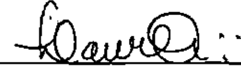
SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 24.04.2000

Assinatura: 

Explorando o Potencial de Algoritmos de Aprendizado com Reforço em Robôs Móveis

Gedson Faria

Orientadora: ***Profa. Dra. Roseli Ap. Francelin Romero***

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Área: Ciências de Computação e Matemática Computacional.

USP – São Carlos
Abril de 2000

Este documento foi preparado com o formatador de textos LaTeX. O sistema de citações de referências bibliográficas utiliza o padrão *Apalike* do sistema bibTeX.

Agradecimentos

Nestes dois anos de trabalho na USP, estive longe de casa, mas sempre pude ter ao meu lado pessoas em que pude confiar. Gostaria de agradecer principalmente:

a meus pais e minha irmã, Kélita, pelo apoio e incentivo que sempre me deram;

à profª. Roseli A. F. Romero pela dedicação com que me orientou;

aos meus amigos: Walter, Cláudia, André, Vanessa, Ivone, Danielle, Enzo, Will, Claudinho, Renato e Daniela, por estarem presentes em todas as horas, principalmente quando precisei;

aos companheiros "labiquianos" que sempre estiveram dispostos a ajudar;

a todos os professores e funcionários do ICMC que de algum modo colaboraram com os meus estudos realizados neste instituto;

à CAPES pelo apoio financeiro fornecido durante parte do desenvolvimento deste trabalho.

Sumário

1 Introdução 1

2 Aprendizado com Reforço 5

2.1 Modelo Padrão de Aprendizado com Reforço 6

2.2 Processo de Decisão de Markov 8

2.3 Função de Custo Ótimo 10

2.4 Programação Dinâmica 12

2.4.1 Teoremas Básicos de Programação Dinâmica 13

2.4.2 Técnicas Básicas de Programação Dinâmica 14

2.5 Métodos de Monte Carlo 15

2.6 Métodos de Diferença Temporal 16

2.7 Algoritmos de Aprendizado com Reforço 19

2.7.1 Algoritmos Q-*learning* e SARSA 19

2.7.2 R-*learning* 22

2.7.3 H-*learning* 23

3 Robô e Software de Controle **27**

4 Experimentos com os Algoritmos: Q-*learning*, R-*learning* e H-*learning* **31**

4.1 Modelagem da Tarefa de Navegação 32

4.2 Modelo Incorporando Conceitos de Lógica *Fuzzy* 42

5 Aplicação utilizando R'-*learning* **47**

5.1 Cálculo da Força de Repulsão 49

5.2 Reconhece Objetos e *Planning* 51

5.3 Evitando Colisões 55

5.3.1 Colisão 55

5.3.2 Risco de Colisão 56

5.3.3 Obstáculo 56

5.4 Modelo MDP 59

5.5 Resultados Obtidos 60

6 Discussão e Conclusões **61**

Lista de Figuras

2.1 Modelo padrão de aprendizado com reforço. 7

2.2 Diagrama de transição de estados. 10

3.1 Pioneer 1 Gripper, (a) vista lateral, (b) vista superior 28

3.2 Retângulo de atuação para a função *sfOccBox* 29

3.3 Retângulo de atuação para a função *sfOccPlane* 30

4.1 Mapeamento da leitura dos sonares em um estado do ambiente. 33

4.2 Q-*learning* com $\alpha = 0.2$, $\gamma = 0.50$ e $\epsilon = 100\%$ 35

4.3 Q-*learning* com $\alpha = 0.2$, $\gamma = 0.99$ e $\epsilon = 100\%$ 35

4.4 R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 100\%$ 35

4.5 Q-*learning* com $\alpha = 0.2$, $\gamma = 0.50$ e $\epsilon = 95\%$ 36

4.6 Q-*learning* com $\alpha = 0.2$, $\gamma = 0.99$ e $\epsilon = 95\%$ 36

4.7 R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 95\%$ 36

4.8 Q-*learning* com $\alpha = 0.2$, $\gamma = 0.75$ e $\epsilon = 100\%$ 37

4.9 R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 100\%$ 37

4.10 H-*learning* com $\epsilon = 100\%$ 37

4.11 Q-*learning* com $\alpha = 0.2$, $\gamma = 0.75$ e $\epsilon = 95\%$ 38

4.12 R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 95\%$ 38

4.13 H-*learning* com $\epsilon = 95\%$ 38

4.14 Estado do Ambiente mapeado através da simplificação das leituras dos setes sonares em três sinais de entrada, diminuindo sensivelmente o número de estados. 39

4.15 Q-*learning* com $\alpha = 0.2$, $\gamma = 0.99$ e $\epsilon = 100\%$ 40

4.16 Q-*learning* com $\alpha = 0.2$, $\gamma = 0.75$ e $\epsilon = 100\%$ 40

4.17 R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 100\%$ 40

4.18 H-*learning* com $\epsilon = 100\%$ 40

4.19 Q-*learning* $\times$ R-*learning*: colisões durante o aprendizado. 41

4.20 Classificação da leitura dos sonares em quatro funções *fuzzy*. 43

4.21 Definição do estado do ambiente incorporando lógica *fuzzy*. 44

4.22 R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 95\%$ 46

4.23 R'-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 95\%$ 46

4.24 R-*learning* $\times$ R'-*learning*: colisões durante o aprendizado. 46

5.1 Representação dos módulos da aplicação utilizando aprendizado com reforço. 48

5.2 Reconhecimento de objetos: (a) objeto estreito, (b) objeto extenso e (c) objeto estreito pronto para ser coletado. 52

5.3 Transformação da força de atração para o sistema LPS. 53

5.4 Escolha do valor de ϕ de forma que R fique paralelo à parede. A direção de R é dada pelo ângulo constante ϕ e pela direção de Fwr. (a) representa as leituras realizadas pelo robô Pioneer 1. (b) é uma representação de uma leitura ideal que marca todos os pontos da parede. 55

5.5 Estados do ambiente mapeados através da força de repulsão. 57

5.6 Funções de Pertinência: $\mu(\theta_{F_r})$ 58

5.7 R'-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 100\%$ 60

5.8 Planta do ambiente onde o robô recolheu objetos e aprendeu a navegar. . . 60

Lista de Tabelas

2.1 Transição de estados. 10

4.1 Reforço imediato definindo a tarefa de navegação. 32

4.2 Classificação das leituras priorizando objetos próximos. 34

4.3 Classificação das leituras priorizando objetos próximos em cinco classes. . . 39

4.4 Recompensas para os algoritmos R-*learning* e R'-*learning*. 45

5.1 Recompensas para o robô pegador de objetos. 59

Lista de Algoritmos

2.1 TD(λ) 18

2.2 Q-*learning* 21

2.3 SARSA 21

2.4 R-*learning* 23

2.5 H-*learning* 25

4.1 R'-*learning* 45

Resumo

O problema de aprendizado com robôs é essencialmente fazer com que o robô execute tarefas sem a necessidade de programá-los explicitamente. Nos últimos anos, Aprendizado de Máquina, um subcampo de Inteligência Artificial, tem procurado substituir programação explicita pelo processo de ensinar uma tarefa. O Aprendizado com Reforço é um dos paradigmas do aprendizado não-supervisionado, podendo ser visto como uma forma de ensinar o robô a realizar uma tarefa sem especificar previamente como realizá-la. O problema de aprendizado com reforço pode ser modelado como: um conjunto de estados do ambiente, um conjunto de ações e um conjunto de recompensas. Neste trabalho explora-se o potencial dos principais algoritmos de aprendizado com reforço: Q-*learning*, R-*learning* e H-*learning*. Desta forma, foram comparados métodos "independentes de modelo" e "baseados em modelo", verificando a eficiência de cada algoritmo para a tarefa de navegação em um ambiente dinâmico contendo obstáculos. Além disso, este trabalho propõe um método de navegação baseado em sensores, chamado R'-*learning*, o qual incorpora conceitos de lógica *fuzzy* ao algoritmo R-*learning* para a navegação de robôs móveis em ambientes desconhecidos. Foi realizada uma aplicação que consiste em ensinar o robô a encontrar pequenos objetos. Para isto, um conjunto de estados foi mapeado através de conceitos de força de repulsão e para navegação foi utilizado o algoritmo R'-*learning*. O robô mostrou ter um comportamento satisfatório ao realizar esta tarefa.

Abstract

The problem of robot learning is essentially one of getting robots to do tasks without the need for explicitly programming them. Machine learning is a sub-area of artificial intelligence (AI), whose ultimate goal is to replace explicit programming by *teaching*. Reinforcement Learning (RL) is an unsupervised learning paradigm and could be seen as a way of programming agents by reward and punishment without specify *how* the task is to be achieved. Formally, the RL model consists of a discrete set of environment states, a discrete set of agent actions and a set of scalar reinforcement signals. In this work, the performance of the most important reinforcement learning algorithms: Q-learning, R-learning, H-learning is investigated. In this way, model-free and model-based are compared, to show the efficiency of each algorithm in the navigation task avoiding obstacles. Furthermore, this work proposes a sensor-based navigation method, called R'-learning, which incorporates fuzzy logic into the R-learning algorithm for navigation of mobile robots in uncertain environment. An application consisting of teaching the robots to find small objects in a corridor is realized. For this, a state set mapping is done through force field concepts and for the navigation R'-leaning algorithm has been used. The robot showed to have behavior satisfactory in the performing this task.

Capítulo 1

Introdução

O problema de aprendizado com robôs é essencialmente fazer com que o robô execute tarefas sem a necessidade de programá-los explicitamente. A programação de robôs é uma tarefa extremamente desafiadora, por muitas razões. O sensor de um robô, como por exemplo o sonar, tem comportamentos imprevisíveis, algumas vezes variando conforme o ambiente. Sendo assim, não basta apenas conhecer o funcionamento dos sensores, alguém terá que fornecer um modelo de trabalho para cada ambiente em particular. Para programar um robô, a tarefa deve ser decomposta em uma sucessão de operações de baixo nível, como por exemplo, movimentar as rodas ou mover as articulações em um dado ângulo. Não há uma linguagem de programação de alto nível, para robôs, em que se possa confiar. Por estas razões, há um interesse considerável em que os robôs possam aprender a realizar tarefas automaticamente.

Aprendizado de máquina é um subcampo de Inteligência Artificial (IA) o qual, nos últimos anos, tem procurado substituir programação explícita pelo processo de ensinar uma tarefa. Pesquisas em aprendizado de máquina tem estudado muitos tipos diferentes de aprendizado [Mitchell, 1997]. Há dois tipos de aprendizado: supervisionado e não-supervisionado.

O aprendizado supervisionado foi implementado pelo Perceptron [Rosenblatt, 1963]

e Adaline [Widrow and Hoff, 1960]. Métodos com aprendizado supervisionado, também chamados de métodos da Correção do Erro, requerem um conjunto de treinamento constituído de pares de vetores de entrada e saída. Baseado no conhecimento da resposta correta, um sinal correspondente ao erro, dado pela diferença entre a resposta correta e a fornecida pela rede, é retornado ao elemento adaptativo: o elemento faz uso deste sinal para adaptar os valores de seus pesos. Esta seqüência é repetida para todos os pares do conjunto de treinamento até que os sinais correspondentes aos erros aproximem-se de zero.

O aprendizado não-supervisionado interage com o ambiente, fazendo com que o aprendiz ganhe uma pequena ou nenhuma realimentação da tarefa de aprendizagem.

O aprendizado com reforço é um dos paradigmas do aprendizado não-supervionado, podendo ser visto como uma forma de ensinar o robô a realizar uma tarefa, sem especificar, previamente, como realizá-la. O problema de aprendizado com reforço pode ser modelado como: um conjunto de estados do ambiente, um conjunto de ações e um conjunto de recompensas. Para cada ação realizada em um estado do ambiente o aprendiz recebe uma recompensa. O aprendiz (o robô) não sabe qual é a melhor ação a tomar, como em muitas formas de aprendizado de máquina, por isso ele deve descobrir, através de tentativas, quais ações lhe rendem maior recompensa.

Outros problemas de aprendizado de robôs como ambiente estocástico[1], resposta em tempo real e aprendizado *on-line*, também são resolvidos utilizando aprendizado com reforço.

Aprendizado por Reforço (RL) é conhecido por servir como uma ferramenta teórica para estudar os princípios de aprendizado de comportamento. Mas também foi usado por vários pesquisadores como uma ferramenta de prática computacional por construir sistemas autônomos que evoluem a partir de suas experiências. Estas aplicações variaram de robótica, produção industrial a problemas de busca combinatória, como em um jogo de

[1]Ambiente estocástico, também conhecido como ambiente não-determinístico, é um ambiente no qual uma mesma ação realizada em um mesmo estado do ambiente, em tempos distintos, pode retornar diferentes estados seguintes e/ou recompensas diferentes.

computador. Littman [Littman et al., 1994] utilizou os algoritmos TD e Q-*learning* para o Jogo de Damas e Thrun [Thrun, 1995] utilizou aprendizado de comportamentos no jogo de Xadrez. Métodos de RL foram utilizados por vários pesquisadores para o controle e navegação de robôs, podendo ser encontrados em [Mataric, 1994], [Schaal and Atkeson, 1994], [Mahadevan and Connell, 1991], [Crites and Barto, 1996] e [Bagnell et al., 1998].

Existem na literatura, experimentos de aprendizado sobre ambientes simulados e perfeitos, nos quais nunca ocorrem erros na execução das ações e nem na percepção dos estados do ambiente. Contudo, esses ambientes não fornecem um modelo consistente para aplicações de navegação em tempo-real. Em outros trabalhos é considerado o ambiente real, mas um mapa do ambiente é fornecido previamente ou construído durante a exploração do robô. Esta abordagem limita a navegação ao mapa do ambiente, pois as regras de navegação para o robô serão criadas de acordo com este mapa. Neste trabalho, o ambiente do robô será considerado como desconhecido e não será mantido nenhum mapa do ambiente para o aprendizado de navegação. Desta forma, a política de controle do robô servirá para qualquer tipo de ambiente.

Este trabalho tem por objetivo explorar o potencial de algoritmos de aprendizado com reforço, em aplicações de tempo real, para o robô *Pioneer 1 Gripper*, em ambientes estocásticos e desconhecidos.

Este trabalho inicia apresentando, no Capítulo 2, uma revisão sobre Aprendizado com Reforço, a forma de modelagem do problema de aprendizado, o modelo MDP e funções de custo utilizadas para prever as melhores ações. Serão também apresentados, no Capítulo 2, métodos de Programação Dinâmica, métodos de Monte Carlo, métodos de Diferença Temporais e os algoritmos de aprendizado Q-*learning* [Watkins, 1989], R-*learning* [Schwartz, 1993] e H-*learning* [Tadepalli and Ok, 1994]. As propriedades do robô *Pioneer 1 Gripper* e o seu *software* de controle serão apresentados no Capítulo 3. Em seguida, a implementação do modelo MDP, para a tarefa de navegação, assim como as comparações de desempenho entre os algoritmos Q-*learning*, R-*learning* e H-*learning*, serão detalhados no Capítulo 4. O teste de desempenho de um novo algoritmo, R'-*learning*

nós proposto, que incorpora conceitos de lógica *fuzzy* [Shaw and Simões, 1999], também são descritos no Capítulo 4. No Capítulo 5, será detalhada a proposta de uma aplicação real para o robô *Pioneer 1 Gripper*. Esta aplicação consiste em fazer com que o robô navegue por um ambiente desconhecido, procurando por pequenos objetos e levando-os para uma "lixeira". Para finalizar, será apresentada no Capítulo 6, uma discussão sobre os três algoritmos de aprendizado e uma conclusão sobre a proposta deste trabalho.

Capítulo 2

Aprendizado com Reforço

Aprendizado com reforço (RL) é um problema no qual um agente deve aprender comportamentos através de iterações de tentativa-e-erro em um ambiente dinâmico. RL não é definido pela caracterização de algoritmos de aprendizagem, mas por caracterizar uma classe de problemas de aprendizagem. Todo o algoritmo que resolver bem esse problema será considerado um algoritmo de aprendizado com reforço.

Aprendizado com reforço é baseado na idéia que, se uma ação é seguida de estados satisfatórios, ou por uma melhoria no estado, então a tendência para produzir esta ação é aumentada, isto é, reforçada. Estendendo esta idéia, ações podem ser selecionadas em função da informação sobre os estados que elas podem produzir, o que introduz aspectos de controle com realimentação.

Atualmente, quase todos métodos de RL em uso são baseados na técnica de Diferenças Temporais (TD) [Sutton, 1988]. A idéia fundamental em TD é o aprendizado por predição: quando o agente recebe um reforço deve propagá-lo de alguma maneira no tempo. Desta forma, os estados que foram anteriormente visitados e que conduziram a esta condição, serão associados a uma predição de conseqüências futuras. Isto está baseado em uma suposição importante em processos dinâmicos, chamada Processo de Decisão de Markov.

A aquisição de informações de aprendizado por experiência direta é uma prática que normalmente não está sob o total controle de um agente: ele pode escolher ações, mas não pode determinar as conseqüências destas ações com antecedência para todos estados, pois normalmente não tem um modelo suficientemente preciso do processo no qual são baseados os julgamentos. Por isto, o agente deve estimar o custo esperado através de visitação direta aos estados; ele deve escolher uma ação, receber um resultado e propagá-lo aos estados anteriores, seguindo o procedimento de TD.

A correta determinação das conseqüências das ações depende de um número razoável de atualizações para cada estado. A convergência de algoritmos estocásticos adaptáveis em geral está condicionada a um número infinito de visitas para todos os estados do processo. A impossibilidade de cumprir esta condição cria o conflito entre *exploration* e *exploitation*. O agente deve encontrar políticas de ações que permitam uma melhor exploração do conjunto de estados(*exploration*) e por conseguinte um melhor modelo. Mas ao mesmo tempo deve-se considerar que um bom desempenho só é atingido se uma melhor política de ação é explorada (*exploitation*). Na literatura de Controle, este conflito entre os parâmetros "objetivo de estimação", determinando um modelo correto do processo, e "objetivo de controle", executando a melhor política de ação possível, é chamado de *dual control problem* [Bertsekas, 1995a]. A visita direta dos estados como o único meio de adquirir conhecimento causa o problema mencionado acima.

2.1 Modelo Padrão de Aprendizado com Reforço

No modelo básico de aprendizado com reforço, um agente é conectado a seu ambiente por percepção e ação, como descrito na Figura 2.1. Em cada passo de interação, o agente recebe como entrada, i, alguma indicação do estado atual, s, do ambiente; o agente escolhe então uma ação, a, gerada como saída. A ação muda o estado do ambiente e é comunicado o valor desta transição de estado ao agente por um *sinal de reforço* escalar, r. O comportamento do agente, B, deveria escolher ações que tendem a aumentar, ao longo do tempo, a soma dos valores do sinal de reforço. Ele pode aprender a fazer isto

com o passar do tempo através da sistemática tentativa-e-erro, guiado por uma grande variedade de algoritmos.

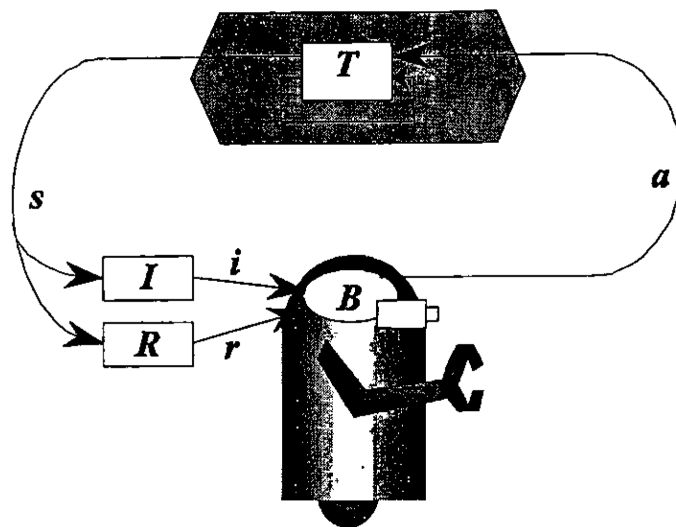


Figura 2.1: Modelo padrão de aprendizado com reforço.

