

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/228734913>

# Explorando o potencial de algoritmos de aprendizado com reforço em robôs móveis

Article · August 2002

CITATIONS

2

READS

367

2 authors:



**Gedson Faria**

Universidade Federal de Mato Grosso do Sul

13 PUBLICATIONS 53 CITATIONS

[SEE PROFILE](#)



**Roseli A. F. Romero**

University of São Paulo

156 PUBLICATIONS 903 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



R-CASTLE: Robotic - Cognitive Adaptive System for Teaching and Learning [View project](#)

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

Gedson Faria, Roseli Francelin Romero  
USP - ICMC - SCE  
São Carlos - SP - Brasil  
E-mails: gedson, rafrance@icmc.sc.usp.br

### Abstract

*There are many methods to solving the reinforcement learning problem. In this work, reinforcement learning algorithms are investigated in the context of the small robot navigation. Some methods as Q-learning, R-learning and H-learning are presented. A MDP environment (Markov Decision Process) is proposed by utilizing the fuzzy logic concepts and experiments performed with the Q-learning algorithm are also presented and discussed.*

### 1. Introdução

Aprendizado com reforço é baseado na idéia de 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-se 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.

Este tipo de aprendizado difere do aprendizado supervisionado implementado pelo Perceptron [1] e Adaline [2]. 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.

Uma diferença importante entre o método da Correção do Erro e o Aprendizado com Reforço, é pelo último não se basear exclusivamente nos seus pesos para determinar suas ações, ele gera ações por um processo aleatório que é polarizado por uma combinação dos valores de seus pesos e de suas entradas. Assim, as ações não são vistas apenas como respostas aos padrões de entrada, mas também dependem dos estados do sistema. Além disso, se a resposta desejada não é conhecida, a avaliação do desempenho do sistema é obtida indiretamente considerando o “efeito” de sua saída no ambiente com o qual o sistema interage. Aprendizado com reforço é aplicado quando este “efeito” é medido através de mudanças em um determinado sinal: *reinforcement* (termo usado na teoria de aprendizagem de animais) [3].

O objetivo deste trabalho é investigar o potencial dos algoritmos de aprendizado com reforço existentes, vi-

sando a navegação de robôs móveis em um ambiente qualquer.

Inicialmente, é apresentado um estudo sobre uma estrutura moderna e bastante utilizada no aprendizado com reforço, chamada de Processo de Decisão de Markov ou MDP (*Markov Decision Process*), discutido na seção 2. Em seguida, alguns dos algoritmos utilizados em aprendizado com reforço, que utilizam métodos *model-free* como Q-learning [4, 5], R-learning [6] e o método *model-based* no H-learning [7], são apresentados na seção 3. Os detalhes de implementação propostos de um MDP finito, para o algoritmo Q-learning é apresentado na seção 4. Para finalizar, são apresentadas as sugestões para trabalhos futuros na seção 5.

### 2. Processo de Decisão de Markov

Uma forma de modelar problemas de aprendizado com reforço é utilizar o Processo de Decisão de Markov.

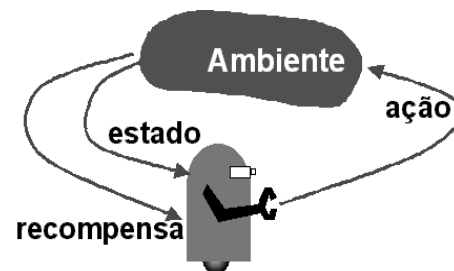


Figura 1: Modelo padrão de aprendizado com reforço.

Três sinais passam de um lado para outro entre o agente e o ambiente, como é mostrado na Figura 1, caracterizando o problema de aprendizagem: um sinal para representar as escolhas feitas pelo agente, um sinal que indica o estado do ambiente e um sinal para definir as metas do agente, representando respectivamente as ações, estados do ambiente e as recompensas.

