CENTRO UNIVERSITÁRIO DA FEI

MURILO FERNANDES MARTINS

APRENDIZADO POR REFORÇO ACELERADO POR HEURÍSTICAS APLICADO AO DOMÍNIO DO FUTEBOL DE ROBÔS

São Bernardo do Campo
2007

MURILO FERNANDES MARTINS

APRENDIZADO POR REFORÇO ACELERADO POR HEURÍSTICAS APLICADO AO DOMÍNIO DO FUTEBOL DE ROBÔS

Dissertação de Mestrado apresentada ao Centro Universitário da FEI para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Reinaldo A. da C. Bianchi

São Bernardo do Campo
2007

Ficha Catalográfica

Martins, Murilo Fernandes

Aprendizado por reforço acelerado por heurísticas aplicado ao domínio do futebol de robôs / Murilo Fernandes Martins. São Bernardo do Campo, 2007.

102 f. : il.

Dissertação de Mestrado - Centro Universitário da FEI.

Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi

1. Inteligência artificial. 2. Aprendizado por reforço. 3. Aprendizado acelerado por heurísticas. 4. Futebol de robôs. 5. Robótica móvel. I. Título.

CDU 007.5

Folha de Aprovação

(anexar na versão final)

Ao meu avô Antônio Augusto Fernandes, por ter-me ensinado a sonhar e ter-me mostrado que sonhos se realizam. Por todos os momentos da minha infância e pela sua eterna companhia.

AGRADECIMENTOS

Não poderia iniciar de outra forma, senão agradecendo aos meus pais, Maria Lucia Fernandes Martins e Leônidas César Martins Filho, pelas renúncias a mim dedicadas, por terem feito muito mais que o possível para que eu tivesse condições de aqui chegar. Agradeço à minha avó Angelina Medeiros Fernandes pelo apoio incondicional e por me fazer acreditar em meus sonhos. Agradeço aos meus tios Antônio Medeiros Fernandes e Francisco Medeiros Fernandes pelo incentivo e confiança, pela importância de ter recebido o apoio deles. Ao meu avô Antônio Augusto Fernandes, em quem me inspiro e busco forças, o maior responsável pelos caminhos da minha vida e o alicerce da brilhante família que tenho.

Aos professores Dr. Carlos E. Thomaz, Dr. Flavio Tonidandel, Dr. Paulo E. Santos e meu orientador Dr. Reinaldo A. C. Bianchi, pela amizade, pelos conselhos, por terem compartilhado conhecimento comigo nas aulas e nas tantas conversas descontraídas de almoço e dos cafés da tarde. Agradeço a eles, responsáveis pela minha formação, por acreditarem em meu trabalho e pelas oportunidades que têm me dado.

Agradeço aos amigos, membros da equipe de Futebol de Robôs do Centro Universitário da FEI, André de Oliveira Santos, Diego Oliveira Fernandes, Fernando Perez Tavarez, José Ângelo Gurzoni Junior, Ronaldo K. Satomi e Valquiria Fenelon Pereira, que contribuíram para que esse trabalho fosse realizado e cuja ajuda foi de um valor inestimável.

Pelos momentos ímpares da fase de aquisição de créditos, agradeço à primeira turma de mestrado do grupo IAAA do Centro Universitário da FEI: Edson Caoru Kitani, Julio A. Sgarbi, Leandro Demari Rodrigues, Luiz Antônio Celiberto Jr., Marcel Lira Gomes, Nelson A. O. de Aguiar, Rodolfo Coura de Brito e Sérgio Henry A. de Oliveira.

Agradeço ao Centro Universitário da FEI por fornecer toda a infra-estrutura necessária para o desenvolvimento desse trabalho, como equipamentos, computadores e laboratórios.

Agradeço a CAPES pela bolsa de estudos de mestrado a mim concedida, a qual foi determinante para o bom desenvolvimento da minha pesquisa durante esses dois anos.

Agradeço a Deus pelo presente da vida.

Mestre não é só quem ensina; mas quem, de repente, aprende.

