.E. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

D.S. Moore, G.P. McCabe & B.A. Craig. *Introduction to the practice of statistics*. Freeman, 6 ed., 2009.

T. Morimura, E. Uchibe & K. Doya. Utilizing the natural gradient in temporal difference reinforcement learning with eligibility traces. In *Proc. of the 2nd International Symposium on Information Geometry and its Application*, pp. 256–263, 2005.

V.B. Mountcastle. An organizing principle for cerebral function: the unit module and the distributed system. *The mindful brain*, pp. 7–50, 1978.

T. Nagel. What is it like to be a bat? *The Philosophical Review*, 83(4):435–450, 1974.

D. Nikovski. *State-aggregation algorithms for learning probabilistic models for robot control*. PhD thesis, Carnegie Mellon University, 2002.

C.H. Papadimitriou & J.N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

J. Peters & S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008a.

J. Peters & S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008b.

J. Peters, S. Vijayakumar & S. Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the Third IEEE-RAS International Conference on Humanoid Robots*, pp. 1–20. Citeseer, 2003.

R. Pfeifer & C. Scheier. *Understanding intelligence*. The MIT Press, 1999.

W.B. Powell. *Approximate dynamic programming: solving the curses of dimensionality*. Wiley, 2007.

L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

C.P. Robert. *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer Verlag, 2007.

S.J. Russell & P. Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2 ed., 2003.

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

N. Sharkey. The programmable robot of ancient Greece. *New Scientist*, 2611:32–35, 2007.

N. Sharkey & A. Sharkey. Electro-mechanical robots before the computer. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 223(1): 235–241, 2009.

H. Shatkay & L.P. Kaelbling. Learning topological maps with weak local odometric information. In *International Joint Conference on Artificial Intelligence*, volume 15, pp. 920–929. Citeseer, 1997.

R. Siegwart & I.R. Nourbakhsh. *Introduction to autonomous mobile robots*. The MIT Press, 2004.

S. Singh, T. Jaakkola & M.I. Jordan. Reinforcement learning with soft state aggregation. *Advances in neural information processing systems*, pp. 361–368, 1995.

W.D. Smart, P. Kaelbling et al. Effective reinforcement learning for mobile robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pp. 3404–3410. IEEE, 2002.

R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1): 9–44, 1988.

R.S. Sutton & A.G. Barto. *Reinforcement learning*. MIT Press, 1998.

R.S. Sutton, D. McAllester, S. Singh & Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12(22), 2000.

G. Tesauro. Practical issues in temporal difference learning. *Machine learning*, 8(3):257–277, 1992.

P. Thagard. Cognitive science. The Stanford Encyclopedia of Philosophy. 2010. URL `http://plato.stanford.edu/entries/cognitive-science/`.

G. Theocharous, K. Rohanimanesh & S. Mahadevan. Learning hierarchical partially observable Markov decision process models for robot navigation. 2001.

G. Theocharous, K. Murphy & L.P. Kaelbling. Representing hierarchical pomdps as dbns for multi-scale robot localization. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pp. 1045–1051. IEEE, 2004.

S. Thrun, W. Burgard & D. Fox. *Probabilistic Robotics*. The MIT Press, 2006.

R. Vaughan. Massively multi-robot simulation in Stage. *Swarm Intelligence*, 2:189–208, 2008.

C.J.C.H. Watkins & P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

N. Wiener. *Cybernetics*. J. Wiley, 1948.

R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.