O sinal de recompensa é a maneira de se comunicar ao agente como atingir a meta, não devendo, de forma alguma, lhe indicar como realizar esta tarefa. A fonte da recompensa está representada fora do agente, mas isto não impede que o agente defina para si um tipo de recompensa interna ou uma sequência de recompensas internas.

Na estrutura da aprendizagem com reforço, o agente faz suas decisões com base num sinal do ambiente chamado de estado do ambiente. Caso um estado contenha toda a informação relevante, então ele é chamado de Markov ou que tem a propriedade de Markov. Pode-se observar esta propriedade na velocidade e posição atuais de uma bola de canhão, observando que estas informações são suficientes para determinar seu voo 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 Processo de Decisão de Markov finito ou MDP finito.

A fim de ilustrar um MDP, é apresentado a seguir um exemplo simples, porém não realista, de um robô que tem por objetivo coletar o maior número de latas possíveis, gastando o mínimo de energia.

Supõe-se que sejam consideradas as três seguintes decisões: procurar ativamente por uma lata, permanecer parado esperando que alguém lhe traga a lata e voltar a base para recarregar a bateria. O melhor modo de se encontrar latas é procurando-as ativamente, mas isto descarrega a bateria do robô. Por outro lado, somente esperar não é uma boa maneira de se conseguir as latas. Sempre que o robô está procurando é possível que sua bateria se esgote; neste caso o robô deve desligar e esperar seu resgate o que provoca uma recompensa baixa.

O agente faz suas decisões baseado no nível de energia da bateria, distinguidos por dois níveis (**alto** e **baixo**). O agente tem a possibilidade de escolher entre **esperar**, **procurar** ou **recarregar** se o nível da bateria estiver baixo. Com isto, pode-se definir o conjunto de estados  $S$  e o conjunto de ações  $A(s)$ , como:

$$S = \{\text{alto}, \text{baixo}\}$$

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

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

A cada lata coletada é adicionado +1 na recompensa e caso ele fique sem energia uma punição de -3 é administrada.  $R^{\text{procurar}}$  e  $R^{\text{esperar}}$  representam o número de latas coletadas enquanto “procurava” e “esperava” respectivamente, tal que  $R^{\text{procurar}} > R^{\text{esperar}}$ . 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, pode-se escrever as probabilidades de transição e as recompensas esperadas como na Tabela 1 ou como um diagrama de transição de estados visto na Figura 2.

Estando com a bateria no nível alto e executando a ação procurar tem-se duas possibilidades: a bateria continuar alta,  $P = \alpha$ , ou baixar,  $P = 1 - \alpha$ . Caso esteja com o nível baixo e execute a ação procurar tem-se duas possibilidades: continuar no nível baixo com  $P = \beta$  ou

Tabela 1: Transição de estados.

$s = s_t$	$s' = s_{t+1}$	$a = a_t$	$P_{ss'}^a$	$R_{ss'}^a$
alto	alto	procurar	$\alpha$	$R^{\text{procurar}}$
alto	baixo	procurar	$1 - \alpha$	$R^{\text{procurar}}$
baixo	alto	procurar	$1 - \beta$	-3
baixo	baixo	procurar	$\beta$	$R^{\text{procurar}}$
alto	alto	esperar	1	$R^{\text{esperar}}$
alto	baixo	esperar	0	$R^{\text{esperar}}$
baixo	alto	esperar	0	$R^{\text{esperar}}$
baixo	baixo	esperar	1	$R^{\text{esperar}}$
baixo	alto	recarregar	1	0
baixo	baixo	recarregar	0	0

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 têm probabilidade 0(zero) de ocorrer. No caso da escolha da ação recarregar o próximo estado será de bateria alta, não havendo outra possibilidade.