Guimarães Rosa

RESUMO

Esse trabalho apresenta uma comparação entre algoritmos de Aprendizado por Reforço com e sem a utilização de heurísticas para aceleração do aprendizado em ambiente simulado e a transferência de conhecimento, através de heurísticas, para o ambiente real. O ambiente de Futebol de Robôs é utilizado como plataforma para os experimentos realizados, pois é um ambiente complexo, dinâmico e não-determinístico. As informações do ambiente foram abstraídas e o conjunto de estados foi definido por regiões, enquanto o conjunto de ações representa diferentes comportamentos de alto nível. Foram efetuados experimentos em ambiente real e simulado. Os testes em ambiente simulado mostraram que heurísticas aceleram o aprendizado significativamente. Para os testes em ambiente real, foi desenvolvido um sistema completo de um time de Futebol de Robôs e o conhecimento adquirido no aprendizado em simulação foi transferido através das heurísticas. Os resultados mostraram que algoritmos de Aprendizado por Reforço acelerados por heurísticas implicam em um melhor desempenho quando comparados com os algoritmos tradicionais de Aprendizado por Reforço.

ABSTRACT

This work presents a comparison between Reinforcement Learning algorithms with and without the use of heuristics to accelerate the learning task in a simulated environment and the knowledge transfer, through heuristics, for the real environment. The Robot-soccer environment is used as a test platform, because it is a complex, dynamic and non-deterministic environment. The environments' informations were abstracted and the state space was defined by regions, while the action space represents different high level behaviors. Experiments were done in real and simulated environments. The tests in simulated environment showed that heuristics accelerate the learning significantly. For the real environment tests, a complete system of a Robot-soccer team was developed and the learning acquired in the simulated environment was transferred through heuristics. The results showed that the heuristically accelerated Reinforcement Learning algorithms imply in a better performance when compared with traditional Reinforcement Learning algorithms.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

AR Aprendizado por Reforço

CRC Cyclic Redundancy Check – Código de Redundância Cíclica

HAQL *Heuristically Accelerated Q-learning* – Q-learning Acelerado por Heurísticas

HAQS *Heuristically Accelerated Q-spreading* – Q-spreading Acelerado por Heurísticas

HAQ(λ) *Heuristically Accelerated Q(λ)* – Q(λ) Acelerado por Heurísticas

IA Inteligência Artificial

PMD Processo Markoviano de Decisão

PWM *Pulse Width Modulation* – Modulação por Largura de Pulso

QS QS-learning

Q(λ) Q(λ)-learning

TD Temporal Diferences – Diferenças Temporais

TH Transformada de Hough

LISTA DE SÍMBOLOS

α Taxa de aprendizado

β Variável de influência da heurística

γ Fator de desconto para os reforços futuros

δ Erro de diferença temporal

ε Taxa de exploração do ambiente

η Variável de influência da heurística

λ Fator de desconto de diferenças temporais futuras

σ Função de espalhamento

ξ Variável de influência da heurística

τ Variável de ponderação da similaridade

S Conjunto de estados

A Conjunto de ações

R Função de recompensa

T Função de transição de estados

H Função heurística

π Uma política

π^* A política ótima

π^H Política heurística

Q Função ação-valor

$\hat{Q}$ Estimativa da função ação-valor

Q^* Função ação-valor ótima

V Função valor

$\hat{V}$ Estimativa da função valor

V^* Função valor ótima

1 INTRODUÇÃO

A idéia de robôs que jogam futebol foi mencionada pela primeira vez pelo professor Alan Mackworth (1992) em um artigo intitulado *"On Seeing Robots"*, apresentado no *Vision Interface'92*, em Vancouver, Canadá.

Nesse mesmo ano, no mês de outubro, um grupo de pesquisadores japoneses organizou um *Workshop* sobre Grandes Desafios para a Inteligência Artificial (IA), em Tóquio. Foram discutidos e propostos novos problemas que representavam grandes desafios para os pesquisadores de IA. Uma das idéias mais discutidas foi a possibilidade de se utilizar um jogo de futebol entre robôs para promover ciência e tecnologia. Estudos apresentados durante esse *Workshop* mostraram que a idéia era viável, desejável e englobava diversas aplicações práticas.

No ano seguinte, em 1993, um grupo de pesquisadores japoneses, incluindo Minoru Asada, Yasuo Kuniyoshi e Hiroaki Kitano, organizou uma competição denominada *Robot J-League* (fazendo uma analogia à *J-League*, nome da Liga Japonesa de Futebol Profissional). Em um mês vários pesquisadores já se pronunciavam dizendo que a iniciativa devia ser estendida ao âmbito internacional. Assim surgia a *Robot World Cup Initiative* (ROBOCUP, 2007).

Em 1995, na Coréia, o professor Jong-Hwan Kim iniciou seus trabalhos sobre o Futebol de Robôs e a primeira competição internacional ocorreu em 1996, em Daejeon, Coréia (FIRA, 2007).