O trabalho do agente é achar uma política π, mapeando estados em ações, para maximizar a medida de reforço ao longo do tempo. Formalmente, o modelo consiste de: um conjunto discreto de estados de ambiente, S; um conjunto discreto de ações de agente, A; e um conjunto de sinais de reforço, tipicamente $\{0,1\}$, ou números reais.

A Figura 2.1 também inclui uma função de entrada I, que determina como o agente vê o estado do ambiente. Somente para estudos dos algoritmos, assume-se que I é a função identidade, ou seja, o agente percebe o estado exato do ambiente. A maioria dos sensores de robô, incluindo dispositivos baratos como os sonares e dispositivos caros como os escâneres a laser, são incertos. Por este motivo, às vezes objetos não são vistos ou suas distâncias não são dadas corretamente. Para atenuar este problema propomos neste trabalho a utilização de lógica *fuzzy* na função I, para tranformar os sinais recebidos pelo sonar em indicações do estado atual.

Uma forma bastante conhecida utilizada atualmente para modelar problemas de RL é o Processo de Decisão de Markov (MDP).

2.2 Processo de Decisão de Markov

Na estrutura de aprendizagem com reforço, o agente faz suas decisões com base num sinal do ambiente, chamado de estado do ambiente. Caso um estado do ambiente contenha toda a informação relevante então ele é chamado de *Markov* ou que tem a propriedade de Markov. Pode-se observar esta propriedade tomando-se a velocidade e posição atual de uma bola de canhão, observando que estas informações são suficientes para determinar seu vôo futuro, não importando com que velocidade saiu e de que posição veio, o que for importante será obtido do estado corrente.

Uma tarefa de aprendizagem com reforço que satisfaça a propriedade de Markov é chamada de Processo de Decisão de Markov ou MDP (*Markov Decision Process*). Se os estados e ações forem finitos, então será chamado de MDP finito.

Formalmente, um MDP consiste de:

- um conjunto de estados do ambiente, S
- um conjunto de possíveis ações $A(s)$
- uma função de probabilidade de transição para s' dado s e a, $P(s'|s,a)$
- recompensas esperadas para a transição para s' dado s e a, $R(s'|s,a)$,

onde $s', s \in S$ e $a \in A(s)$,

Existem boas referências para MDPs, que podem ser encontradas em [Bellman, 1957], [Puterman, 1994], [Howard, 1960], [Bertsekas, 1987].

Exemplo

Para ilustrar um MDP apresentamos a seguir um exemplo simples, porém não realista, de um robô que tem por objetivo pegar o maior número de latas possíveis, gastando o mínimo de energia.

Suponha que sejam consideradas as três seguintes decisões: procurar ativamente por uma lata, permanecer parado esperando que alguém lhe traga a lata e voltar à base

para recarregar a bateria. O melhor modo de encontrar latas é procurando-as ativamente, mas isto descarrega a bateria do robô. Por outro lado, somente esperar não é uma boa estratégia de se conseguir as latas. Sempre que o robô está procurando latas é possível que sua bateria se esgote, neste caso o robô desliga e espera seu resgate o que provoca uma recompensa baixa.

O robô baseia-se no nível de energia da bateria para fazer suas decisões, distinguidas por dois níveis (**alto** e **baixo**), e tem a possibilidade de escolher entre **esperar** que lhe tragam as latas, **procurar** por latas ou **recarregar** se o nível da bateria estiver baixo. Definimos então os conjuntos de estados S e o conjunto de ações $A(s)$, como:

$S = \{alto, baixo\}$

$A(alto) = \{procurar, esperar\}$

$A(baixo) = \{procurar, esperar, recarregar\}$

A cada lata coletada é adicionada +1 na recompensa e caso ele fique sem energia uma punição de -3 é administrada. $R^{procurar}$ e $R^{esperar}$ representam o número de latas coletadas enquanto "procurava" e "esperava" respectivamente, tal que $R^{procurar} > R^{esperar}$. Finalmente, para deixar as coisas simples, supõe-se que nenhuma lata pode ser coletada durante a ida à base para recarregar e que nenhuma lata pode ser coletada em um passo no qual a bateria é esvaziada. Por ser este um sistema MDP finito, nós podemos escrever as probabilidades de transição e as recompensas esperadas como na Tabela 2.1 ou como um diagrama de transição de estados visto na Figura 2.2.

Estando com a bateria no nível alto e executando a ação "procurar" podem ocorrer duas coisas: a bateria continuar alta, $P = \alpha$, ou baixa, $P = 1 - \alpha$. Caso esteja com o nível baixo e executa a ação "procurar" teremos duas possibilidades: continuar no nível baixo com $P = \beta$ ou descarregar a bateria, $P = 1 - \beta$, precisando que alguém o leve para recarregar. Pelo objetivo proposto o robô não deve ficar sem energia e por isso ele foi punido com uma recompensa negativa. Quando se escolhe a opção "esperar" não há gasto de energia, ficando o robô no mesmo estado, desta forma as opções em que há mudança de estado tem probabilidade 0(zero) de ocorrer. No caso da escolha da ação "recarregar"

$s = s_t$	$s' = s_{t+1}$	$a = a_t$	$P(s'\|s,a)$	$R(s'\|s,a)$
alto	alto	procurar	α	$R^{procurar}$
alto	baixo	procurar	$1-\alpha$	$R^{procurar}$
baixo	alto	procurar	$1-\beta$	-3
baixo	baixo	procurar	β	$R^{procurar}$
alto	alto	esperar	1	$R^{esperar}$
alto	baixo	esperar	0	$R^{esperar}$
baixo	alto	esperar	0	$R^{esperar}$
baixo	baixo	esperar	1	$R^{esperar}$
baixo	alto	recarregar	1	0
baixo	baixo	recarregar	0	0

Tabela 2.1: Transição de estados.

o próximo estado será de bateria alta, não havendo outra possibilidade.

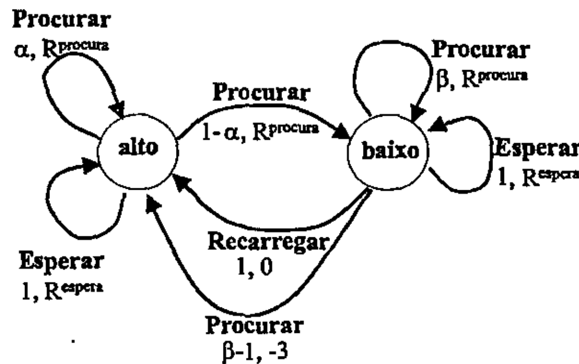


Figura 2.2: Diagrama de transição de estados.

2.3 Função de Custo Ótimo

Todos os algoritmos de aprendizado são baseados na estimativa de *funções de custo* — funções de estados (ou de par estado-ação) que estimam *quão bom* é para o agente estar em determinado estado (ou quão bom é executar uma dada ação em um dado estado). A noção de "quão bom" é definida aqui em termos de recompensas futuras, que podem ser valores esperados ou precisos, em termos do retorno esperado. As recompensas que o agente espera receber, depende das ações que ele escolherá. Deste modo, funções de custo são definidas de acordo com políticas particulares.

A política π é um mapeamento de estados, $s \in S$, e ações, $a \in A(s)$, em uma probabilidade $\pi(s,a)$ de se escolher uma ação a em um estado s. O *custo* de um estado s sob uma política π será denotado por $V^{\pi}(s)$, sendo definido formalmente para um MDP como:

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

onde E_{π} denota o valor dado por um agente que segue uma política π, e t é um passo de tempo. R é o conjunto de recompensas, e $r \in R$, $0 \leq \gamma < 1$ é o fator de desconto que garante que a soma não será infinita, ou ainda, fazendo com que estados mais próximos de s_t tenham maior influência no mesmo.

De forma similar, o custo da escolha de uma ação a em um estado s seguindo uma política π é definido por:

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

Uma política π é definida como sendo melhor ou igual a uma política π' se o seu retorno esperado for maior ou igual que π', para todos os estados. Em outras palavras, $\pi \geq \pi' \Leftrightarrow V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in S$.

A **função de custo ótimo** denotada por V^*, é definida abaixo pela equação de Bellman (2.4), na qual π^* representa uma política ótima. A equação de Bellman expressa o fato de que o valor de um estado sob uma política ótima deve ser igual ao retorno esperado pela melhor ação deste estado:

$$\begin{aligned}
V^*(s) &= \max_{\pi} V^{\pi}(s), \forall s \in S. && (2.3)\\
&= \max_{a \in A(s)} Q^{\pi}(s,a) \text{ — usando a equação (2.2)} \\
&= \max_{a \in A(s)} E_{\pi}\Big\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\Big\} \\
&= \max_{a \in A(s)} E_{\pi}\Big\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|s_t = s, a_t = a\Big\} \\
&= \max_{a \in A(s)} E\Big\{r_{t+1} + \gamma V^*(s_{t+1})|s_t = s\Big\} && (2.4)
\end{aligned}$$

Dado uma função de custo ótima, pode-se especificar uma **política ótima** π^* como:

$$\pi^*(s) = \arg V^*(s). \tag{2.5}$$

Similarmente, a função de custo ótimo para o par estado-ação é definida como:

$$\begin{aligned} Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a), \forall s \in S. & (2.6)\\ &= E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\} & (2.7) \end{aligned}$$

Pode-se escrever Q^* em função de V^*, como segue:

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\}. \tag{2.8}$$

Em [Sutton and Barto, 1998], pode ser encontrada uma boa referência para o estudo de funções de custo.

2.4 Programação Dinâmica

Programação Dinâmica (DP) é um método básico que utiliza uma política estacionária π^* para problemas estocásticos. DP altualmente envolve um grande variedade de técnicas, todas elas baseadas em um simples princípio de busca ótima e em alguns teoremas básicos.

Dois operadores usuais utilizados em teoremas DP são:

- **O Operador de Aproximação Sucessiva** T_π.

  Para qualquer função de custo $V : S \mapsto \Re$ e uma política de ação $\pi : S \mapsto A$, temos:

$$(T_\pi V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V(s'). \tag{2.9}$$

- **O Operador de Iteração de Custo T.**

  Para qualquer função de custo $V : S \mapsto \Re$, temos:

$$(TV)(s) = \max_a \Big[r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s') \Big]. \tag{2.10}$$

2.4.1 Teoremas Básicos de Programação Dinâmica

Serão apresentados abaixo teoremas fundamentais de DP para problemas *infinite-horizon discounted*. As provas destes teoremas podem ser encontradas em [Bertsekas, 1995b], [Bellman, 1957], [Puterman, 1994].

O modelo *infinite-horizon discounted*, representado na equação (2.11), recebe as recompensas com o passar do tempo, sendo que as recompensas futuras são geometricamente descontadas de acordo com o fator de desconto γ, $(0 \leq \gamma < 1)$:

$$E(\sum_{t=0}^{\infty} \gamma^t r_t). \tag{2.11}$$

Podemos interpretar γ de diversas formas. Ele pode ser visto como uma taxa de interesse, uma probabilidade de estar em outro passo ou como um truque matemático para evitar a soma infinita.

Teorema 1 *Para qualquer função inicial $V(s)$ e estado s, a função de custo ótimo satisfaz a seguinte relação:*

$$V^*(s) = \lim_{N \to \infty} (T^N V)(s) \tag{2.12}$$

Teorema 2 *Para todas as políticas de ação estacionária π e estado s, a função custo satisfaz a seguinte relação:*

$$V^\pi(s) = \lim_{N \to \infty} (T_\pi^N V)(s) \tag{2.13}$$

Quanto mais instâncias de $V(s)$ são utilizadas, melhor será a função de custo. Seguindo esta idéia, quando o número de instâncias tender ao infinito, teremos uma

função de custo ótima, como mostrado nas Equações (2.12) e (2.13).

Teorema 3 (Princípio de Otimalidade de Bellman) *A função de custo ótima V^* é o ponto fixo do operador T, ou em outras palavras, V^* é a única função que satisfaz $V^* = TV^*$.*

Teorema 4 *Para toda política de ação estacionária π, a função de custo associada V^π é o ponto fixo do operador T_π, ou em outras palavras, V^π é a única função que satisfaz $V^\pi = T_\pi V^\pi$.*

Teorema 5 *Uma política de ação estacionária π é ótima se e somente se $T_\pi V^* = TV^*$.*

2.4.2 Técnicas Básicas de Programação Dinâmica

Há duas classes principais de métodos bem estabelecidos para se descobrir políticas ótimas em MDPs. Ambas são baseadas nos teoremas descritos acima.

Método de Iteração de Custo. Este método consiste de aplicação recursiva do operador T em uma aproximação inicial arbitrária V da função de custo ótimo. Como V^* é o ponto fixo de T (Teorema 3), o método encontra uma política ótima através de:

$$\pi^*(s) = \arg\Big[\lim_{N\to\infty}(T^N V)(s)\Big] = \arg\max_a\Big[r(s,a) + \gamma\sum_{s'\in S} P(s'|s,a)V^*(s')\Big] \tag{2.14}$$

Método de Iteração de Política. Dado uma política inicial π^0, dois passos sucessivos são executados em *loop* até que π^*, ou uma boa aproximação, seja encontrada.

No primeiro passo calcula-se a política atual π_k (i.e., tem seus custos associados calculados) exatamente através da solução de um sistema de equações lineares:

$$V_{\pi_k} = T_{\pi_k} V_{\pi_k} \tag{2.15}$$

ou aproximadamente através de um número finito de aplicações M do operador de Apro-

ximação Sucessiva:

$$V_{\pi_k} \approx T_{\pi_k}^M V_0 \tag{2.16}$$

A Equação (2.16) é o passo de *Avaliação da Política*.

No segundo passo, uma política estacionária melhorada π^{k+1} é obtida através de:

$$\pi_{k+1}(s) = \arg[(TV_{\pi_k})(s)] = \arg\max_a \Big[r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_{\pi_k}(s')\Big]. \tag{2.17}$$

A política da Equação (2.17) é chamada de política *gulosa* em relação a V_{π_k}.

2.5 Métodos de Monte Carlo

A Programação Dinâmica requer um modelo completo e explícito do processo a ser controlado, ou seja, deve ter disponível as probabilidades de transição. Por outro lado, aprendizagem autônoma está completamente baseado em experiência interativa e não requer nenhum modelo. Em RL as transições são simuladas em amostra geradas por se ter disponível um modelo "fraco", ou seja, sem uma descrição analítica completa baseada em probabilidades de transição. Entretanto, é interessante considerar um caso intermediário, o qual faz uma ponte entre DP e RL. Considere o problema: Para uma dada política de ação fixa π, como é possível calcular *por simulação* — i.e., sem a ajuda de DP — a função custo V^π? A solução mais simples é executar muitas trajetórias simuladas para cada estado e calcular a média aritmética dos reforços acumulados obtidos, ou seja, uma *simulação de Monte Carlo.*

Para um dado estado s_t, denota-se $v^\pi(s_t, m)$ como o custo acumulado discontado obtido por simulação, depois que s_t é visitado pela m-ésima vez:

$$v_\pi(s_t, m) = r(s_t) + \gamma r(s_{t+1}) + \gamma^2 r(s_{t+2}) + \ldots \tag{2.18}$$

onde $s_t, s_{t+1}, \ldots$ são os estados visitados sucessivamente em uma execução em particular

e $r(s_t) \equiv r(s_t, \pi(s_t))$ é uma notação mais simples considerando que a política de ação é fixa. Assumindo que as simulações produzem as somas desejadas, tem-se:

$$V_\pi(s_t) = E[v_\pi(s_t, m)] = \lim_{M \to \infty} \frac{1}{M} \sum_{m=1}^{M} v^\pi(s_t, m) \tag{2.19}$$

Esta média pode ser calculada iterativamente por procedimentos de Robbins-Monro [Robbins and Monro, 1951]:

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha_m[v^\pi(s_t, m) - V^\pi(s_t)], \text{ onde } \alpha_m = \frac{1}{m}. \tag{2.20}$$

2.6 Métodos de Diferença Temporal

Quase todas as técnicas atualmente em uso estão baseadas no método de Diferença Temporal (TD), apresentado por [Sutton, 1988]. Em contraste com tentativas anteriores de implementar a idéia de reforço, TD fornece uma estrutura matemática consistente.

Tanto TD como métodos de Monte Carlo utilizam a experiência para resolver o problema de predição. Dada alguma experiência seguindo uma política π, ambos os métodos atualizam suas estimativas v de V^π. Para propostas *on-line* a Equação (2.20) apresenta um problema: $V_\pi(s_t)$ pode ser atualizado somente depois que $v_\pi(s_t, m)$ for calculado através de uma execução completa da simulação. O método de TD fornece uma boa solução para este problema, forçando atualizações imediatas depois de visitar um novo estado.

A fim de derivar Diferenças Temporais, a Equação (2.20) será expandida como:

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha_m[r(s_t) + \gamma r(s_{t+1}) + \gamma^2 r(s_{t+2}) + \cdots - V^\pi(s_t)], \tag{2.21}$$

Utilizando um truque matemático, adiciona-se a cada recompensa o termo $\gamma V^\pi(s_{k+1})$ e subtrai-se o mesmo termo da recompensa seguinte (onde k é o tempo para $r(s_k)$), sendo

assim temos:

$$\begin{aligned} V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha_m[\quad & (r(s_t) + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)) + \\ \gamma \quad & (r(s_{t+1}) + \gamma V^{\pi}(s_{t+2}) - V^{\pi}(s_{t+1})) + \\ \gamma^2 \quad & (r(s_{t+2}) + \gamma V^{\pi}(s_{t+3}) - V^{\pi}(s_{t+2})) + \\ & \cdots \quad] \end{aligned}$$

ou

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha_m[\quad d_t + \gamma d_{t+1} + \gamma^2 d_{t+2} + \cdots \quad] \tag{2.22}$$

onde os termos $d_t = r(s_t) + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$ são chamados de *diferenças temporais*. Ele representam uma estimativa da diferença de tempo k entre o custo *esperado* $V^{\pi}(s_t)$ e o custo *predito* $r(s_t) + \gamma V^{\pi}(s_{t+1})$. A base desta idéia está no fato de $r(s_t) + \gamma V^{\pi}(s_{t+1})$ ser uma amostra do valor de $V^{\pi}(s_t)$, sendo mais provável de estar correta, pois esta incorpora o valor real $r(s_t)$.

A Equação (2.22) é o método $TD(1)$ para calcular os custos esperados. Uma fórmula mais geral é descontar a influência de diferenças temporais independentemente de γ, utilizando um fator $\lambda \in [0,1]$, originando o método $TD(\lambda)$:

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha_m[d_t + \gamma\lambda d_{t+1} + \gamma^2\lambda^2 d_{t+2} + \cdots] \tag{2.23}$$

Deve-se tomar muito cuidado para não confundir as diferentes regras de γ e λ. A constante γ é um desconto sobre as recompensas futuras e faz parte da especificação do problema, já a constante λ é um desconto sobre futuras diferenças temporais e faz parte de uma escolha sobre o algoritmo utilizado para resolver o problema.

O algoritmo $TD(\lambda)$ seguindo a Equação (2.23) não é diretamente implementável, pois diferenças futuras $d_t, d_{t+1}, \ldots$ são utilizadas para atualizar $V^{\pi}(s_t)$. Isto pode ser contornado utilizando os chamados *sinais de elegibilidade* $e_t(s)$, definido como:

$$e_t(s) = \begin{cases} \lambda\gamma e_{t-1}(s) & \text{se } s \neq s_t \\ \lambda\gamma e_{t-1}(s) + 1 & \text{se } s = s_t \end{cases}$$

Uma forma implementável de $TD(\lambda)$, equivalente à Equação (2.23), é apresentada no Algoritmo 2.1.