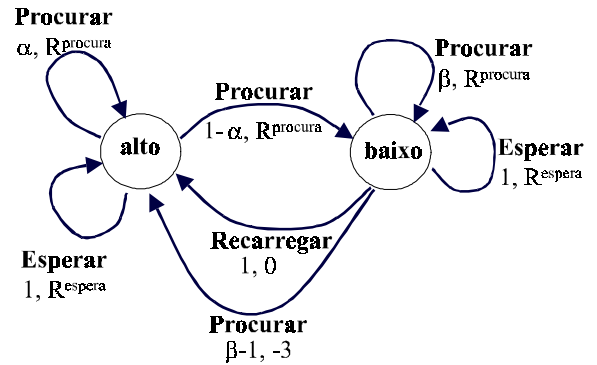


Figura 2: Diagrama de transição de estados.

A definição de um MDP finito é bem ilustrada no exemplo do robô reciclador, podendo ser generalizada através dos seguintes elementos:

$S$  conjunto de estados do ambiente

$A(s)$  conjunto de ações possíveis no estado  $s$

$P_{ss'}^a$  probabilidade de transição de  $s$  para  $s'$  dado  $a$

$R_{ss'}^a$  recompensa pela transição de  $s$  para  $s'$  dado  $a$

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

Existem boas referências para MDPs, que podem ser encontradas em [8, 9, 10, 11].

Na maioria das vezes, o modelo MDP não está completo, ou seja, não se conhece a função de probabilidade de transição de estado ou a função de recompensas esperadas. Tem-se para isso métodos chamados *model-free* que aprendem um controlador sem aprender um modelo.

### 3. Métodos de Aprendizado com Reforço

O aprendizado com reforço dispõe de vários métodos de aprendizagem. Foram escolhidos alguns métodos, com características distintas, para que fossem estudados e suas eficiências avaliadas e comparadas entre si. A seguir são apresentados os algoritmos *Q-learning* e *Re-learning* que utilizam o método *model-free* e o algoritmo *model-based*, *H-learning*.

#### 3.1. Q-learning

O algoritmo *Q-learning* [4, 5] consiste na atualização de valores descontados de recompensas esperadas,  $Q(s, a)$ . A cada iteração com o ambiente, os valores de  $Q$  são atualizados de acordo com a equação (1).

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma eQ(s')] \quad (1)$$

$\gamma$  é o fator de desconto utilizado para garantir que os valores de  $Q$  sejam finitos e  $\alpha$  é a constante de aprendizado, sendo que:  $0 < \alpha \leq 1$  e  $0 \leq \gamma < 1$ .

Após executar a ação  $a$ , o agente sai do estado  $s$  e vai para um estado  $s'$ , recebendo por esta ação uma recompensa imediata  $r$ . No estado  $s'$  é feita uma busca, entre as ações disponíveis, para encontrar a ação  $a'$  que tenha o maior valor de retorno esperado, representado por  $eQ(s') = \max_{a'} Q(s', a')$ .

Caso a ação  $a'$  seja tomada como sendo a próxima a ser executada, tem-se uma probabilidade maior de cair em máximos locais. No algoritmo *Q-learning*, para cada passo do episódio, deve-se escolher uma ação que não é necessariamente igual a  $a'$ . Uma boa escolha, por exemplo, é escolher, em 70% dos casos, a ação que retorne o valor máximo e nos outros 30% faz-se escolhas aleatórias para evitar os máximos locais.

Na Figura 3 apresenta-se uma descrição do algoritmo *Q-learning*, na qual nota-se uma reestruturação da equação (1). Tal reestruturação eliminou uma multiplicação e adicionou uma subtração, melhorando o custo computacional.

```

Inicialize  $Q(s, a)$  arbitrariamente
Repita (para cada episódio)
  Inicialize  $s$ 
  Repita para cada passo do episódio
    Escolha  $a \in A(s)$ 
    Execute a ação  $a$ 
    Observe os valores  $s'$  e  $r$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma eQ(s') - Q(s, a)]$ 
     $s \leftarrow s'$ 
  até que  $s$  seja terminal
  
```

Figura 3: Algoritmo *Q-learning*

Para melhor entendimento, será especificado um modelo MDP para um ambiente bem simples, no exemplo seguinte.