Em 1997, durante o Torneio *Micro-Robot Soccer'97*, foi criada a *Federation of International Robot-Soccer Association* (FIRA, 2007). O ano de 1997 é lembrado como um marco na história da Robótica e IA. Em maio desse ano, o supercomputador da IBM, o *DeepBlue* (IBM, 2007), derrotou Gary Kasparov, o humano campeão mundial de xadrez. Ainda nesse ano, a missão espacial da NASA chamada *Pathfinder* (NASA, 2007) obteve sucesso com a sonda *Sojourner*, primeiro sistema robótico autônomo para exploração da superfície de Marte.

O Futebol de Robôs posicionou-se, então, como o novo desafio para os pesquisadores da área de IA, criando um novo campo interdisciplinar de pesquisa em robótica autônoma inteligente, utilizando partidas de futebol. A *RoboCup* propôs o desafio de se desenvolver

robôs jogadores de futebol capazes de enfrentar a seleção humana campeã mundial em 2050. Os campeonatos da *RoboCup* acontecem anualmente e, paralelamente, acontece um congresso onde as equipes podem apresentar a contribuição científica de suas pesquisas na área de Futebol de Robôs. O mesmo ocorre com a FIRA, que promove congresso e competição anuais. Ambas as entidades têm o objetivo de promover o desenvolvimento de sistemas robóticos autônomos multi-agentes, integrando áreas diversas, como IA, visão computacional, robótica, controle, engenharia elétrica e mecânica, entre outras.

Tanto a FIRA quanto a *RoboCup* possuem diversas categorias de Futebol de Robôs. Esse trabalho está focado nas categorias de Futebol de Robôs *Simurosot* e *Mirosot*, propostas pela FIRA (2007).

1.1 Objetivo

O objetivo desse trabalho é verificar o comportamento de um agente robótico de Aprendizado por Reforço (AR) quando inserido em um ambiente de Futebol de Robôs real, através da comparação entre algoritmos de AR sem a utilização de heurísticas e com a utilização de heurísticas, técnica de aceleração do aprendizado proposta por Bianchi (2004).

1.2 Proposta

Esse trabalho propõe a utilização do ambiente de Futebol de Robôs simulado como plataforma para efetuar testes de desempenho de alguns algoritmos de AR. Os resultados desses testes serão utilizados como referência para a comparação com os resultados do desempenho dos algoritmos de AR quando o agente robótico for inserido em um ambiente real de Futebol de Robôs.

Através dos testes comparativos, pretende-se verificar a influência das heurísticas definidas *a priori* no desempenho dos algoritmos de AR, tanto em ambiente simulado como em ambiente real.

Também pretende-se investigar, nesse trabalho, a transferência do aprendizado adquirido em ambiente simulado para o ambiente real. As experiências adquiridas em simulação serão utilizadas para definir as heurísticas aplicadas nos algoritmos de AR em ambiente real.

1.3 Organização do trabalho

O capítulo 2 apresenta uma revisão bibliográfica sobre AR, descrevendo os algoritmos de AR fundamentais para o presente trabalho. O capítulo 3 apresenta propostas para aumentar o desempenho dos algoritmos descritos no capítulo 2 através de técnicas de aceleração do aprendizado.

No capítulo 4, são descritas, através de uma breve revisão bibliográfica, todas as partes que compõem o domínio do Futebol de Robôs, tanto em ambiente simulado, como em ambiente real.

O capítulo 5 descreve o time de Futebol de Robôs desenvolvido para que os experimentos desse trabalho pudessem ser feitos. As partes que compõem o sistema completo de Futebol de Robôs desenvolvido são descritas em detalhes.

O capítulo 6 apresenta a arquitetura do sistema de AR proposto nesse trabalho, assim como as peculiaridades de implementação dos algoritmos Q-learning, Q(λ), QS e HAQL, já conhecidos na literatura, além da introdução de dois novos algoritmos – HAQ(λ) e HAQS.

O capítulo 7 descreve os experimentos efetuados tanto em ambiente simulado, como em ambiente real e apresenta uma análise crítica dos resultados obtidos.

Por fim, o capítulo 8 conclui esse trabalho e sugere possíveis trabalhos futuros, baseando-se nos resultados obtidos.

2 APRENDIZADO POR REFORÇO

O AR é um campo multidisciplinar que reúne pesquisadores das áreas de engenharia de controle, neurociência, psicologia, entre outras áreas.