Algoritmo 2.1 TD(λ)

Requisito: $s, s' \in S$ e $a \in A(s)$
1: $V(s) \leftarrow 0, e(s) \leftarrow 0$ para todo $s \in S$
2: $s \leftarrow$ estado atual
3: para cada passo do episódio:
4: **repita**
5: $a \leftarrow$ ação dada por $\pi(s)$ {por exemplo ϵ-gulosa}
6: execute a ação a, receba a recompensa r e o novo estado s'
7: $\delta \leftarrow r + \gamma V(s') - V(s)$
8: $e(s) \leftarrow e(s) + 1$
9: **para todo** $s \in S$ **faça**
10: $V(s) \leftarrow V(s) + \alpha\delta e(s)$
11: $e(s) \leftarrow \gamma\lambda e(s)$
12: **fim-para**
13: $s \leftarrow s'$
14: **até que** s seja terminal

Uma variação que tem tido bons resultados em alguns problemas, consiste em usar o sinal de elegibilidade $e_t(s_t) = 1$, em vez de $e_t(s_t) + 1$, na linha 8 do Algoritmo 2.1.

Tem sido provado que o algoritmo TD(λ) converge para o custo discontado correto, contanto que algumas condições sejam satisfeitas [Jaakkola et al., 1994], a mais importante dessas é fazer com que todos os estados sejam visitados um número infinito de vezes. O uso de $0 \leq \lambda < 1$ tem sido verificado na prática [Tesauro, 1992] e demonstrado analiticamente em alguns casos muito simples [Singh and Dayan, 1994], mas ainda não se tem um resultado genérico consistente.

O método TD mais simples, conhecido como TD(0), é descrito por:

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha_m[r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)]. \tag{2.24}$$

Muito frequentemente, TD(0), é adotado com um ponto de partida para os estudos de implementação teórico e prático.

2.7 Algoritmos de Aprendizado com Reforço

Os métodos de aprendizado com reforço estão divididos em: "métodos independentes de modelo" e "métodos baseados em modelo". Um modelo consiste em se ter o conhecimento prévio da função de probabilidade de transição $P(s'|s,a)$ e da função de reforço $R(s'|s,a)$. Os métodos independentes de modelo (*model-free*) aprendem um controlador sem ter um modelo explícito dos efeitos das ações, já os métodos baseados em modelo (*model-based*) aprendem e usam um modelo de ações enquanto simultaneamente aprendem um controle ótimo [Barto et al., 1993]. A preocupação principal em aprendizado com reforço é obter uma política ótima quando o modelo for desconhecido.

Os algoritmos Q-*learning* [Watkins, 1989] e R-*learning* [Schwartz, 1993] são exemplos de métodos independentes de modelo, enquanto H-*learning* [Tadepalli and Ok, 1994] é um exemplo de método basedado em modelo.

2.7.1 Algoritmos Q-*learning* e SARSA

O algoritmo Q-*learning*, uma técnica proposta por [Watkins, 1989], é um método iterativo para o aprendizado de uma política de ação em agentes autônomos. Este método é baseado na medida de custo de ações $Q^*(s_t,a_t)$, o qual representa o custo descontado esperado por executar uma ação a_t no estado s_t, seguindo uma política ótima. $Q^*(s_t,a_t)$ pode ser escrito recursivamente como:

$$\begin{aligned} Q^*(s_t,a_t) &= E[r(s_t,a_t) + \gamma V^*(s_{t+1})] \\ &= r(s_t,a_t) + \gamma \sum_{s_{t+1}\in S} P(s_{t+1}|s_t,a_t) V^*(s_{t+1}), \end{aligned} \tag{2.25}$$

Desta definição e como uma consequência do "princípio de otimalidade de Bellman", tem-se:

$$V^*(s_t) = TV^*(s_t)$$

$$
\begin{aligned}
&= \max_a \Big[r(s_t, a) + \gamma \sum_{s_{t+1} \in S} P(s_{t+1}|s_t, a) V^*(s_t + 1) \Big] \\
&= \max_a Q^*(s_t, a) \qquad (2.26)
\end{aligned}
$$

e a Equação (2.25) pode ser reescrita como:

$$
Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} P(s_{t+1}|s_t, a_t) \max_a Q^*(s_{t+1}, a). \qquad (2.27)
$$

Como $V^*(s_t) = \max_a Q^*(s_t, a)$, pode-se escrever uma política ótima como

$$
\pi^* = \arg\max_a Q^*(s_t, a).
$$

Devido a função Q produzir ações explícitas, pode-se estima-se o custo de Q de forma *on-line* utilizando essencialmente o mesmo método TD(0). Para definir a política, o método TD(0) também será utilizado, pois uma ação pode ser escolhida justamente por levar ao custo máximo de Q no estado atual. A regra do Q-*learning* é:

$$
Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t [r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q_t(s_t, a_t)]
$$

Se cada ação for executada em cada estado um número infinito de vezes em um número infinito de episódios e α for decrementado apropriadamente, os custos de Q irão convergir com probabilidade 1 para Q^* [Watkins, 1989], [Jaakkola et al., 1994] e [Tsitsiklis, 1994]. O algoritmo Q-*learning* tem sido bastante usado por convergir para uma política ótima, não importando a ordem de como os estados são visitados. O Algoritmo 2.2 descreve uma rotina implementável para um episódio do Q-*learning*.

Uma variação interessante para o Q-*learning* é o algoritmo SARSA [Sutton, 1996], o qual utiliza o Q-*learning* como parte do mecanismo de Iteração de Política. A atualização dos custos das ações são realizadas em cada passo de acordo com a equação abaixo:

$$
Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t [r(s_t, a_t) + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]
$$

Algoritmo 2.2 Q-*learning*

1: inicialize $Q(s,a)$ arbitrariamente
2: $s \leftarrow$ estado atual
3: para cada passo do episódio:
4: **repita**
5: $a \leftarrow$ ação derivada de Q {por exemplo ϵ-gulosa}
6: execute a ação a.
7: receba a recompensa r e o próximo estado s'
8: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
9: $s \leftarrow s'$
10: **até que** s seja terminal

Naturalmente, caso a ação escolhida a_{t+1} seja $\arg\max_a[Q(s_{t+1},a)]$, este algoritmo será equivalente ao Q-*learning* padrão. SARSA entretanto admite que a_{t+1} seja escolhido aleatóriamente com uma probabilidade predefinida. Por eliminar o uso do operador max sobre as ações, este método é mais rápido que o Q-*learning* para situações onde o conjunto de ações tenham alta cardinalidade. A descrição do algoritmo SARSA é mostrada no Algoritmo 2.3.

Algoritmo 2.3 SARSA

1: inicialize $Q(s,a)$ arbitrariamente
2: $s \leftarrow$ estado atual
3: $a \leftarrow$ ação derivada de Q {por exemplo ϵ-gulosa}
4: para cada passo do episódio:
5: **repita**
6: execute a ação a.
7: receba a recompensa r e o próximo estado s'
8: $a' \leftarrow$ ação derivada de Q, onde $a' \in A(s')$
9: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
10: $s \leftarrow s'$
11: $a \leftarrow a'$
12: **até que** s seja terminal

Algumas variações são obtidas combinando o $TD(\lambda)$ com algoritmos Q-*learning* e SARSA. Tais variações são conhecidas respectivamente como Q(λ)[Watkins, 1989] e SARSA(λ) e suas descrições podem ser encontradas em [Sutton and Barto, 1998].

2.7.2 R-*learning*

R-*learning* é uma técnica proposta por Schwartz [Schwartz, 1993] que maximiza a recompensa média a cada passo. R-*learning* é um método de controle independente de política para versões avançadas de aprendizado com reforço nas quais não se utilizam descontos e nem dividem as experiências em episódios distintos com retornos finitos.

O algoritmo Q-*learning* não maximiza a recompensa média, mas descontos acumulados de recompensa, por isso R-*learning* deve fornecer de fato resultados melhores que o Q-*learning*. Em R-*learning* utiliza-se o *average-reward model*, ou seja, a função de custo para uma política π é definida em relação à média das recompensas esperadas em cada passo de tempo, como:

$$\rho^{\pi} = \lim_{n \to \infty} E\Big(\frac{1}{h} \sum_{t=0}^{n} r_t\Big). \tag{2.28}$$

Será assumido que este é um processo *ergodic* (a probabilidade de se sair de qualquer estado e ir para outro sob uma política deve ser diferente de zero) e deste modo ρ^{π} é independente do estado inicial.

O custo da ação $R^{\pi}(s, a)$ representa o valor ajustado da média de realizar uma ação a no estado s, seguindo uma política π subsequentemente, ou seja:

$$R^{\pi}(s_t, a_t) = r(s_t, a_t) - \rho_t^{\pi} + \sum_{s_{t+1} \in S} P(s_{t+1}|s_t, a_t) V^{\pi}(s_{t+1}), \tag{2.29}$$

onde $V^{\pi}(s_{t+1}) = \max_{a \in A} R^{\pi}(s_{t+1}, a)$.

Os valores de R são ajustados a cada ação baseados na seguinte regra:

$$R_{t+1}(s_t, a_t) \leftarrow R_t(s_t, a_t) + \alpha_t[r(s_t, a_t) - \rho_t + \max_a R_t(s_{t+1}, a) - R_t(s_t, a_t)], \tag{2.30}$$

que difere da regra do Q-*learning*, simplesmente por subtrair a recompensa média ρ do reforço imediato $r(s, a)$ e por não ter desconto γ para o próximo estado. A recompensa

média é calculada como:

$$\rho_{t+1} \leftarrow \rho_t + \beta_t[r(s_t, a_t) + \max_a R_t(s_{t+1}, a) - \max_a R_t(s_t, a) - \rho_t] \qquad (2.31)$$

O ponto chave é que ρ somente é atualizado quando uma ação não aleatória foi tomada, ou seja, $\max_a R_t(s_t, a) = R_t(s_t, a_t)$. A recompensa média ρ não depende de algum estado particular, ela é uma constante para todo o conjunto de estados. A descrição do algoritmo R-*learning* é apresentada no Algoritmo 2.4.

Algoritmo 2.4 R-*learning*

1: inicialize ρ e $R(s, a)$ arbitrariamente
2: **repita** para sempre
3: $s \leftarrow$ estado atual
4: $a \leftarrow$ ação baseda numa política de comportamento ϵ-gulosa
5: Execute a ação $a \in A(s)$
6: receba a recompensa r e o próximo estado s'
7: $R(s, a) \leftarrow R(s, a) + \alpha[r - \rho + \max_{a'} R(s', a') - R(s, a)]$
8: se $R(s, a) = \max_a R(s, a)$ **então**
9: $\rho \leftarrow \rho + \beta[r - \rho + \max_{a'} R(s', a') - \max_a R(s, a)]$
10: **fim-se**
11: **fim-repita**

2.7.3 H-*learning*

O algoritmoH-*learning* [Tadepalli and Ok, 1994] é um algoritmo que utiliza métodos baseados em modelo e assim como o R-*learning* foi introduzido para otimizar a recompensa média sem utilizar desconto.

Um conjunto de estados é *ergodic* com respeito a uma política, se a probabilidade de sair de qualquer estado e ir para outro for diferente de zero, sendo que não pode haver transições para fora do conjunto.

A seguir, assume-se que as garantias de estratégia de exploração para todo estado pertencente a S é visitado com alguma probabilidade, de forma que o conjunto total de estados é *ergodic* para toda política durante o treinamento.

Sob estas condições, as recompensas finitas iniciais obtidas, indo de um estado s para s', não contribuem em nada à recompensa de média esperada, a longo prazo. Consequentemente, se μ é uma política intermediária usada em treinamento, então $\rho^{\mu}(s) = \rho^{\mu}(s')$. Denota-se apenas por $\rho(\mu)$ e considera-se o problema de encontrar uma política ótima μ^* que maximize $\rho(\mu)$. Este problema é resolvido pelo seguinte teorema e é provado em [Bertsekas, 1987].

Teorema 6 *Se um escalar ρ e um vetor n-dimensinal h satisfazem a relação recorrente*

$$h(i) = \max_{u \in U(i)} \{r(i,u) + \sum_{j=1}^{n} p_{ij}(u)h(j)\} - \rho, \quad i = 1, \ldots, n. \tag{2.32}$$

então ρ é uma ótima recompensa média $\rho(\mu)$ e μ^* atinge o máximo para cada estado i.

O H-*learning* estima as probabilidades $P(s'|s,a)$ e os reforços $R(s,a)$ por contagem direta e atualiza os valores de h utilizando a equação (2.32), como podemos observar no Algoritmo 2.5 a descrição do H-*learning*.

O H-*learning* faz escolhas aleatórias de ações com uma probabilidade fixa, ou seja, também utiliza a política de escolha de ações ϵ-gulosa. O método utilizado para atualizar a recompensa média ρ segue a idéia de Schwartz [Schwartz, 1993]. Isso pode ser percebido comparando-se a Equação (2.31), que faz a atualização de ρ para o algortimo R-*learning* utilizando-se $\beta = 1/T$, com a atualização de ρ feita pelo H-*learning*.

Algoritmo 2.5 H-*learning*

Considere $N(s,a)$ como o número de vezes que a ação a foi executada no estado s, e $N(s,a,s')$ como o número de vezes que $N(s,a)$ resultou no estado s'. $P(s'|s,a)$ é a função de probabilidade de atingir o estado s' estando no estado s e executando a ação a. $r(s,a)$ é a recompensa esperada e r' é a recompensa imediata. $h(s)$ representa o recompensa esperada estando no estado s, $A_{best}(s)$ é o conjunto de ações ótimas no estado s e T é o número total de passos que uma ação aparentemente ótima foi executada.

1: Inicialize as matrizes $P(s'|s,a)$, $r(s,a)$, $h(s)$ e o escalar ρ com 0's.
2: $A_{best}(s) \leftarrow A(s)$
3: $T \leftarrow 0$
4: $s \leftarrow$ valor aleatório do estado corrente.
5: **repita**
6:   **se** a estratégia de exploração sugere uma ação aleatória para s **então**
7:     $a \leftarrow$ ação aleatória
8:   **senão**
9:     $a \leftarrow A_{best}(s)$
10:   **fim-se**
11:   execute a ação a
12:   receba a recompensa imediata r' e o estado resultante s'
13:   $N(s,a) \leftarrow N(s,a)+1$
14:   $N(s,a,s') \leftarrow N(s,a,s')+1$
15:   **para todo** $s' \in S$ **faça**
16:     $P(s'|s,a) \leftarrow N(s,a,s')/N(s,a)$
17:   **fim-para**
18:   $r(s,a) \leftarrow r(s,a) + (r' - r(s,a))/N(s,a)$
19:   **se** $a \in A_{best}(s)$ **então**
20:     $T \leftarrow T+1$
21:     $\rho \leftarrow \rho + (r' - h(s) + h(s') - \rho)/T$
22:   **fim-se**
23:   **para todo** $a \in A(s)$ **faça**
24:     $H(s,a) = r(s,a) + \sum_{s' \in S} P(s'|s,a)h(s')$
25:   **fim-para**
26:   $A_{best}(s) \leftarrow \arg\max_{a \in A(s)} H(s,a)\}$
27:   $h(s) \leftarrow H(s,a) - \rho$, onde $a \in A_{best}(s)$
28:   $s \leftarrow s'$
29: **até que** atinja convengência ou MAX-STEPS vezes

O valor $h(s)$ representa a recompensa esperada estando no estado s, sendo equivalente a $\max_{a \in A(s)} R(s,a)$ do algoritmo R-*learning*.

Todos os métodos em Aprendizado com Reforço, exceto o H-*learning*, tem um

ou mais parâmetros, como por exemplo, o *Q-learning* tem α e γ e o *R-learning* tem α e β. A performance de todos estes algoritmos são sensíveis a estes parâmetros, e conseqüentemente fica necessário ajustá-los para obter um melhor desempenho.

Esses três algoritmos foram implementados e testados para uma trarefa de aprendizado para o robô *Pioneer 1 Gripper*. Deta forma, o robô, seus recusos e seu software de controle são apresentados no próximo capítulo e a modelagem do problema, assim como os resultados dos testes com os algoritmos são apresentados no Capítulo 4.

Capítulo 3

Robô e Software de Controle

Neste capítulo serão apresentados os sensores, atuadores e algumas limitações do robô *Pioneer 1 Gripper*, seguido do seu software de controle *Saphira* e algumas funções de sua biblioteca que serão de grande utilidade para implementação das tarefas de aprendizagem.

O robô *Pioneer 1 Gripper*[1] está montado sobre um eixo de duas rodas que lhe permite fazer rotações e movimentos para frente ou para trás. Como se pode observar na Figura 3.1, este é um robô de pequenas dimensões. A garra possui dois braços que podem erguer objetos com no máximo 1.4Kg. Possui um sistema de elevação vertical que permite pegar objetos com no mínimo 2cm de altura. Possui três possibilidades de conexão com o microcomputador: via rede, porta serial ou transmissão, via ondas de rádio. Para utilizar a transmissão via ondas de rádio o transmissor é conectado à porta serial do microcomputador.

Quanto aos sensores, possui sete sonares ultra-sônicos sendo cinco frontais, um na lateral direita e um na lateral esquerda. Cada sonar leva 40ms para realizar uma leitura, ou seja, possui uma taxa de disparo de 25Hz. Possui também sensores de colisão nas rodas e nas extremidades da garra. Os valores de leitura dos sonares variam de 200mm à 5000mm. Desta forma para objetos detectados abaixo deste intervalo será retornado

[1] *Activ*Media Robotics

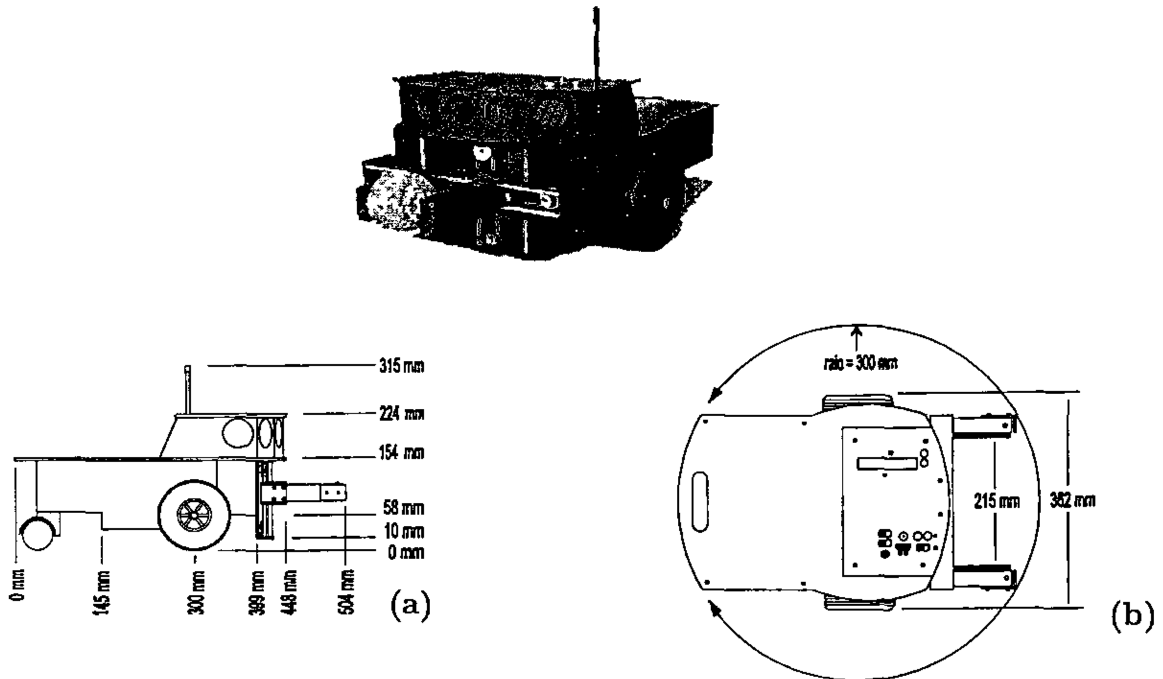


Figura 3.1: Pioneer 1 Gripper, (a) vista lateral, (b) vista superior

200mm e para objeto detectados acima deste intervalo ou objeto não detectado será retornado 5000mm. Os sonares emitem raios em projeção cônica e nem sempre é possível captar de volta os raios refletidos, devido ao ângulo com que os raios atingem o objeto, esses raios podem divergir e nenhum retornar aos sensores. Essa limitação dos sonares praticamente impede a detecção de objetos pequenos ou estreitos e também de objetos esféricos. Com freqüência, os sonares não conseguem detectar objetos mais próximos que 200mm e paredes quando estas estão a 45° da normal do sonar.