O ambiente é uma matriz 5x5 no qual o aprendiz sai de um ponto qualquer e tem que chegar no estado meta. Para movimentar-se nesse ambiente o aprendiz dispõe de quatro alternativas: subir, descer, direita e esquerda. O objetivo é bem simples: andar numa matriz e alcançar o mais rápido possível o estado meta. Para atingir o objetivo, utiliza-se as recompensas para definir como aprendiz deve agir. Portanto, se o aprendiz atingir o estado meta a recompensa será  $r = 10$  e  $r = -1$  nos outros casos.

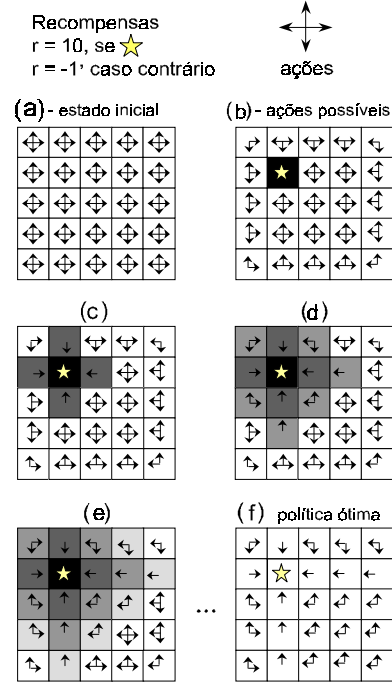


Figura 4: Esquema de aprendizado numa matriz 5x5. A ordem das matrizes não indica suas atualizações em tempos subsequentes. Elas apenas ilustram uma sequência hipotética do processo de aprendizado.

Na Figura 4 (a), observa-se que inicialmente todas as ações são possíveis. As ações que levam para fora do limite da matriz não podem ser executadas e por isso foram retiradas, e são apresentadas na Figura 4 (b). Partindo-se deste ponto, para qualquer ação realizada em qualquer estado a recompensa será de -1, exceto para as ações que levam ao estado meta, por isso o par estado-ação, isto é,  $Q(s, a)$ , que levou ao estado meta será privilegiado nas próximas escolhas de ações, Figura 4 (c). A atualização dos valores de  $Q$  não dependem somente da recompensa  $r$  mas também do estado seguinte, como mostra a equação (1). Sendo assim, os estados vizinhos que com alguma ação atinjam os estados anteriormente “privilegiados” serão os próximos a levarem uma parte deste privilégio, como pode ser visto na Figura 4 (d). A propagação dos privilégios segue continuamente, como visto na Figura 4 (e), até chegar ou pelo menos se aproximar de uma política ótima, como mostrado na Figura 4 (f).

### 3.2. R-learning

A técnica proposta por Schwartz [6], chamada de *R-learning*, maximiza a recompensa média a cada passo, ou seja, utiliza *average-reward model*. *Q-learning* não maximiza a recompensa média, mas descontos acumulado de recompensa, por isso *R-learning* deve fornecer de fato resultados melhores que o *Q-learning*.

O algoritmo *R-learning* possui regra similar ao *Q-learning*, sendo baseado na dedução de valores  $R(s, a)$ , e devendo escolher ações  $a$  num estado  $s$ . A cada situação, o aprendiz escolhe a ação que tem o maior valor  $R$ , exceto que algumas vezes ele escolhe uma ação qualquer. Os valores de  $R$  são ajustados a cada ação, baseado na seguinte regra de aprendizagem:

$$R(s, a) \leftarrow (1 - \alpha)R(s, a) + \alpha[r - \rho + eR(s')], \quad (2)$$

que difere da regra do *Q-learning*, simplesmente por subtrair a recompensa média  $\rho$  do reforço imediato  $r$  e por não ter desconto  $\gamma$  para o próximo estado,  $eR(s') = \max_a R(s', a)$ . A recompensa média é calculada como:

$$\rho \leftarrow (1 - \beta)\rho + \beta[r - \rho + eR(s') - eR(s)] \quad (3)$$

O ponto chave é que  $\rho$  somente é atualizado quando uma ação não aleatória foi tomada, ou seja,  $\max_a R(s, a) = R(s, a)$ . A recompensa média  $\rho$  não depende de algum estado particular, ela é uma constante para todo o conjunto de estados.

Na Figura 5, é apresentado o algoritmo *R-learning*, na qual pode-se observar pequenas reestruturações nas equações de atualização de  $R$  e  $\rho$ , que melhoram o custo computacional.

```

Inicialize  $\rho$  e  $R(s, a)$  arbitrariamente
Repita para sempre
   $s \leftarrow$  estado atual
  Escolha  $a \in A(s)$ 
  Execute a ação  $a$ 
  Observe os valores  $s'$  e  $r$ 
   $R(s, a) \leftarrow R(s, a) + \alpha[r - \rho + eR(s') - R(s, a)]$ 
  se  $R(s, a) = \max_a R(s, a)$  então
     $\rho \leftarrow \rho + \beta[r - \rho + eR(s') - eR(s)]$ 

```

Figura 5: Algoritmo *R-learning*

### 3.3. H-learning

O algoritmo *H-learning* [7] foi introduzido para otimizar a recompensa média sem utilizar desconto. Este método é totalmente diferente dos demais vistos até agora, pois utiliza o método *model-base*, tendo de construir o modelo e deste derivar um controlador.

O algoritmo *H-learning* estima as probabilidades  $p_{ik}(a)$  e os reforços  $r(i, a)$  por contagem direta e atualiza os valores de  $h$  utilizando uma equação que segundo

1. Seja  $N(i, u)$  ser o número de vezes que a ação  $u$  foi executada no estado  $i$ , e seja  $N(i, u, j)$  ser o número de vezes que ela resultou no estado  $j$ . Inicialize as matrizes  $p_{ij}(u)$ ,  $r(i, u)$ ,  $h(i)$  e o escalar  $\rho$  com 0's.  $p_{ij}(u)$  é a probabilidade de ir de um estado  $i$  para um estado  $j$  executando a ação  $u$ ,  $r(i, a)$  é a recompensa estimada por executar a ação  $a$  no estado  $i$ ,  $h(i)$  é a recompensa máxima esperada para o estado  $i$  e corresponde ao  $eQ(s')$  no algoritmo *Q-learning*. A constante  $\rho$  representa a média das recompensas, assim como no *R-learning*.  $U_{best}(i)$  é o conjunto de ações ótimas no estado  $i$  e é inicializado com  $U(i)$ .  $T$  é o número total de passo que uma ação aparentemente ótima foi executada e é inicializada com 0. Atribua a  $i$  uma valor aleatório do estado corrente.
2. Repetir
  - (a) Se a estratégia de exploração sugere uma ação aleatória, pegue uma ação aleatória para  $i$ , senão execute a ação  $a \in U_{best}(i)$ . Deixe  $k$  ser o estado resultante, e  $r'$  a recompensa imediata recebida.
  - (b)  $N(i, a) \leftarrow N(i, a) + 1$
  - (c)  $N(i, a, k) \leftarrow N(i, a, k) + 1$
  - (d)  $p_{ik}(a) \leftarrow N(i, a, k)/N(i, a)$
  - (e)  $r(i, a) \leftarrow r(i, a) + (r' - r(i, a))/N(i, a)$
  - (f) Se a ação executada  $a \in U_{best}(i)$ , então
    - $T \leftarrow T + 1$
    - $\rho \leftarrow \rho + (r' - h(i) + h(k) - \rho)/T$
  - (g) Deixe  $H(i, u) = r(i, u) + \sum_{j=1}^n p_{ij}(u)h(j)$ 
    - $U_{best}(i) \leftarrow \{v | H(i, v) = \max_{u \in U(i)} H(i, u)\}$
    - $h(i) \leftarrow H(i, a) - \rho$ , onde  $a \in U_{best}(i)$
  - (h)  $i \leftarrow k$

Até convergir ou MAX-STEPS vezes

Figura 6: Algoritmo *H-learning*

teorema provado por Bertsekas [11] converge para uma política ótima. O algoritmo *H-learning*, pode ser observado na Figura 6.