O software utilizado para manipulação dos movimentos do robô Pioneer 1 é o Saphira da *Activ*Media, Inc., versão 6.1. Este software vem acompanhado de uma biblioteca de funções em linguagem C, sendo compatível com Microsoft Windows 95 / NT, FreeBSD, Linux e UNIX. Para versões Windows95/NT o compilador deve ser somente o Microsoft Visual C/C++. O software Saphira também vem acompanhado de um simulador para robôs Pioneer, no qual é possível montar ambientes estáticos e observar os movimentos do robô.

Das funções da API Saphira, somente foram utilizadas as que realizavam o con-

trole direto de movimentos do robô e da garra. A leitura dos sonares pode ser realizada executando uma função que retorna a leitura de um sonar específico ou utilizando *buffers* de leitura internos ao Saphira, os quais armazenam as últimas leituras realizadas.

Para armazenar os valores dos sonares existem três *buffers*: um para os sonares frontais, um para a lateral direita e um para a lateral esquerda. Para ler os dados destes *buffers* existem as funções de ocupação *sfOccBox* e *sfOccPlane*. No uso destas funções é necessário manter em mente que os parâmetros estão em coordenadas LPS, ou seja, as distâncias são tomadas a partir do centro do robô. Estas funções são utilizadas para verificar se existem objetos próximos do robô, sendo as coordenadas LPS muito úteis neste caso.

A função **sfOccbox** realiza leituras somente sobre o *buffer* dos sonares frontais, considerando um retângulo centrado em cx, cy com altura h e largura w. Ela retorna a distância em milímetros do ponto mais próximo do centro do robô na direção do eixo X ou do eixo Y. A utilização da função é mostrada na Figura 3.2, utilizando o sistema de coordenadas LPS.

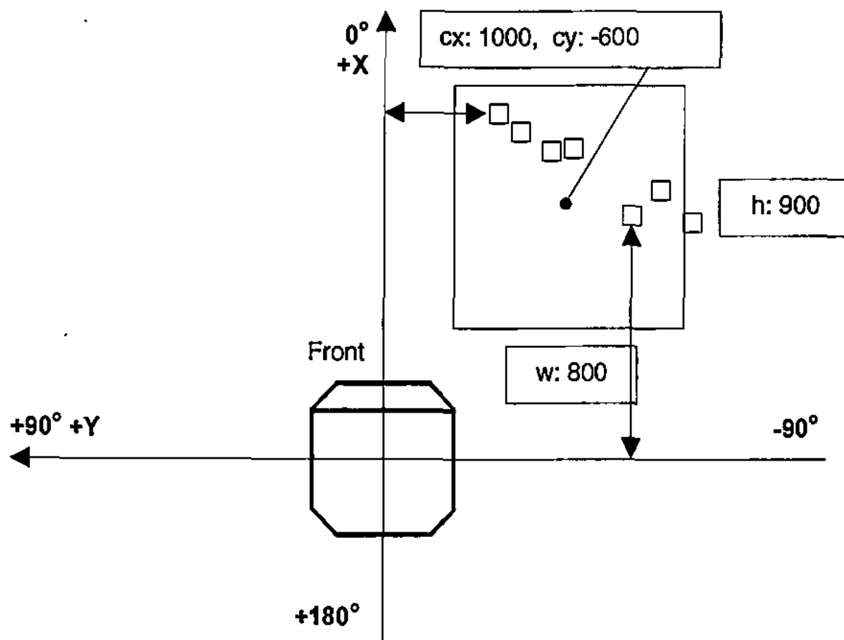


Figura 3.2: Retângulo de atuação para a função *sfOccBox*

A função **sfOccplane** é ligeiramente diferente da função anterior. Ela considera um retângulo infinito definido por três lados: uma linha perpendicular à direção em questão e dois limites laterais. A Figura 3.3 mostra uma área definida na direção do eixo X, com limites laterais indicados por $S1$ e $S2$. Esta função retorna somente a distância do ponto mais próximo ao robô para o eixo especificado. O motivo da utilização desta função é recuperar as leituras dos sonares laterais, com a vantagem de se poder escolher qual dos *buffers* será utilizado: frontal, lateral ou ambos.

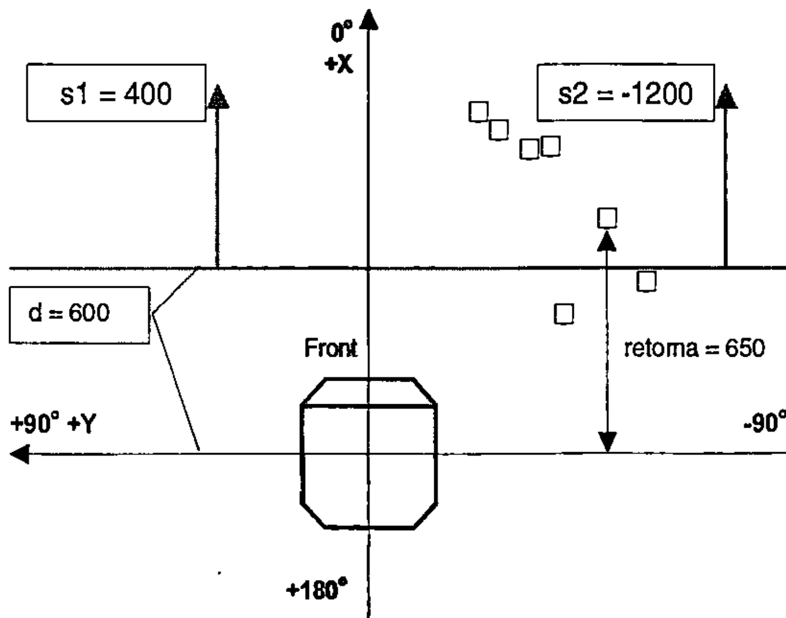


Figura 3.3: Retângulo de atuação para a função *sfOccPlane*

Uma descrição detalhada sobre estas e outras funções são encontradas no manual do software Saphira [Konolige, 1997].

Os sensores do robô servem para lhe dar uma noção do mundo real. É através das informações recebidas pelos sonares que o ambiente é mapeado em um conjunto de estados necessários aos algoritmos de aprendizado. Alguns mapeamentos de estados do ambiente são apresentados no próximo capítulo, assim como o resultado dos algoritmos Q-*learning*, R-*learning* e H-*learning* para cada um desses mapeamentos.

Capítulo 4

Experimentos com os Algoritmos: Q-*learning*, R-*learning* e H-*learning*

Neste capítulo serão detalhados os passos seguidos na modelagem de uma tarefa de aprendizado em um modelo MDP finito. O mesmo modelo MDP será usado nos três algoritmos e então será apresentada uma comparação de desempenho dos algoritmos. Posteriormente, será proposto um novo algoritmo de aprendizado que incorpora ao R-*learning*, conceitos de lógica *fuzzy*. Este algoritmo foi denominado por nós de R'-*learning*.

A tarefa escolhida para ser modelada como um problema de aprendizado com reforço foi fazer o robô aprender a navegar em um ambiente desconhecido evitando obstáculos. Para adquirir um modelo MDP que melhor se ajustasse à tarefa proposta foram realizados vários experimentos utilizando o algoritmo Q-*learning*. A escolha do Q-*learning* se deve ao fato deste ser o mais simples dos três algoritmos, sendo esperado que se este atingisse um comportamento ótimo os demais métodos também atingiriam.

4.1 Modelagem da Tarefa de Navegação

Para criar um modelo MDP deve-se definir: o conjunto de estados do ambiente, S; o conjunto de ações, $A(s)$; as probabilidades de transições entre os estados, $P(s'|s, a)$; e as recompensas para essas transições, $R(s'|s, a)$. Os conjuntos P e R não são conhecidos, portanto os algoritmos de aprendizado utilizarão somente os conjuntos S, $A(s)$ e r' para definir uma política de controle ótima, onde r' é o conjunto de recompensas imediatas.

Em navegação espera-se que o robô sempre ande para frente e só gire para evitar colisões com obstáculos, portanto todas as vezes que escolher ir para frente o robô deve ser recompensado. Não se espera que o robô ande para trás ou que ele colida com objetos, desta forma estas são situações que o robô deve ser punido. Com isso, define-se o conjunto de ações como: $A = \{$*avançar, girar à direita, girar à esquerda, recuar*$\}$. Com base nas recompensas imediatas utilizadas por Bagnell com o robô "Charm" [Bagnell et al., 1998], foi criado um conjunto de recompensas que pode ser observado na Tabela 4.1. Note que a recompensa imediata pode ser dada tanto sobre as ações como sobre o estados. No caso da tarefa de navegação, se for detectado uma colisão, o robô recebe a punição pela colisão, não importando com qual ação o robô colidiu.

colisão	avançar	girar à direita	girar à esquerda	recuar
-1000	100	50	50	-50

Tabela 4.1: Reforço imediato definindo a tarefa de navegação.

Como já foi visto no capítulo anterior, o robô Pioneer 1 Gripper possui 7 sonares e sensores de colisão. É através da leitura destes sensores que robô deverá determinar o seu estado no ambiente. Para realizar este mapeamento considere uma função $f(d_i)$ que classifica a distância, d, retornada pelo sonar, i, em noções de distância. Estas noções de distância são representadas por números binários, sendo estes diretamente proporcional à distância do objeto detectado pelo sonar.

O estado do ambiente, s, será obtido através da concatenação dos os valores binários de $f(d_i)$, ou seja, $s = f(d_0)f(d_1)f(d_2)...f(d_6)$. Um estado deve ser acrescentado

para representar o estado de colisão, pois este estado não é detectado através dos sonares. Este mapeamento de leitura dos sonares em estados do ambiente pode ser visualizado na Figura 4.1.

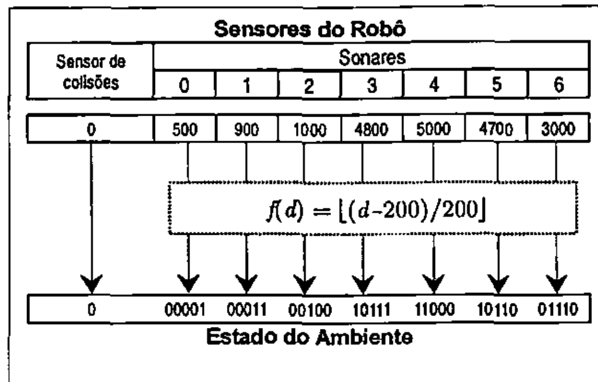


Figura 4.1: Mapeamento da leitura dos sonares em um estado do ambiente.

Através do número de classes, dado pela função $f(d)$, e do número de sonares que estão sendo considerados é possível obter o número total de estados do ambiente como é mostrado na equação (4.1).

$$n^o.estados = n^o.classes^{n^o.sonares} + 1. \tag{4.1}$$

Neste ponto surge um dilema, pois quanto maior o número de classes melhor será a representação do mundo real em estados do ambiente e quanto menor o número de estados do ambiente mais rápido será obtido uma convergência para uma política de controle ótima. A seguir serão apresentadas algumas estratégias seguidas a fim de alcançar um mapeamento que represente bem o mundo real e que não tenha um número muito elevado de estados do ambiente.

Nos experimentos realizados está sendo considerando que cada ação tem um tempo de 100ms para ser executada, após esse tempo, o robô cancela os movimentos e verifica através da leitura dos sonares em qual estado ele se encontra.

A **primeira tentativa** foi considerar $f(d) = \lfloor (d - 200)/300 \rfloor$. Nesta função, o

intervalo de leitura do sonar (200 à 5000mm) é dividido em partes de 300. A subtração $(d-200)$ é colocada devido às leituras começarem em 200mm. Nesta representação tem-se um número de classes igual à $f(5000)$, ou seja, 16 classes, em um total de 16^7 estados do ambiente. O número de estados é muito alto e tentar um intervalo maior que 300mm só iria piorar a precisão do modelo e conseqüentemente do aprendizado.

Uma **segunda tentativa** foi enfatizar o objetivo do robô: navegar evitando colisões. Nesta tentativa foi montado um modelo de ambiente onde se considera somente objetos a menos de 1.1m, as leituras cujas distâncias ultrapassarem este valor serão colocadas em uma única classe, como é mostrado na Tabela 4.2.

d (mm)	$f(d)$ (binário)
000 a 400	00
401 a 700	01
701 a 1100	10
1101 a 5000	11

Tabela 4.2: Classificação das leituras priorizando objetos próximos.

Esta representação utiliza 4 classes, ou seja, 4^7 estados do ambiente. Este modelo foi implementado para os algoritmos Q-*learning*, R-*learning* e H-*learning*. No entanto, devido ao grande número de estados do ambiente não houve memória suficiente para alocar o conjunto de variáveis necessárias ao algoritmo H-*learning*.

A seguir são apresentados os gráficos de aprendizado para os algoritmos Q-*learning* e R-*learning*. Da Figura 4.2 à Figura 4.4 é utilizada uma política de escolha de ações 100%-gulosa e como já foi dito anteriormente, isto pode causar uma convergência para um máximo local.

O que se espera de uma política ótima, neste caso, é que esta execute a ação *avançar* e somente gire para evitar as colisões. Portanto, é de se esperar que os gráficos de uma política ótima tenham uma convergência para o valor da recompensa da ação *avançar*,isto é, igual a 100.

Como se pode notar, o gráfico da Figura 4.2 foi o único que convergiu para uma

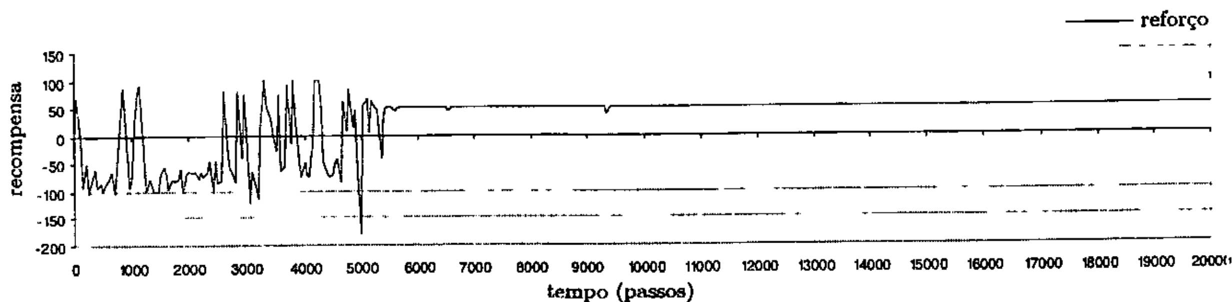


Figura 4.2: Q-*learning* com $\alpha = 0.2$, $\gamma = 0.50$ e $\epsilon = 100\%$

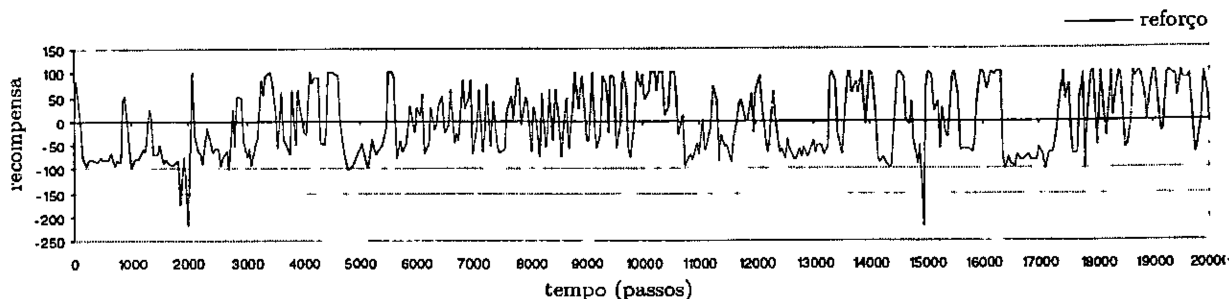


Figura 4.3: Q-*learning* com $\alpha = 0.2$, $\gamma = 0.99$ e $\epsilon = 100\%$

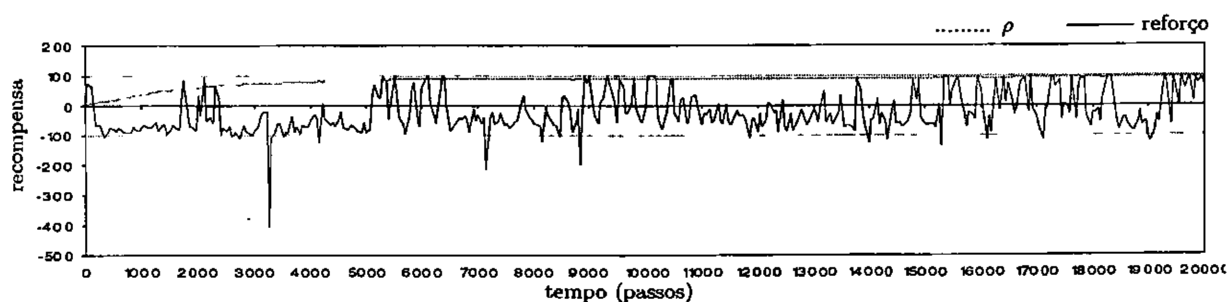


Figura 4.4: R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 100\%$

política que evita colisões. A política encontrada considerou que as ações *girar à direita* e *girar à esquerda* são as melhores, fazendo com que o robô fique girando sobre seu próprio eixo. Isto significa que a política encontrada não é ótima, pois o robô deixou de executar a ação *avançar*. Uma possível causa de não se ter obtido uma política ótima é a utilização de uma política de escolha de ações 100%-gulosa, podendo ser solucionada incluindo-se uma porcentagem de ações aleatórias como mostrado nos gráficos da Figura 4.5 à Figura 4.7.

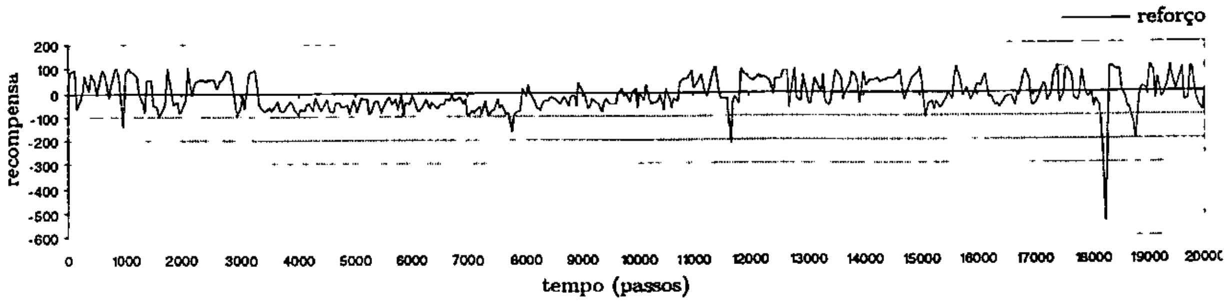


Figura 4.5: Q-*learning* com $\alpha = 0.2$, $\gamma = 0.50$ e $\epsilon = 95\%$

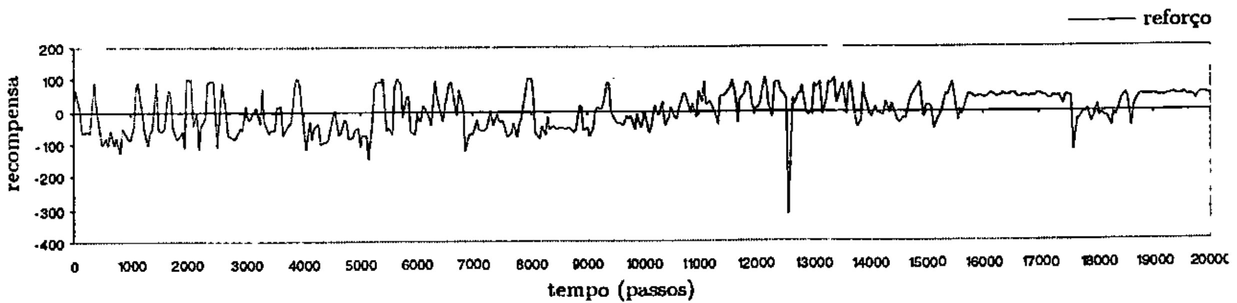


Figura 4.6: Q-*learning* com $\alpha = 0.2$, $\gamma = 0.99$ e $\epsilon = 95\%$

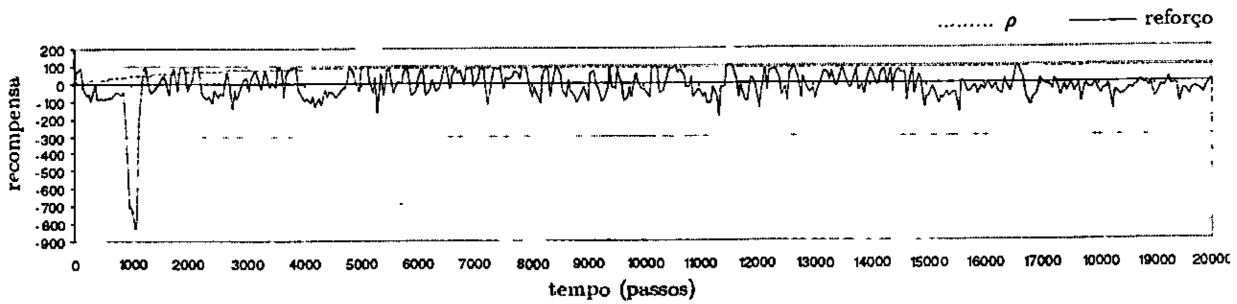


Figura 4.7: R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 95\%$

Mesmo com a introdução de escolhas aleatórias, os algoritmos não obtiveram uma convergência para uma política ótima.

Uma **terceira tentativa** foi considerar apenas duas classes, uma indicando a possível existência de um objeto e outra indicando passagem livre. Esta representação, indicada na equação (4.2), utiliza apenas 2^7 estados, mas em contrapartida não é uma

representação fiel ao ambiente real.

$$f(d) = \begin{cases} 0 & \text{se } d \leq 500 \\ 1 & \text{caso contrário} \end{cases} \quad (4.2)$$

Esta classificação foi utilizada para os três algoritmos: Q-*learning*, R-*learning* e H-*learning*. Os três algoritmos obtiveram convergência para uma política que aboliu a ação *avançar*, ou seja, convergiram para um máximo local (Figura 4.8 a Figura 4.10).

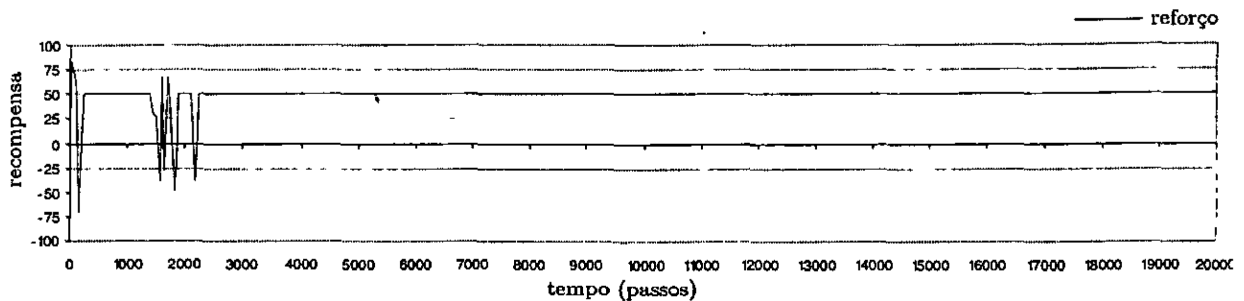


Figura 4.8: Q-*learning* com $\alpha = 0.2$, $\gamma = 0.75$ e $\epsilon = 100\%$

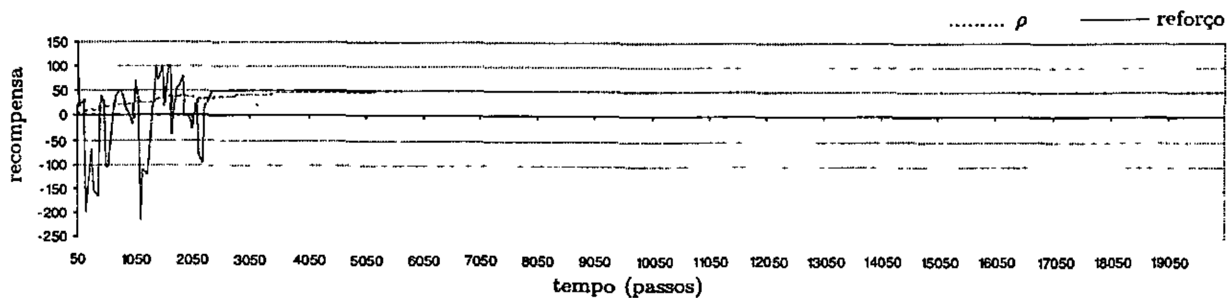


Figura 4.9: R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 100\%$

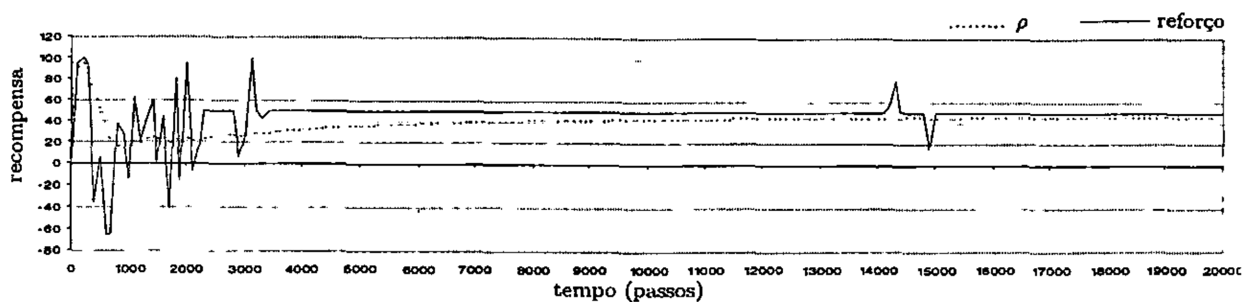


Figura 4.10: H-*learning* com $\epsilon = 100\%$

Para evitar os máximos locais foram realizados os mesmos testes, mas adotando uma política 95%-gulosa para a escolha das ações. Os resultados destes testes podem ser observados através das Figuras 4.11 à 4.13. Observa-se que o algoritmo R-*learning*, Figura 4.12, não convergiu para nenhuma política e os outros dois algoritmos continuaram em máximo local.

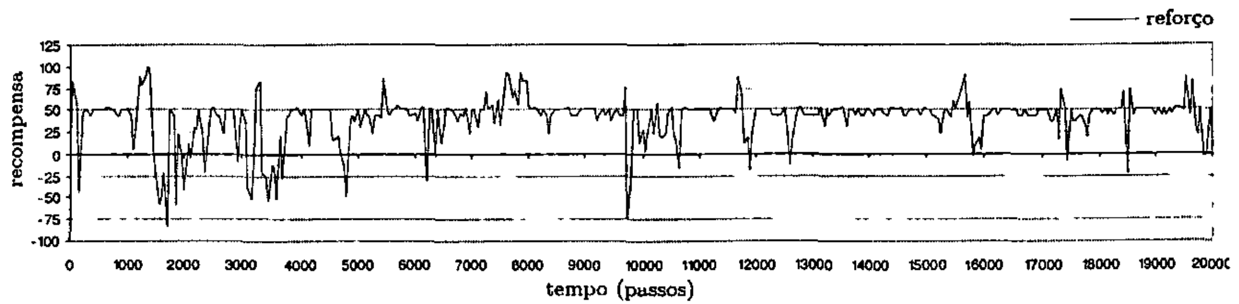


Figura 4.11: Q-*learning* com $\alpha = 0.2$, $\gamma = 0.75$ e $\epsilon = 95\%$

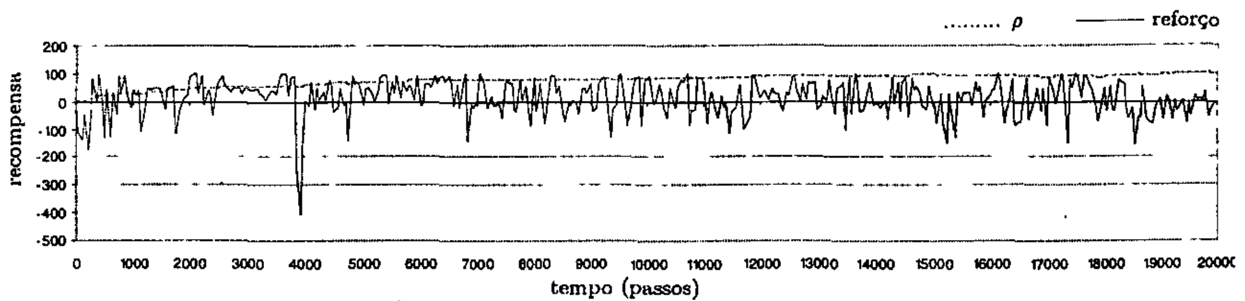


Figura 4.12: R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 95\%$

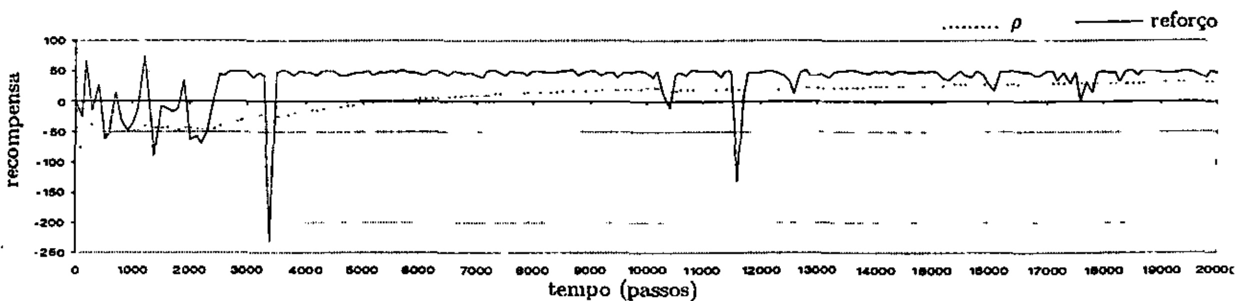


Figura 4.13: H-*learning* com $\epsilon = 95\%$

Na **quarta tentativa**, os sinais recebidos pelos sete sonares foram classificados como uma distância à esquerda, uma distância à direita e uma distância à frente, para reduzir o número de estado do ambiente. A distância dos objetos à frente foi adotada

como sendo a distância mínima recebida pelos cinco sonares frontais, Figura 4.14. Então, na realidade estão sendo classificados apenas 3 sinais. A classificação dos sinais é feita em 5 classes, segundo os dados da Tabela 4.3. Desta forma, tem-se um total 5^3 estados do ambiente.

d (mm)	$f(d)$ (binário)
000 a 400	000
401 a 600	001
601 a 800	010
801 a 1000	011
1001 a 5000	100

Tabela 4.3: Classificação das leituras priorizando objetos próximos em cinco classes.

Com este modelo, representado esquematicamente na Figura 4.14, foi alcançada uma convergência máxima, ou seja, uma política de controle ótima (Figura 4.16 a Figura 4.18). A política encontrada faz com que o robô navegue pelo ambiente evitando colisões. Todavia, os movimentos do robô não parecem tão inteligentes, desviando bruscamente dos obstáculos. Isto acontece devido, talvez, à própria simplificação realizada nos cinco sonares frontais.

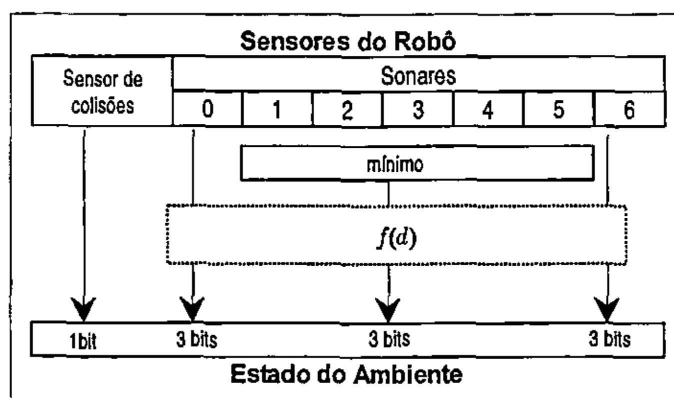


Figura 4.14: Estado do Ambiente mapeado através da simplificação das leituras dos setes sonares em três sinais de entrada, diminuindo sensivelmente o número de estados.

Da Figura 4.15 à Figura 4.18 são mostrados os gráficos para os três algoritmos testados, podendo-se notar as seguintes características no comportamento desses algoritmos:

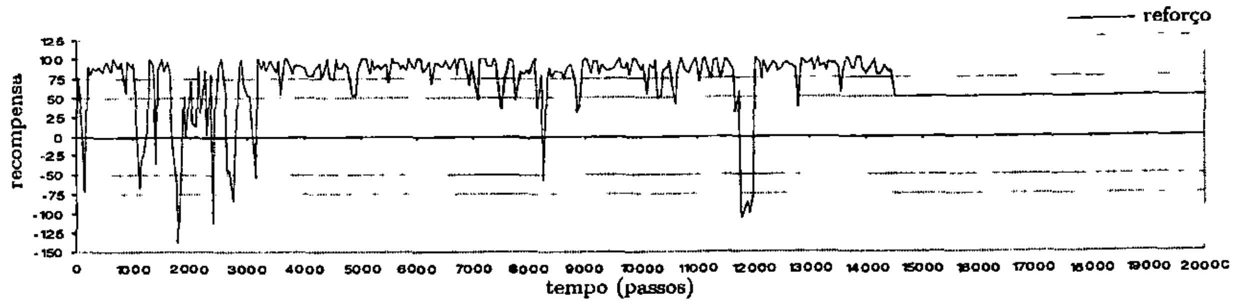


Figura 4.15: Q-*learning* com $\alpha = 0.2$, $\gamma = 0.99$ e $\epsilon = 100\%$

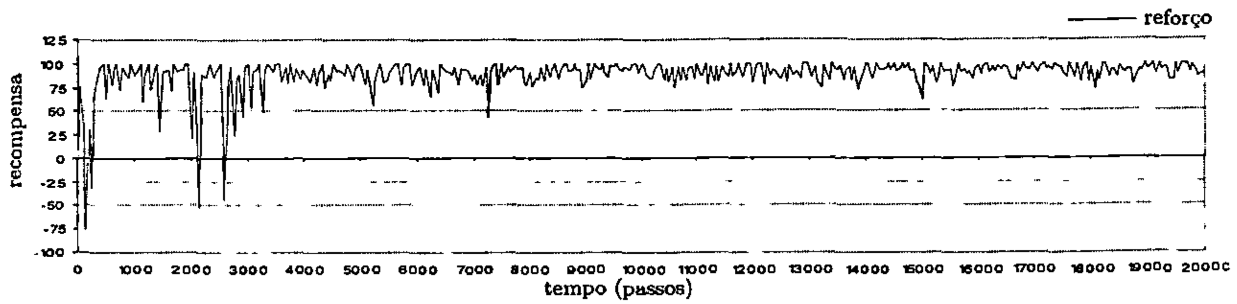


Figura 4.16: Q-*learning* com $\alpha = 0.2$, $\gamma = 0.75$ e $\epsilon = 100\%$

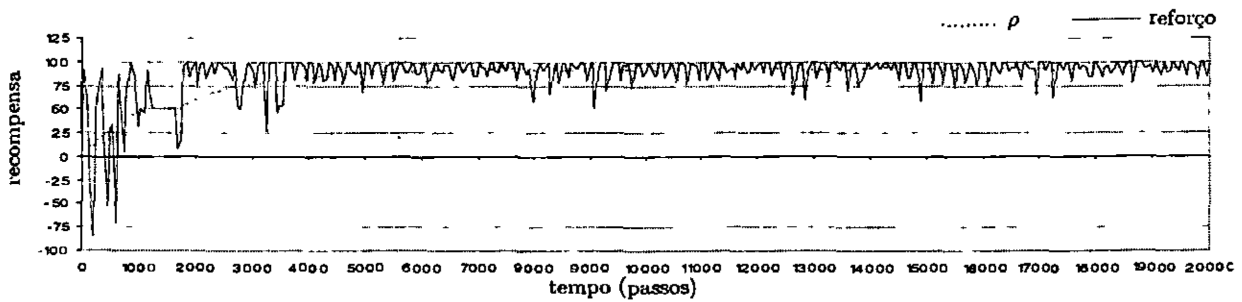


Figura 4.17: R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 100\%$

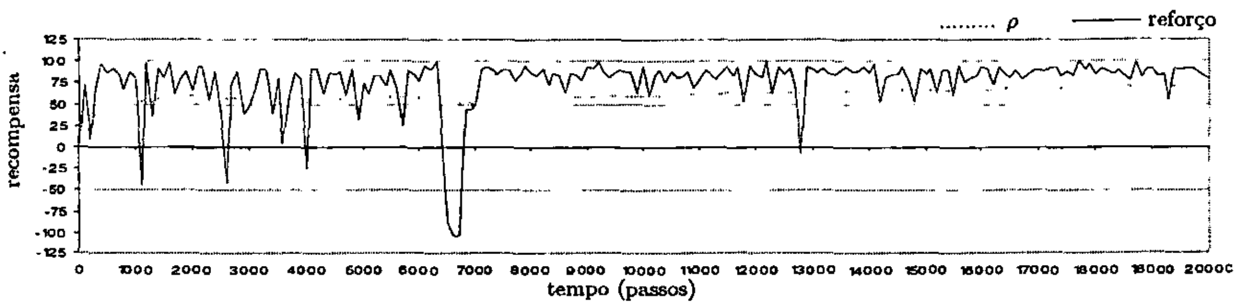


Figura 4.18: H-*learning* com $\epsilon = 100\%$

1. O algoritmo Q-*learning* é muito sensível a variável γ e simples variações refletem em resultados bem diferentes. Como poder ser observado na Figuras 4.15, a utilização de $\gamma = 0.99$ provocou valores negativos na altura dos passos 8000 e 12000, onde uma convergência do algoritmo já era esperada; $\gamma = 0.99$ ainda provocou uma convergência do aprendizado para um máximo local. Quando comparado a $\gamma = 0.75$, Figura 4.16, verifica-se que a partir do passo 2500 não apareceram mais valores negativos e que a convergência foi atingida próximo ao passo 3000. A maior vantagem está no fato de que com $\gamma = 0.75$ encontrou-se uma política ótima.