## 4. Implementação

O algoritmo *Q-learning* foi implementado e testado com o robô Pioneer 1, mostrado na Figura 7, utilizando uma estação PC Windows 95 com Visual C/C++ da Microsoft e o software Saphira.

A decisão inicial de trabalhar com o *Q-learning*, é pelo fato deste ser um algoritmo mais simples que os outros apresentados na seção 3, e ele certamente fornecerá um bom modelo MDP para os outros algoritmos.

Inicialmente, foi proposta uma única tarefa: andar num ambiente qualquer evitando colisões. Esta tarefa pode parecer simples, mas é a base do problema. Após o robô encontrar uma forma de aprendizado boa o suficiente para esta tarefa, tarefas mais complexas poderão ser analisadas, como por exemplo: andar num ambiente evitando colisões e juntando o maior número de latas possível; ou ainda servir de guia dentro de um prédio, de uma escola, de um museu ou qualquer outro local. Tais tarefas dependem das limitações do robô e do algoritmo de aprendizagem.

#### 4.1. O Robô Pioneer Gripper

O robô está montado sobre um eixo de duas rodas que permite fazer rotações e movimentos para frente ou para trás. Como pode-se observar na Figura 7, este robô tem pequenas dimensões. A garra pode pegar objetos de até 21,5cm de comprimento e possui um sistema de elevação vertical que permite pegar objetos com no mínimo 2cm de altura.

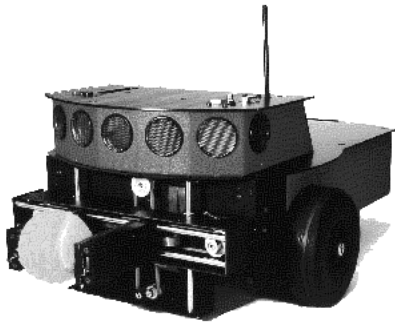


Figura 7: Pioneer 1 Gripper

Também possui cinco sonares frontais, um na lateral direita e um na lateral esquerda. Os sensores de colisão estão localizados nas rodas e nas extremidades da garra. Os sonares emitem raios em projeção cônica. Dependendo do formato do objeto e do ângulo de reflexão, tais raios podem não ser captados, mesmo que o objeto esteja próximo. Os sonares devolvem como valor de retorno a distância, que pode variar de 200mm até 5000mm. Isto causa um problema, pois pode existir um objeto encostado no robô mas o sensor vai indicar que ele está a 200mm de distância.

O software utilizado para manipulação dos movimentos do robô é o Safira da ActivMedia, 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 FreeBSD, Linux ou UNIX é necessário a biblioteca Motif GUI. Para versões Windows95/NT, deve-se usar somente o Microsoft Visual C/C++.

#### 4.2. Definindo o Ambiente MDP

Inicialmente tentou-se montar um conjunto de estados como sendo um mapa do ambiente real. Tal definição foi descartada pois o aprendizado ocorreria apenas para um ambiente específico.

Bagnell [12] propôs para o robô “Charm” um vetor de estados com 128 entradas, sendo 3 bits para cada um de seus dois sensores de infravermelho, localizados na parte dianteira, e 1 bit para indicar colisão. Para cada estado existem seis entradas para cada uma das seguintes ações: “*forward, backwark, spin left, spin right, turn left, turn right*”. Definiu também as recompensas para cada ação.