2. O algoritmo H-*learning* convergiu para uma política ótima a partir de 7000 passos, como pode ser observado na Figura 4.18. Dentre os algoritmos que convergiram, para este problema de aprendizado, este foi o que levou mais tempo.

3. Os algoritmos Q-*learning* e R-*learning* atingiram a convergência próxima de 3000 passos (Figura 4.16 e Figura 4.17). Portanto, para verificar quais destes algoritmos apresentaram melhor desempenho foi feita uma comparação com o número de colisões durante o aprendizado. Na Figura 4.19 é possível perceber que o algoritmo R-*learning* diminuiu o número de colisões a partir de 1000 passos, enquanto que o algoritmo Q-*learning* só vem a fazê-lo em 3000 passos.

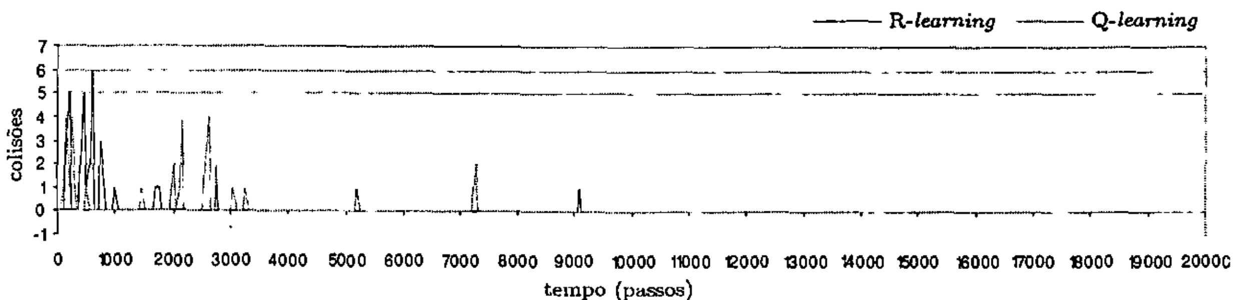


Figura 4.19: Q-*learning* × R-*learning*: colisões durante o aprendizado.

Neste experimento, o desempenho dos algoritmos Q-*learning* e R-*learning* foram muito parecidos. O algoritmo Q-*learning* teve de ser testado utilizando vários valores para γ até encontrar um valor que atendesse as nossas expectativas. Para o algoritmo R-*learning* foi bem mais fácil obter um valor para β, pois o ajuste de β se faz ao observar

a convergência de ρ. Portanto, o valor de $\beta = 0.001$ foi definido logo no início dos experimentos e foi mantido para todos os testes com o algoritmo R-*learning*. Quanto a taxa de aprendizado, α, ambos algoritmos são sensíveis a sua modificação, sendo que os melhores valores encontrados estão entre $\alpha = 2$ e $\alpha = 2.5$. Foi observado que quanto menor o valor de α mais tempo leva o aprendizado. Por outro lado, valores altos influem no cálculo da recompensa esperada de tal modo que não se consegue a convergência do aprendizado. O algoritmo H-*learning* tem apresentado resultados muito bons na literatura. Neste experimento, apesar de ter convergido para uma política ótima, ele teve um desempenho pior que os demais. O H-*learning* não possui variáveis como α e β do R-*learning*, por este motivo ele depende exclusivamente do conjunto de recompensas imediatas, estados e ações para construir o modelo para uma política ótima, sendo bastante sensível às modificações destes conjuntos. Por estes motivos, o algoritmo R-*learning* foi considerado por nós como o melhor algoritmo para resolver o problema de navegação.

Buscando uma melhor forma de classificar as leituras dos sonares em estados do ambiente, foi por nós proposto uma classificação de leituras dos sonares através do uso de funções *fuzzy* [Shaw and Simões, 1999]. O modelo utilizando lógica *fuzzy* foi incorporado ao algoritmo R-*learning*, gerando o novo algoritmo R'-*learning* que é apresentado na próxima seção.

4.2 Modelo Incorporando Conceitos de Lógica *Fuzzy*

As funções de classificação, $f(d)$, definidas nas seções anteriores, consideram igualmente todos os valores de uma única classe, sem diferenciar os valores que estão próximos às vizinhanças de outras classes. Por não ter distinção entre os elementos de uma classe a recompensa recebida por eles será a mesma. Valores vizinhos de outras classes deveriam ter um grau de incerteza, avisando ao algoritmo de aprendizado que a classificação feita não tem certeza do mapeamento do estado do ambiente. Desta forma, o algoritmo poderia ponderar a recompensa imediata para estados do ambiente mapeados através de valores localizados nas vizinhanças das classes.

Para resolver este problema é proposto um novo algoritmo denominado R'-*learning*. Neste algoritmo as leituras dos sonares são classificadas através de funções *fuzzy* para que valores muito próximos de outras classes não sejam considerados tão precisos quanto os demais. Para classificar a distância recebida por um sonar, foram definidas quatro funções *fuzzy*, cada uma representando uma noção de distância, como pode ser observado na Figura 4.20.

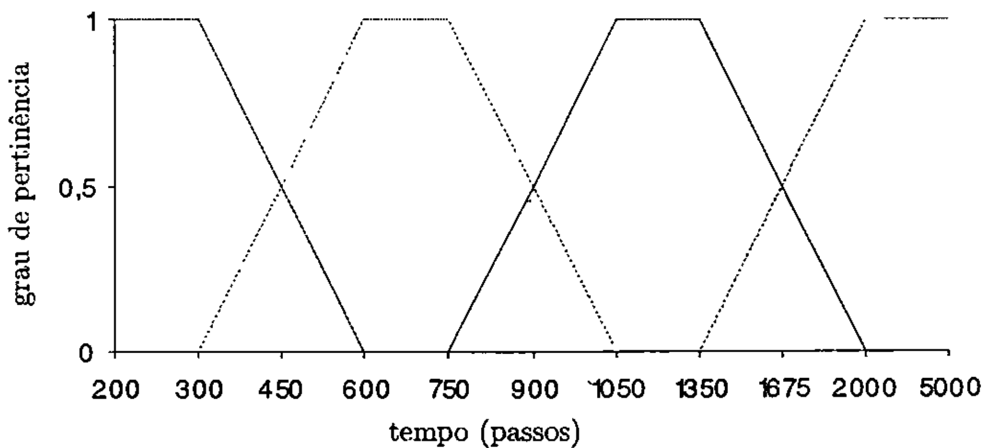


Figura 4.20: Classificação da leitura dos sonares em quatro funções *fuzzy*.

Para o mapeamento dos estados do ambiente, assim como na Figura 4.14, serão considerados apenas três sinais de leituras dos sonares, onde a distância à frente está sendo considerada como a distância mínima entre os cinco sonares frontais. Além de classificar os três sinais de leitura dos sonares, as funções *fuzzy* também retornam o grau de pertinência para cada classe. Com estes três novos valores foi criada uma nova variável, φ (*grau de certeza*), proposta aqui pela necessidade de considerar o fato que o robô possa estar na fronteira de dois conjuntos *fuzzy* vizinhos. A variável φ é definida como sendo a média aritmética dos graus de pertinência recebidos das funções *fuzzy* que codificam os sinais de entrada.

O mapeamento dos estados do ambiente utilizando funções *fuzzy* pode ser observado na Figura 4.21, onde é mostrada a classificação dos três sinais de distância e inserção da nova variável φ.

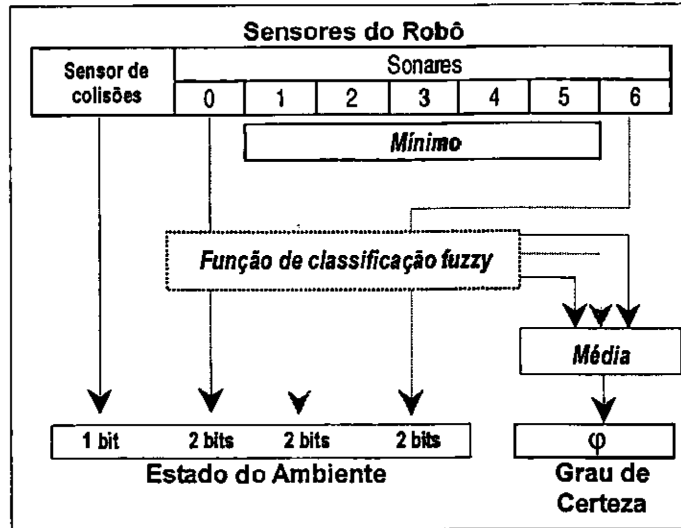


Figura 4.21: Definição do estado do ambiente incorporando lógica *fuzzy*.

A diferença entre os algoritmos R'-*learning* e R-*learning*, está no fato do primeiro considerar a variável φ, para ponderar o reforço imediato, r. Desta forma, o novo valor da recompensa imediata, r', foi definido como segue:

$$r' = \begin{cases} r & \text{se o robô tem certeza de onde está} \\ \varphi r & \text{caso contrário.} \end{cases} \tag{4.3}$$

De acordo com a Figura 4.21, a cada leitura dos sonares, um estado do ambiente com respectivo grau de certeza é computado. Desta forma, a descrição do R'-*learning* é apresentada no Algoritmo 4.1.

Ambos algoritmos, R-*learning* e R'-*learning*, foram testados usando o simulador do robô Pioneer1, adotando $\alpha = 0.2$ e $\beta = 0.001$. Para se estabelecer as primeiras noções de comportamento de acordo com [Kaelbing and Littman, 1996], durante os primeiros 2000 passos o robô foi controlado via teclado e os outros 8000 passos via treinamento de cada algoritmo. O desempenho do método de aprendizado com reforço é sensível à exploração e ao conjunto de recompensas. Para estratégia de exploração foram utilizadas escolhas aleatórias tanto para evitar máximos locais, quanto para evitar que as ações escolhidas nos primeiros 2000 passos fossem tomadas como ações ótimas. Portanto, as ações aleatórias

Algoritmo 4.1 R'-*learning*

1: inicialize ρ e $R(s,a)$ arbitrariamente
2: **repita**
3: $s \leftarrow$ estado atual
4: $a \leftarrow$ ação baseda numa política de comportamento ϵ-gulosa
5: execute a ação $a \in A(s)$
6: receba a recompensa r
7: receba o próximo estado s' e o seu grau de certeza φ
8: $r' \leftarrow \varphi r$
9: $R(s,a) \leftarrow R(s,a) + \alpha[r' - \rho + \max_{a'} R(s',a') - R(s,a)]$
10: se $R(s,a) = \max_a R(s,a)$ **então**
11: $\rho \leftarrow \rho + \beta[r' - \rho + \max_{a'} R(s',a') - \max_a R(s,a)]$
12: **fim-se**
13: **até que** um número máximo de passos seja atingido

servem para fazer com que qualquer ação tenha a possibilidade de se tornar ótima. Para isso, foi utilizada uma política de escolha de ações 95%-gulosa. Ambos os algoritmos utilizam o conjunto de recompensas para as ações, mostrado na Tabela 4.4 e para o estado de colisão foi definido uma punição de -700.

Ações	avançar	girar à direita	girar à esquerda	recuar
Recompensa	100	50	50	-50

Tabela 4.4: Recompensas para os algoritmos R-*learning* e R'-*learning*.

Os gráficos de aprendizado para os algoritmos R-*learning* e R'-*learning* são mostrados na Figura 4.22 e Figura 4.23, respectivamente. Na Figura 4.24 é possível notar que o número de colisões através do algoritmo R'-*learning* é menor que o obtido usando o algoritmo R-*learning*. Desta forma, a nossa proposta de incorporação de lógica *fuzzy* no algoritmo R-*learning*, para mapear os estados do ambiente e calibrar o valor da recompensa imediata, propiciou melhorias no aprendizado da tarefa de navegação.

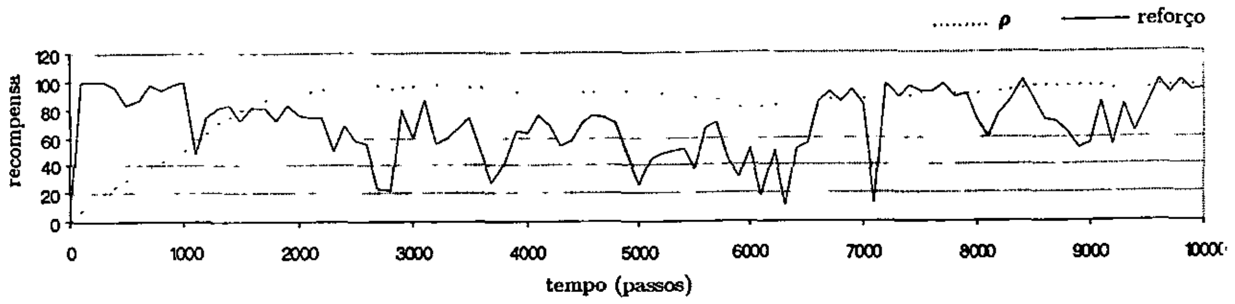


Figura 4.22: R-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 95\%$

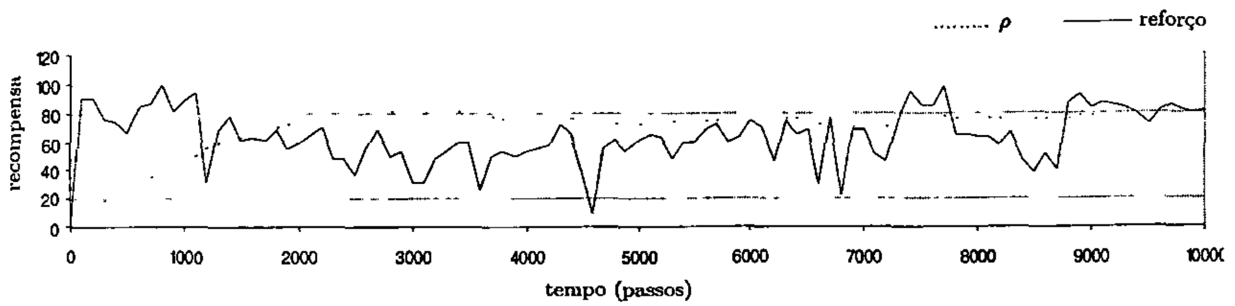


Figura 4.23: R'-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 95\%$

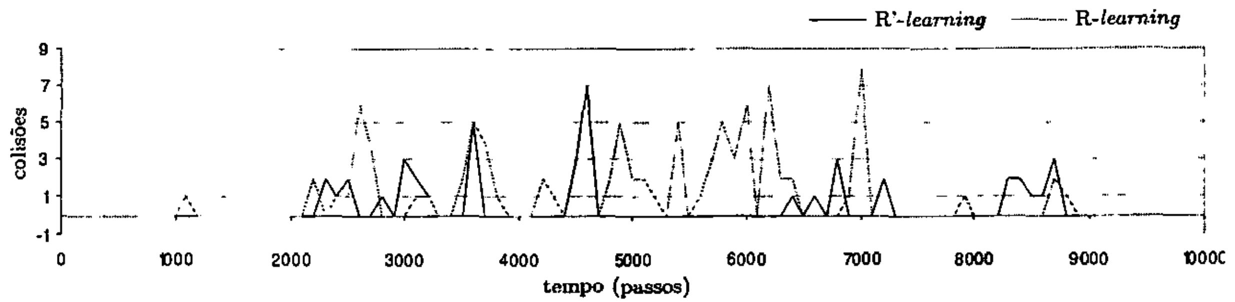


Figura 4.24: R-*learning* × R'-*learning*: colisões durante o aprendizado.

Capítulo 5

Aplicação utilizando R'-*learning*

Para escolher uma aplicação real para ao robô, considerou-se o fato do robô não depender de um mapa do ambiente. Desta forma, a tarefa implementada consiste em fazer com que o robô navegue por um ambiente desconhecido, procurando por pequenos objetos que possam ser recolhidos e levados a um local específico (lixeira).

Para implementar esta tarefa, foi escolhido o algoritmo R'-*learning*, pois foi este o algoritmo que apresentou os melhores resultados no aprendizado da tarefa de navegação e por possuir um conjunto de variáveis de mais fácil configuração que os outros algoritmos implementados no capítulo anterior.

Os sonares não são os melhores sensores para se reconhecer objetos, por este motivo algumas vezes o robô é enganado por seus sonares e tenta pegar objetos grandes como cadeiras e mesas. Para levar os objetos até uma lixeira, a aplicação utiliza uma rotina de *planning* de tempo-real, proposta por [Borenstein and Koren, 1989], que utiliza conceitos de força de repulsão para escolher o caminho até o ponto de destino. Neste capítulo será proposta a utilização da força de repulsão para mapear os estados do ambiente. Este mapeamento reduz consideravelmente o número de estados do ambiente do modelo MDP, por este motivo o modelo do ambiente nesta aplicação não mais mapeia as leituras dos sonares em conjuntos *fuzzy* e sim considera o ângulo e a amplitude da força de repulsão

entre o robô e os obstáculos.

A tarefa proposta foi trabalhada em módulos e seguem o esquema do fluxograma apresentado na Figura 5.1. Nas seções deste capítulo serão detalhados: o módulo do "cálculo da força de repulsão", a implementação do questionamento "reconhece objeto" e o seu módulo de *planning*. Os questionamentos "colisão", "risco de colisão" e "obstáculo" serão descritos juntamente com seus módulos na seção 5.3. Ao final deste capítulo serão apresentados o modelo MDP e o um gráfico de conversão para uma política ótima, obtido através do aprendizado com o algoritmo R'-*learning* para problema proposto.

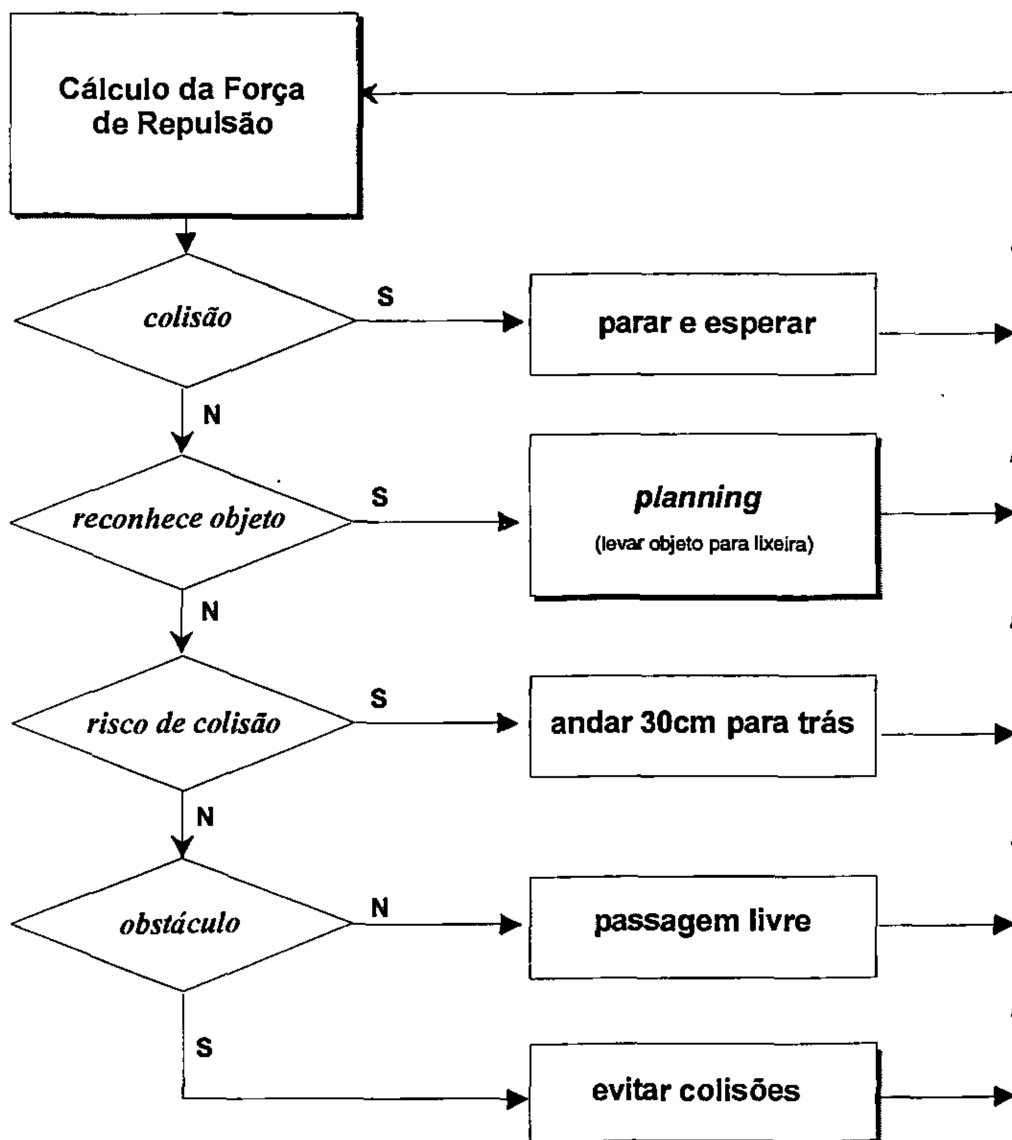


Figura 5.1: Representação dos módulos da aplicação utilizando aprendizado com reforço.

5.1 Cálculo da Força de Repulsão

Este é o módulo principal do sistema (Figura 5.1), pois a força de repulsão é que responde os questionamentos "risco de colisão" e "obstáculo". Os módulos "*planning*" e "evitar colisões" são totalmente dependentes da força de repulsão, pois é sobre ela que são tomadas as decisões de escolha de ações para ambos os módulos.

O cálculo da força foi baseado no trabalho de Borenstein e Koren, no qual o mapeamento do ambiente é feito através de uma tabela de certezas [Borenstein and Koren, 1989]. Para criar esta tabela a área de trabalho do robô foi dividida em muitos quadrados (denominados células), formando uma tabela. Cada célula (i,j) desta tabela possui um nível de certeza $C(i,j)$ que indica a possibilidade de haver um objeto naquela área. Quanto maior o valor de $C(i,j)$, maior o nível de confiança que a célula está ocupada por um objeto. Através de uma varredura na tabela de certezas, calcula-se a força de repulsão, $\vec{F}$, para cada célula (i,j). A equação (5.1) mostra o vetor força, $\vec{F}$, representado por seu módulo $|\vec{F}|$ e sua direção θ_F.

$$|\vec{F}| = \frac{F_{cr}C(i,j)}{d^2} \qquad \theta_F = \arctan(d_y/d_x), \tag{5.1}$$

onde F_{cr} é uma força constante de repulsão; $d(i,j)$ é a distância entre a célula (i,j) e o robô; $C(i,j)$ é o nível de confiança da célula (i,j) e (d_x, d_y) são as projeções da distância $d(i,j)$ nos eixos x e y, respectivamente.

Note que a força, $\vec{F}$, é inversamente proporcional à distância ao quadrado, fazendo com que seja muito maior quando uma célula próxima ao robô estiver ocupada. O quadrante do ângulo retornado por arctan é determinado pelos sinais de d_y e d_x, ou seja, $0^o \leq \arctan(d_y/d_x) \leq 360^o$.

Para calcular a força resultante, $\vec{F_r}$, deve-se realizar a soma vetorial das forças de cada célula, dada por:

$$\vec{F_r} = \sum_{\forall(i,j)} \vec{F} \tag{5.2}$$

A tabela de certezas acaba sendo na realidade, um mapa de todas as posições do ambiente onde existem objetos. Desta forma, quanto maior for o ambiente, maior será o número de posições da tabela. A aplicação neste trabalho é fazer o robô navegar em um ambiente real, incluindo objetos móveis, a força de repulsão foi calculada diretamente sobre as distâncias recebidas pelos sonares, isto é, não foi usado o mapeamento proposto por [Borenstein and Koren, 1989]. Desta forma, a força é calculada pela posição atual dos objetos, não importando se estes são móveis ou fixos.

Em uma primeira instância, propôs-se um cálculo da força de repulsão, $\vec{F}$, através da leitura direta dos sete sonares do robô. A utilização da força resultante $\vec{F_r}$ no módulo de *planning* teve ótimos resultados, quando executada sobre simulador do robô. Os mesmos testes foram realizados para o robô em um ambiente real e para nossa surpresa, o robô colidia muito nas laterais e não conseguia contornar os obstáculos. Para contornar este problema seriam necessários mais sonares laterais ou que o robô lembrasse momentaneamente dos objetos pelo qual passou.

Como alternativa para este problema utilizou-se as funções sfOccBox e sfOccPlane (Capítulo 3) para recuperar informações de leituras anteriores dos sonares frontais e laterais, respectivamente. Estas informações foram colocadas em uma tabela de ocupação binária marcando as posições de possíveis objetos. Cada célula da tabela representa um espaço de 100mmx100mm, como o sistema das funções está em LPS, a tabela também terá uma representação em LPS. Esta é uma grande vantagem, pois para o cálculo da força o que importa é a distância do objeto ao robô.

O tamanho da tabela de ocupação é definido através de três argumentos: a distância máxima percebida pelas funções sfOccBox e SfOccPlane é de 2000mm; cada célula têm 100mmx100mm; devido ao sistema LPS, o robô sempre ocupa uma posição no centro da tabela. Portanto, deve-se reservar 20 posições para cada direção: à frente, atrás, à direita e à esquerda, tendo desta forma, uma tabela de 41x41 posições.

Considerando que a primeira posição da tabela é o ponto $(0,0)$, então o robô estará posicionado no ponto $(20,20)$. Desta forma, as componentes da distância, d, entre o robô

e uma célula (i, j) são calculadas como:

$$d_x = (20 - i) * 100$$
$$d_y = (20 - j) * 100.$$

e a força de repulsão para cada célula (i, j) é definida pela equação (5.3).

$$|\vec{F}| = \frac{F_{cr}}{d^2} \qquad \theta_F = \arctan(d_y/d_x) \tag{5.3}$$

Para esta aplicação o valor de F_{cr} foi definido de forma que se $d < 500mm$ então $|\vec{F}| < 1$, para que essa sentença fosse verdadeira, foi assumido $F_{cr} = 250000$. Da mesma forma que a equação (5.2), o cálculo da força resultante, $\vec{F_r}$, é a soma vetorial das forças de cada célula (i, j).

As utilização da força resultante $\vec{F_r}$ no algoritmo de *planning* será detalhada na seção 5.2 e a construção de um modelo MDP que modela os estados do ambiente através de $\vec{F_r}$ será apresentada na seção 5.3.

5.2 Reconhece Objetos e *Planning*

Para reconhecimento de pequenos objetos foram utilizadas apenas as informações recebidas pelos cinco sonares frontais. Na Figura 5.1, o questionamento "reconhece objeto", não só verifica se existe um objeto estreito como também posiciona o robô em sua direção. Quando o objeto estiver à frente do robô o módulo "*planning*" é ativado, fazendo com que o robô recolha o objeto e leve-o até uma lixeira. Desta forma, será considerado que "objeto reconhecido" é um dos estados do ambiente e "levar objeto à lixeira" é a única ação para este estado.

Considerando que os sonares frontais são numerados de 1 à 5, da esquerda para direita, verifica-se qual sonar, s_i, retorna a menor distância, onde $1 \leq i \leq 5$. Seja s_j este

sonar, d_1 a distância entre s_j e s_{j-1}, d_2 a distância entre s_j e s_{j+1}, e D_{min} uma constante que é definida como a distância mínima para considerar dois objetos como distantes. Um objeto é reconhecido como estreito quando d_1 e d_2 forem maiores que D_{min}, como pode ser observado na Figura 5.2(a). A Figura 5.2(b) mostra o reconhecimento de um objeto extenso, pois a condição $d_1 \geq D_{min}$ não foi satisfeita, para o D_{min} adotado.

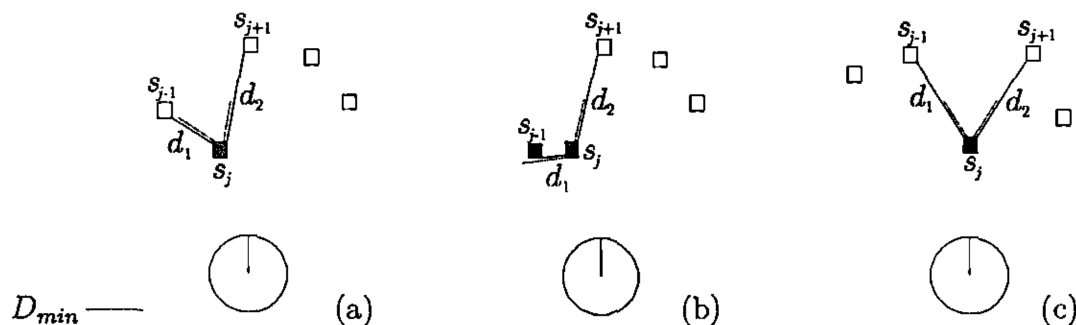


Figura 5.2: Reconhecimento de objetos: (a) objeto estreito, (b) objeto extenso e (c) objeto estreito pronto para ser coletado.

Quanto maior for o valor de D_{min}, maior terá que ser a distância entre o objeto a ser pego e as demais superfícies. Entretanto, para valores pequenos, pequenas oscilações nas leituras dos sonares podem fazer com que superfícies como portas e paredes sejam reconhecidas como objetos estreitos. Neste trabalho, o melhor valor encontrado para D_{min} foi 300mm.

Ao reconhecer um objeto estreito, o robô deverá seguir em sua direção, girando à direita ou à esquerda conforme for necessário, até que o objeto fique exatamente à frente do robô, como é mostrado na Figura 5.2(c).

Devido à falhas nas leituras dos sonares, o reconhecimento de um objeto extenso pode oscilar entre estreito e extenso, dependendo da posição do robô em relação ao objeto. Quando isto ocorre, o robô fica alternando em giros à direita e à esquerda, ora tentando ir ao encontro de um objeto, ora tentando desviar dele. Ao detectar estas oscilações de posições, o questionamento "reconhece objeto" fica inativo por um certo tempo. Desta forma, se o robô não estiver colidido (questionamento "colisão") o próximo questionamento passa a ser "risco de colisão" (Figura 5.1).

Após pegar o objeto, o robô ativa o módulo "*planning*", Figura 5.1, para levar o objeto coletado até uma lixeira. Neste trabalho, o ponto onde o robô começou a se mover foi definido como sendo a "lixeira".

O algoritmo de *planning* está baseado em [Borenstein and Koren, 1989]. Este algoritmo faz com que o robô siga sempre em direção a ponto de destino e quando isto não for possível, faz com que ele siga a parede mais próxima. A rotina "seguir parede" foi adicionada ao algoritmo de *planning* para permitir que o robô saia de "armadilhas" do ambiente, tais como becos, paredes côncavas e até labirintos.

Para fazer com que o robô siga em direção à meta, utiliza-se uma força de atração, $\vec{F_t}$, entre o robô e a meta. A força $\vec{F_t}$ não depende da distância entre o robô e meta, sendo constante durante todo o percurso. A direção da força é dada por σ, sendo calculada através da distância d entre o robô, A, e a meta, M, como pode ser observado na Figura 5.3.

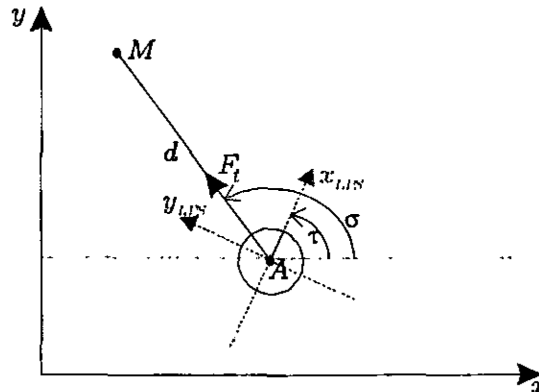


Figura 5.3: Transformação da força de atração para o sistema LPS.

A direção dada por σ pertence ao sistema de coordenadas do mundo, mas a direção de $\vec{F_t}$ deve estar no sistema LPS. Esta transformação de coordenadas é feita descontando-se de σ a direção atual do robô, τ. Desta forma, a força de atração $\vec{F_t}$ é definida como:

$$|\vec{F_t}| = F_{ct} \qquad \theta_{F_t} = \sigma - \tau, \tag{5.4}$$

onde $\sigma = \arctan(d_y/d_x)$; d_x e d_y são as componentes de d nos eixos x e y, respectivamente.

F_{at} é uma força de atração constante, que definirá quão perto o robô poderá chegar dos objetos antes de tentar desviar.

O valor de F_{at} foi definido por $|\vec{F}_r|$ para um objeto detectado a 250mm, ou seja, $F_{at} = 4$. Isto significa que para distâncias menores que 250mm ou para objetos extensos detectados por mais de um sonar a força de repulsão será maior que a força de atração, $|\vec{F}_r| > |\vec{F}_t|$, fazendo com que o robô desvie do obstáculo.

É necessário que o robô desvie dos possíveis obstáculos durante o percurso até a meta. Para realizar esta tarefa, será utilizada a força de repulsão $\vec{F}_r$. Desta forma, uma força resultante, $\vec{R}$, dada pela soma vetorial de $\vec{F}_r$ e $\vec{F}_t$, dará a direção a ser seguida. O cálculo de $\vec{R}$ é mostrado na equação (5.5).

$$\vec{R} = \vec{F}_r + \vec{F}_t \tag{5.5}$$

Quando o robô estiver desviando mais de 90° da direção da meta, $|\theta_{Ft}(-)\theta_R| > 90^o$, o robô deve passar a seguir uma parede. Considere $(-)$ como um operador especialmente definido para dois ângulos a e b (em graus) que retorna a menor diferença rotacional entre a e b, portanto, $0^o \leq |a(-)b| \leq 180^o$.

Para seguir uma parede, uma nova direção deve ser calculada e atribuída à θ_R, como é mostrado na equação (5.6).

$$\theta_R = \begin{cases} \theta_{F_{wr}} - \phi & \text{se a parede está à direita} \\ \theta_{F_{wl}} + \phi & \text{se a parede está à esquerda} \end{cases}, \tag{5.6}$$

onde a força $\vec{F}_{wr}$ é calculada considerando unicamente as leituras feitas pelo sonar da lateral direita, assim como $\vec{F}_{wl}$ considera apenas as leituras do sonar da lateral esquerda do robô. A constante ϕ [graus] é um ajuste à direção da força de repulsão ($\vec{F}_{wr}$ ou $\vec{F}_{wl}$), para que $\vec{R}$ fique paralelo à parede.

Em um caso perfeito, ao seguir uma parede, todos os pontos da parede seriam

percebidos (Figura 5.4 (b)), e para seguir paralelamente à parede o melhor valor para ϕ seria 90°. Entretanto, como só existe um sonar na lateral do robô *Pioneer 1*, a parede é percebida como se fosse um rastro deixado por leituras anteriores. Por este motivo, os pontos que representam a parede estão em sua maioria localizados abaixo do eixo y, como pode ser visto na Figura 5.4 (a). Através de testes exaustivos foi verificado que o melhor valor para ϕ, neste caso, é $\phi = 68°$.

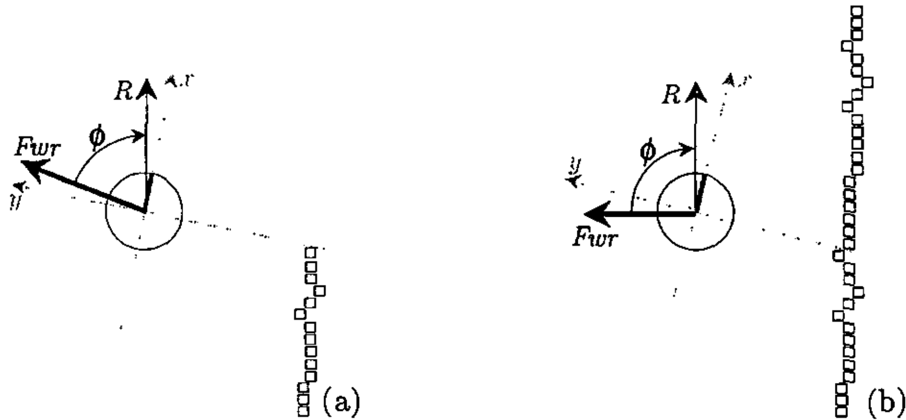


Figura 5.4: Escolha do valor de ϕ de forma que R fique paralelo à parede. A direção de R é dada pelo ângulo constante ϕ e pela direção de Fwr. (a) representa as leituras realizadas pelo robô Pioneer 1. (b) é uma representação de uma leitura ideal que marca todos os pontos da parede.

5.3 Evitando Colisões

Nesta seção serão detalhados os questionamentos "colisão", "risco de colisão" e "obstáculo" e o módulo "evitar colisões". A posição dos obstáculos é definida através do cálculo da força de repulsão entre o robô e os objetos.

5.3.1 Colisão

Em uma primeira instância, foi atribuído para este evento a ação "recuar", para fazer com que o robô saísse das proximidades do obstáculo. Logo, percebeu-se que esta não era uma boa alternativa, pois, em muitos casos o robô vinha a colidir quando estava recuando,

ficando desta forma, preso no estado de colisão. Durante os testes, percebeu-se que as colisões do robô Pioneer1 são devidas principalmente a erros de leitura dos sonares, por este motivo, ao ser detectado uma colisão todas as ordens de movimento são canceladas e novas leituras do sonar são realizadas para atualizar sua percepção do ambiente. Com isto, o estado de colisão não escolhe uma determinada ação a realizar, ficando esta tarefa para os outros módulos do sistema.

Com estas definições, "colisão" será considerado como um estado do ambiente e a única ação para este estado será "cancelar movimentos".

5.3.2 Risco de Colisão

Este não é um módulo necessário para a implementação do aprendizado do robô. Ele foi implementado para evitar que o impacto da colisão danificasse a parte mecânica do robô. Neste módulo, quando o robô detectar que um obstáculo está **muito próximo** a ele, a ação *recuar* é realizada para evitar uma possível colisão. Com isto, definiu-se que "risco de colisão" será um estado do ambiente e a sua ação correspondente será "recuar 30cm".

Assim como o estado de colisão, este não é um bom estado para o robô, portanto será punido todas as vezes que estiver nestes estados. Neste trabalho o conceito de **muito próximo** foi traduzido como $|\vec{F}_r| > 8$.

5.3.3 Obstáculo

No questionamento "obstáculo" deve-se saber onde estão posicionados os objetos e então escolher a melhor ação que "evite colisões". Portanto, este não é como os demais questionamentos que possuem um único estado, ele possui vários estados e várias ações a serem escolhidas.

Para detectar os obstáculos, foram consideradas somente as leituras dos sonares

frontais, pois a força de repulsão dada pelos sonares laterais impede que o robô ande paralelamente às paredes. Quando não for detectado obstáculos, o estado do robô será considerado como "passagem livre" e a ação "avançar" deve ser executada.

Para definir os estados do ambiente, que representam obstáculos, utiliza-se a força $\vec{F}_r$, como pode ser observado na Figura 5.5. Este modelo foi por nós proposto ao perceber que o robô tivera um ótimo desempenho no algoritmo de *planning* e também por diminuir sensivelmente o número de estados do ambiente.