Seguindo-se esta proposta, definiu-se um conjunto de estados como sendo o conjunto de valores retornados pelo robô, ou seja, as distâncias dos objetos e o sinal de colisão.

Neste trabalho propõe-se a utilização de lógica fuzzy [13] para indicar “quão perto” um objeto está do robô, pois nenhuma informação sobre como representar tais sinais foi encontrada. Propõe-se ainda uma entrada de quatro sinais de estado do ambiente, indicando as distâncias dos objetos à direita, à esquerda e à frente do robô, e um sinal de colisão. Os sinais dos cinco sonares frontais foram convertidos em um único sinal de distância à frente, calculado como sendo o valor mínimo entre eles.

Os seguintes passos foram utilizados para decidir com quantos bits deve-se representar o sinal de entrada do sonar e como classificá-los utilizando lógica fuzzy:

- O sinal recebido pelo sonar foi subdividido em quatro partes iguais, representado por 2 bits. Com 2 bits por sonar e 1 bit para colisão, tem-se um total de 7 bits, ou seja  $2^7 = 128$  estados. As decisões tomadas para evitar colisões, ocorrem quando os objetos estão próximos do robô. Para a subdivisão proposta acima, tal estado é muito amplo e força o robô a desviar de objetos em mais de 1,5m de distância.
- Visto que era necessário mais estados para quando o robô estivesse próximo de algum objeto, o sinal do sonar foi dividido em dezesseis partes iguais. Esta representação de 4 bits por sonar e 1 para colisão totaliza  $2^{13} = 8192$  estados. Um número elevado de estados, não mostrou ser de grande valia, pois o robô teve de treinar vários estados em que os objetos estavam distantes, obtendo em todos um mesmo aprendizado.
- Ao retornar à configuração de 2 bits por sonar, fez-se uma melhor divisão entre os valores recebidos do sonar, classificando-os de acordo com a Tabela 2.

Tabela 2: Classificação de distâncias em 4 partes.

Intervalo (mm)	Especificação
200 → 375	MUITO PERTO
376 → 500	PERTO
501 → 1000	MEIO
1001 → 5000	LONGE

O processo de aprendizagem progrediu, pois o robô começou a evitar as colisões, mas também começou a evitar os movimentos para frente. Geralmente, as colisões ocorrem quando pelo menos um sensor é classificado como MUITO PERTO e para cada colisão uma recompensa bastante negativa é dada. Por este motivo, o robô começou a evitar as ações que o levaram a colidir, ou seja, os movimentos para frente. As recompensas atrasadas se tornaram visíveis ao observar o robô evitando movimentos para frente cada vez mais distantes dos objetos, chegando

ao ponto de ficar apenas girando sobre seu próprio eixo.

- Nota-se um sucesso parcial para quatro estados, mas percebe-se que ainda são insuficientes. Ao aumentar para 3 bits por sonar, ou seja,  $2^{10} = 1024$  estados, deixou-se um intervalo de distância menor para objetos próximos, como é mostrado na Tabela 3. Bons resultados foram obtidos com essa classificação.

Tabela 3: Classificação das distâncias em 8 partes.

Intervalo (mm)	Especificação
200 → 300	ENCOSTADO
301 → 400	PRÓXIMO
401 → 500	MUITO PERTO
501 → 600	PERTO
601 → 750	MEIO
751 → 900	LONGE
901 → 1500	MUITO LONGE
1501 → 5000	DISTANTE

O fato da maioria das colisões ocorrerem quando se escolhe uma ação de deslocamento para frente, não trouxe problemas só para o conjunto de estados, mas também para o conjunto de ações. Primeiramente definiu-se um conjunto de ações como:  $A(s) = \{\text{avançar, recuar, avançar girando } 90^\circ, \text{ avançar girando } -90^\circ\}$ , para dar a impressão de um movimento contínuo. Tal representação teve  $\frac{3}{4}$  das ações sendo severamente punidas, o que não trouxe bons resultados. Um melhor conjunto de ações foi definido como:  $A(s) = \{\text{avançar, recuar, girar } 45^\circ, \text{ girar } 90^\circ, \text{ girar } -45^\circ, \text{ girar } -90^\circ\}$ .