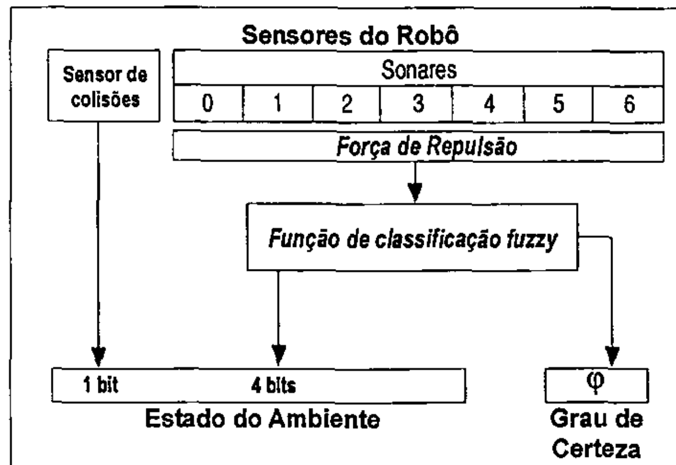


Figura 5.5: Estados do ambiente mapeados através da força de repulsão.

Somente a direção de $\vec{F}_r$ foi utilizada para modelar o conjunto de estados ambiente. Para isto, foram definidas funções *fuzzy* que classificam as direções θ_{F_r} e tamb retornam o grau de pertinência à classe. Por existir somente um sinal a ser classifica cada classe corresponde a um estado do ambiente e o grau de pertinência, μ, correspor ao grau de certeza, φ.

Considerando $0^o \leq \theta_{F_r} \leq 359^o$ dividido em intervalos de 10°, tem-se um total de 36 classes, onde cada classe é representada por um conjunto *fuzzy*. Para que o valor de φ influa no aprendizado é necessário que ele esteja entre 0 e 1, portanto, conclui-se que não deve haver intersecção entre os conjuntos *fuzzy*. Desta forma, os conjuntos *fuzzy* não se intercalam e o único ponto de intersecção entre as classes vizinhas tem, para ambas as classes, $\mu = 0$, como pode ser visto na Figura 5.6.

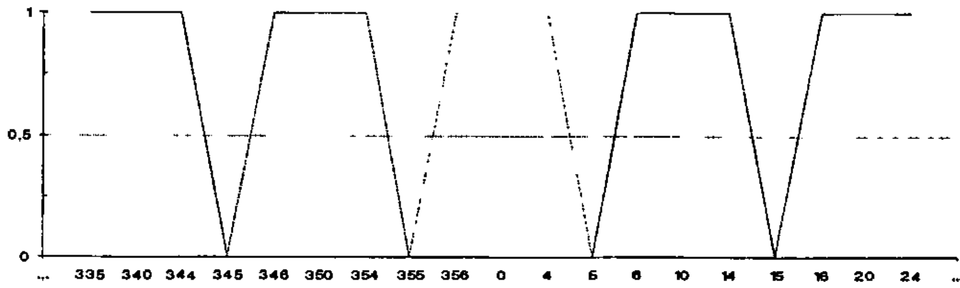


Figura 5.6: Funções de Pertinência: $\mu(\theta_{F_r})$

A classe de θ_{F_r} é definida pela equação (5.7) e o seu grau de pertinência ao conjunto é definido na equação (5.8). Na Figura 5.6, pode ser observado o comportamento da função da função de pertinência em relação à θ_{F_r}.

$$classe(\theta_{F_r}) = int(f(\theta_{F_r})), \tag{5.7}$$

onde $f(\theta_{F_r}) = (\theta_{F_r} - 5)/10$ e $int(f(\theta_{F_r}))$ considera a parte inteira do valor $f(\theta_{F_r})$.

$$\mu(\theta_{F_r}) = \begin{cases} 10\delta & \text{se } \delta < 0.1 \\ -10\delta + 10 & \text{se } \delta > 0.9 \\ 1 & \text{se } 0.1 \leq \delta \leq 0.9 \end{cases}, \tag{5.8}$$

onde $\delta = f(\theta_{F_r}) - int(f(\theta_{F_r}))$.

Com a perspectiva de tornar os movimentos do robô mais suaves, foi escolhido um conjunto de ações, que mantêm somente ângulos entre -90° e 90°. Para montar o conjunto de ações, foi considerado um intervalo de 10° entre elas. O conjunto de ações foi definido como giros sobre seu prôprio eixo, desta forma, $A(s)$ é descrito como:

$$A(s) = \{girar\ 0^o, girar\ 10^o, girar\ -10^o, ..., girar\ 80^o, girar\ -80^o, girar\ 90^o, girar\ -90^o\},$$

onde s é um estado do ambiente que representa obstáculos. A ordem das ações influi bastante na política de controle do robô, por este motivo as ações foram colocadas em ordem de preferência de execução, isto é, que os desvios sejam os mínimos necessários.

5.4 Modelo MDP

De acordo com as especificações realizadas nas seções anteriores, os conjuntos de estados, ações e recompensas, do modelo MDP da aplicação proposta, são apresentados a seguir.

O conjunto de **estados do ambiente** foi definido como:

$$S = \{\textit{colisão, objeto reconhecido, risco de colisão, passagem livre, } \{\textit{obstáculo}\}\},$$

onde *obstáculo* é um conjunto de 36 estados, representado pela equação (5.7), que considera as posições do obstáculo em relação ao robô. Desta forma, tem-se um total de 40 estados do ambiente.

O conjunto de **ações** foi definido de acordo com cada estado do ambiente, como pode ser visto abaixo:

$$\begin{aligned}
&A(\textit{colisão}) && = \{\textit{cancelar movimentos}\} \\
&A(\textit{objeto reconhecido}) && = \{\textit{levar para a lixeira}\} \\
&A(\textit{risco de colisão}) && = \{\textit{recuar 30cm}\} \\
&A(\{\textit{obstáculo}\}) && = \{\textit{girar } 0^o, \textit{ girar } 10^o, \textit{ girar } -10^o, \ldots, \textit{ girar } 90^o, \textit{ girar } -90^o\}
\end{aligned}$$

As **recompensas imediatas** são atribuídas conforme o estado que robô atinge, como é mostrado na tabela abaixo:

estados	recompensas
passagem livre	10
colisão	-100
risco de colisão	-100
levar para lixeira	100
obstáculo	-50

Tabela 5.1: Recompensas para o robô pegador de objetos.

5.5 Resultados Obtidos

Nesta seção é apresentado o resultado obtido pelo robô recolhedor de objetos, utilizando o modelo MDP descrito na seção anterior. Como os estados *colisão, objeto reconhecido* e *risco de colisão* possuem uma única ação, o problema de aprendizado fica centralizado em descobrir qual a melhor ação para cada um dos 36 estados que representam obstáculos. O modelo MDP, utilizado nesta aplicação, é diferente dos modelos do Capítulo 4, não é possível fazer uma comparação de aprendizado entre eles. Contudo, na Figura 5.7, é possível verificar que o robô atingiu uma política ótima a partir do passo 1000.

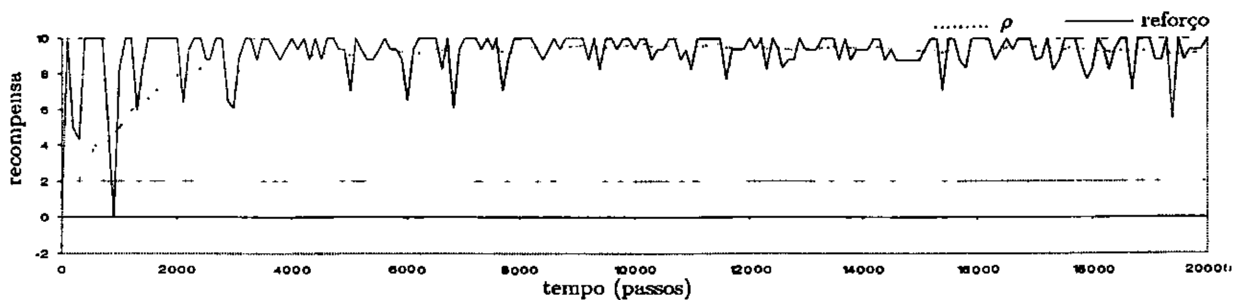


Figura 5.7: R'-*learning* com $\alpha = 0.2$, $\beta = 0.001$ e $\epsilon = 100\%$

Com o conjunto de ações proposto, conseguiu-se atingir os resultados esperados, ou seja, que os movimentos do robô se tornassem mais suaves ao desviar de obstáculos. Além disto, o robô conseguiu executar a tarefa proposta, isto é, pegar objetos pequenos e levá-los para uma lixeira. Para verificar o desempenho do robô nesta tarefa foram colocados 2 corpos de prova em um ambiente como mostrado na Figura 5.8.

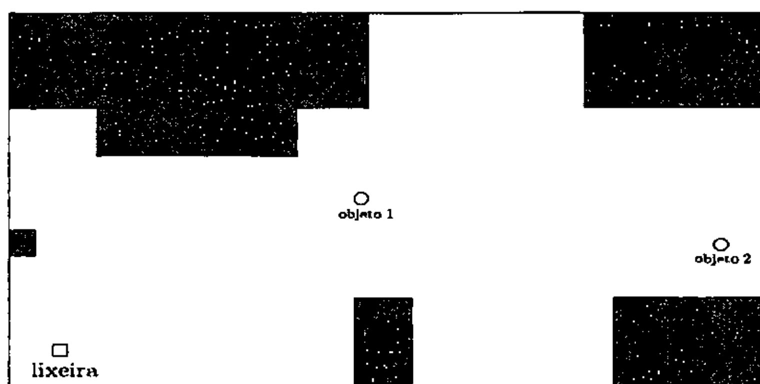


Figura 5.8: Planta do ambiente onde o robô recolheu objetos e aprendeu a navegar.

Capítulo 6

Discussão e Conclusões

Este trabalho teve como meta à investigação de algoritmos de aprendizado com reforço, visando a navegação, em tempo-real, de robôs móveis em ambientes desconhecidos. Existe uma abordagem que se utiliza de um mapa pré-estabelecido do ambiente e através dele aprendem uma política de navegação para o robô. Existe uma outra abordagem que constrói este mapa através da própria exploração que o robô faz no ambiente. Contudo, existe uma terceira abordagem, como a estudada neste trabalho, que não utiliza um mapa do ambiente, fazendo com que o robô possa mudar de ambiente e continuar usando a mesma política de navegação.

Dentro desta última abordagem foram estudados três métodos de aprendizado com reforço no domínio de robôs móveis. Estes métodos foram comparados entre si, considerando a tarefa de navegação em um ambiente real. Logo no princípio, percebeu-se a necessidade de montar um modelo MDP que representasse adequadamente o mundo real. Todos os algoritmos são bastante sensíveis aos conjuntos de estados, ações e recompensas, principalmente o algoritmo H-*learning* que não possui parâmetros, sendo estes conjuntos a única forma de ajustar o aprendizado. Ao fazer ajustes no conjunto de estados do ambiente, concluiu-se que quanto menor fosse o conjunto de estados, mais rápido seria o aprendizado, já que estes algoritmos trabalham por busca exaustiva da melhor ação para cada estado. No entanto, quanto menos estados, menos informações a respeito do

ambiente são passadas para o robô. Durante os experimentos com os três algoritmos, foi utilizado um conjunto de ações que se aproximava de outros trabalhos com robôs móveis. O conjunto de ações foi modificado e percebeu-se que grandes mudanças ocorriam, mesmo quando o conjunto permanecia o mesmo e só a ordem de seus elementos eram alteradas. Isto ocorre devido ao algoritmo de busca da melhor ação por uma política ϵ-gulosa. O conjunto de recompensas imediatas também deve ser bem ajustado, pois após muitos testes, percebeu-se que dependendo do valor da recompensa imediata, ações nas quais se espera uma punição acabam sendo consideradas ações ótimas. Para exemplificar isto, suponha um passo $t \neq 0$, uma ação a_t com recompensa imediata $r = -50$ e as recompensas esperadas $Q(s_t) = 0$ e $Q(s_{t+1}) = 100$. Ao fazer a atualização de $Q(s_t)$ com $\alpha = 0.2$ e $\gamma = 0.75$, tem-se:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$
$$Q(s_t, a_t) = 0 + 0.2[-50 + 0.75(100) - 0] = 5$$

Neste caso, a_t será tomada como ação ótima, pois $Q(s_t)$ é um valor positivo, embora seu valor de recompensa seja $r = -50$. Pode ser que exista uma outra ação que tenha uma recompensa r muito maior, mas ela não será testada, pois o valor $Q(s_t, a_t)$ é positivo. Para que outras ações possam ser também testadas, a recompensa r deve ser negativa o suficiente para que o valor de $Q(s_t)$ possa diminuir. Contudo, se a punição r para uma determinada ação for muito negativa, a política de escolha de ações, ficará alternando entre as ações e não conseguirá encontrar uma ação ótima. Portanto, o conjunto de recompensas deve ser escolhido com bastante cuidado.

Tendo como proposta a tarefa de navegação de robôs móveis, foi mostrado e comparado, neste trabalho, o desempenho dos algoritmos Q-*learning*, R-*learning* e H-*learning*. Para esta tarefa, chegou-se à conclusão que o algoritmo R-*learning* apresenta os melhores resultados, pois consegue aprender a tarefa com um número menor de colisões, além de possuir um conjunto de variáveis de mais fácil manipulação que o Q-*learning*. Com algoritmo Q-*learning* também foram obtidos bons resultados, mas o ajuste da variável γ, até encontrar uma política ótima, é uma tarefa bastante exaustiva. Em outros trabalhos

na literatura, o algoritmo H-*learning* mostrou ter conseguido convergir para a política ótima com bem menos passos que os demais algoritmos. Neste trabalho, porém, não foi conseguida uma política ótima, talvez devido ao próprio modelo MDP utilizado.

Além das implementações dos três algoritmos citados, foi proposta neste trabalho uma modificação no algoritmo R-*learning* com a finalidade de classificar melhor os sinais de entrada. Esta modificação foi realizada através da incorporação de lógica *fuzzy* ao algoritmo R-*learning*, gerando um novo algoritmo denominado R'-*learning*. Este algoritmo apresentou melhores resultados que o R-*learning*, no domínio de navegação de robôs móveis, por calibrar o reforço imediato.

Além disto, uma aplicação para o robô Pioneer 1 foi realizada utilizando o algoritmo R'-*learning*. A aplicação consistiu em fazer com que o robô navegasse em um ambiente desconhecido, evitando colisões e procurando por pequenos objetos para serem colocados em uma lixeira. Para tanto, a abordagem do mapeamento dos estados foi feita utilizando força de repulsão entre possíveis objetos e o robô. Pelo fato do robô possuir poucos sonares, a força de repulsão é calculada através das últimas leituras dos sonares. Este mapeamento propiciou um número menor de estados do ambiente, por fundir as leituras dos sonares em uma única informação, permitindo que o robô aprendesse a navegar em menos tempo. Por outro lado, foi proposto um conjunto de ações que fez com que o robô tivesse movimentos mais suaves.

Assim sendo, a experiência obtida com a realização deste trabalho nos permite concluir que devido à incerteza no ambiente, pois estes algoritmos atuam em ambientes desconhecidos, existe necessidade de muitos ajustes tanto nos conjuntos do modelo MDP quanto nas variáveis internas aos algoritmos.

Como continuidade deste trabalho pretende-se: realizar experimentos com o algoritmo H-*learning* utilizando o modelo MDP definido no Capítulo 5, pois como não houve um bom aprendizado com o MDP mostrado no Capítulo 4, acredita-se que valeria a pena analisar o novo MDP; realizar experimentos de navegação de robôs em outros algoritmos de aprendizado com reforço, como o ARTDP, para que se possa determinar qual algorit-

mo é melhor para realizar uma tarefa específica; implementar algoritmos de aprendizado com reforço via *hardware* e comparar o desempenho com os resultados já obtidos, com a finalidade de construir uma biblioteca de algoritmos de aprendizado para robôs móveis, que possa ser utilizada em aplicações de tempo real.

Referências Bibliográficas

[Bagnell et al., 1998] Bagnell, J., Doty, K., and Arroyo, A. (1998). Comparison of Reinforcement Learning Techniques for Automatic Behavior Programming. In *Proceedings of the CONALD*. CMU-USA.

[Barto et al., 1993] Barto, A. G., Bradtke, S. J., and Singh, S. P. (1993). Learning to act using real-time dynamic programming. *Artificial Intelligence.*

[Bellman, 1957] Bellman, R. (1957). *Applied Dynamic Programming.* Princeton University Press, Princeton, N.J.

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

[Bertsekas, 1995a] Bertsekas, D. P. (1995a). A counterexample to temporal differences learning. *Neural Computation*, 7:270–279.

[Bertsekas, 1995b] Bertsekas, D. P. (1995b). *Dynamic Programming and Optimal Control*, volume 1 and 2. Athena Scientific, Belmont, Massachusetts.

[Borenstein and Koren, 1989] Borenstein, J. and Koren, Y. (1989). Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19:1179–1187.

[Crites and Barto, 1996] Crites, R. H. and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In Touretzky, D., Mozer, M., and Hasselmo, M., editors, *Neural Information Processing Systems 8.*

[Howard, 1960] Howard, R. A. (1960). *Dynamic Programming and Markov Process.* The MIT Press, Cambridge, MA.

[Jaakkola et al., 1994] Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201.

[Kaelbing and Littman, 1996] Kaelbing, L. and Littman, M. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

[Konolige, 1997] Konolige, K. (1997). *Saphira Software Manual.* ActivMedia.

[Littman et al., 1994] Littman, M. L., Dean, T., and Kaelbling, L. P. (1994). Markov games as a framework for multi-agent reinfocement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, San Francisco, CA. Morgan Kaufmann.

[Mahadevan and Connell, 1991] Mahadevan, S. and Connell, J. (1991). Automatic programming of behavior-based robots using reinforcement learning. In *Proceedings of the Ninth National Conference on Altificial Intelligence*, Anaheim, CA.

[Mataric, 1994] Mataric, M. J. (1994). Reward functions for accelerated learning. In Cohen, W. W. and Hirsh, H., editors, *Proceedings of the Eleventh International Conference on Machine Learning.* Morgan Kaufman.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning.* McGraw Hill.

[Puterman, 1994] Puterman, M. L. (1994). *Markov Decison Process–Discrete Stochastic Dynamic Programming.* Inc. John Wiley & Sons, New York, NY.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

[Rosenblatt, 1963] Rosenblatt, F. (1963). *Principles of Neurodynamics.* Spartan, New York.

[Schaal and Atkeson, 1994] Schaal, S. and Atkeson, C. (1994). Robot juggling: An implementation of memory-based learning. *Control Systems Magazine 14*.

[Schwartz, 1993] Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Machine Learning: Proceedings of the Tenth International Conference*, San Mateo, CA. Morgan Kaufmann.

[Shaw and Simões, 1999] Shaw, I. S. and Simões, M. G. (1999). *Controle e Modelagem Fuzzy*. Editora Edgar Blücher LTDA, 1st edition.

[Singh and Dayan, 1994] Singh, S. P. and Dayan, P. (1994). Analytical mean squared error curves for temporal difference learning. *Machine Learning*.

[Sutton, 1988] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

[Sutton, 1996] Sutton, R. S. (1996). Generalization in reinforcement learning: Succesful examples using sparse coarse coding. In Touretzky, D. S., MJozer, C. M., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

[Tadepalli and Ok, 1994] Tadepalli, P. and Ok, D. (1994). A reinforcement learning method for optimizing undiscounted average reward. Technical Report 94-30-01, Department of Computer Science, Oregon State University.

[Tesauro, 1992] Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257–277.

[Thrun, 1995] Thrun, S. (1995). Learning to play the game of chess. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*. The MIT Press.

[Tsitsiklis, 1994] Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202.

[Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD thesis, University of Cambridge.

[Widrow and Hoff, 1960] Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pages 96–104, New York.