Para não deixar que o robô pare de andar para frente, incentiva-se esta ação através de recompensa bastante positiva, como é mostrado na Tabela 4.

Na implementação do *Q-learning*, definiu-se os seguintes valores para as constantes  $\alpha$  e  $\gamma$ , sendo o fator de desconto  $\gamma = 0.99$  e a taxa de aprendizado  $\alpha = 0.25$ .

Tabela 4: Reforço para cada ação.

Ação ( $a$ )	Recompensa ( $r$ )
Colisão	-700
Avançar	90
Recuar	-50
Giros	50

Na Figura 8 pode-se observar a convergência do aprendizado para dois casos distintos de escolhas de ações para o algoritmo *Q-learning*. Um utiliza somente o método guloso, escolhendo sempre a ação que lhe rende maior recompensa e o outro faz 30% das escolhas aleatórias. Pode-se perceber que a linha que representa o método 100%-guloso convergiu mais rápido, mas não conseguiu maximizar a recompensa, ou seja, caiu em máximo local.

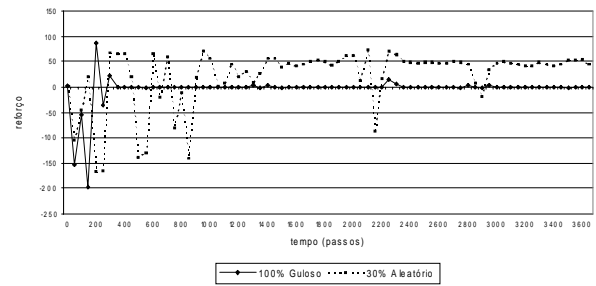


Figura 8: Comparação de aprendizados

## 5. Trabalhos Futuros

Todos os testes realizados até então, foram feitos utilizando-se o algoritmo *Q-learning*. No entanto, este algoritmo é um dos mais simples existentes para aprendizado em tempo real. Assim sendo, outros algoritmos de aprendizado com reforço serão implementados e comparados entre si, com a finalidade de definir uma abordagem que possibilitará a navegação do robô Pioneer em um ambiente qualquer.

## Referências

- [1] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1963.
- [2] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pages 96–104, New York, 1960.
- [3] C. W. Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Mag.*, pages 31–37, Apr. 1989.
- [4] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.
- [5] C. J. C. H. Watkins and P. Dayan. Technical note: Q learning. *Machine Learning*, 8:279–292, 1992.
- [6] A. Schwartz. A reinforcement Learning Method for Maximizing Undiscounted Rewards. In *Machine Learning: Proceedings of the Tenth International Conference*, San Mateo, CA, 1993. Morgan Kaufmann.
- [7] P. Tadepalli and D. Ok. A reinforcement learning method for optimizing undiscounted average reward. Technical Report 94-30-01, Department of Computer Science, Oregon State University, 1994.
- [8] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N.J., 1957.
- [9] M. L. Puterman. *Markov Decision Process—Discrete Stochastic Dynamic Programming*. Inc. John Wiley & Sons, New York, NY, 1994.
- [10] R. A. Howard. *Dynamic Programming and Markov Process*. The MIT Press, Cambridge, MA, 1960.
- [11] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Pentice-Hall, Englewood Cliffs, NJ, 1987.
- [12] J. Bagnell, K. Doty, and A. Arroyo. Comparison of Reinforcement Learning Techniques for Automatic Behavior Programming. In *Proceedings of the CONALD*. CMU-USA, 1998.
- [13] S. V. Kartalopoulos. *Understanding Neural Networks and Fuzzy Logic*. IEEE Press, 1996.