Para a área de IA, o AR é uma abordagem computacional para o estudo do problema de aprendizado de máquina. Segundo Sutton e Barto (1998), o crescente interesse dos pesquisadores em AR é justificado em parte pelo desafio em se desenvolver sistemas inteligentes que devem operar em ambientes dinâmicos reais.

2.1 O problema do aprendizado

Diferentemente de outros métodos de aprendizado de máquina, no AR não há exemplos de entrada, tampouco há a especificação das saídas. O aprendizado acontece a partir da interação de um agente aprendiz com o ambiente em que está inserido, o qual irá responder às ações do agente, retornando um reforço, também denominado de recompensa. A tarefa do agente é aprender um mapeamento das diferentes respostas do ambiente em relação às diferentes ações que esse agente venha a executar, buscando maximizar as recompensas recebidas a cada interação com o ambiente, acumuladas ao longo do tempo.

2.2 Propriedades de ambientes

Ambientes possuem diversas propriedades, as quais definem, entre outras características, a maneira a qual esse ambiente pode ter seu estado alterado e como o mesmo reage às ações executadas pelo agente.

Para que um agente possa interagir com o ambiente no qual está inserido, uma das primeiras tarefas é observar esse ambiente. Um ambiente onde o agente é capaz de observar todas as informações relevantes para a escolha da ação a ser executada, é denominado ambiente completamente observável. Esses ambientes completamente observáveis são convenientes, pois não é necessário que o agente armazene informações sobre o ambiente ao longo do tempo para ser capaz de selecionar a ação a ser executada.

No entanto, a presença de ruído e a imprecisão na leitura de sensores podem impossibilitar que o agente seja capaz de observar todas as variáveis do ambiente que definem um estado. Pode acontecer, ainda, que um agente não seja capaz de observar todas as variáveis que definem um estado. Logo, o ambiente é denominado parcialmente observável.

Depois de observar o estado atual em que se encontra e selecionar a ação a ser executada, é conveniente que o agente seja capaz de determinar o estado futuro do ambiente. Quando se pode determinar o estado futuro apenas com a observação do estado atual e com a informação de qual ação será executada pelo agente, o ambiente é denominado determinístico. Caso apenas as informações de estado atual e ação a ser executada pelo agente não sejam suficientes para determinar o estado futuro, esse ambiente é denominado não-determinístico.

Quando se trata da mudança de estados de um ambiente, caso seu estado seja alterado apenas pela execução da ação selecionada pelo agente, então é denominado ambiente estático. Do contrário, quando o estado do ambiente pode sofrer alterações enquanto o agente está selecionando qual ação deve ser executada, esse ambiente é denominado dinâmico.

2.3 Processos Markovianos de Decisão

Uma das maneiras de formalizar um agente de AR é através do Processo Markoviano de Decisão (PMD), descrito em Russell e Norvig (2004, cap. 17). Utilizando o PMD é possível modelar matematicamente um agente de AR, isto é, definir as funções matemáticas para os componentes de um agente de AR.

Segundo Kaelbling, Littman e Moore (1996), a propriedade de Markov define que as transições de estados são independentes do histórico de estados visitados e ações executadas anteriormente pelo agente.

Um PMD é definido da seguinte maneira:

- Um conjunto finito de estados *S*;
- Um conjunto finito de ações *A*;
- Uma função recompensa $R : S \times A \rightarrow \Re$;
- Uma função de transição de estados $T : S \times A \rightarrow \Pi(S)$, onde $\prod(S)$ é um mapeamento da transição de estados em probabilidades.

As funções de recompensa e transição de estados são definidas de acordo com as propriedades do ambiente, representando um modelo desse ambiente.

Um agente de AR, segundo Kaelbling, Littman e Moore (1996), é composto por quatro componentes: uma política, uma função de recompensa, uma função valor e uma função de transição de estados.

A política é responsável por selecionar a ação a ser executada, dependendo da situação em que o agente se encontra, para que uma meta seja alcançada. Seja qual for o estado inicial, essa política deve indicar uma seqüência de ações para se chegar ao objetivo, sendo essa seqüência a que maximiza o ganho de recompensas acumuladas ao longo do tempo até um estado terminal ou até que um critério de parada seja atingido. Uma política que maximiza esse ganho de recompensas acumuladas é tida como uma política ótima.

Uma política pode determinar uma seqüência de ações que não alcance o objetivo, ou ainda, que alcance o objetivo, mas cuja recompensa acumulada ao longo do tempo não seja a máxima possível. Dessa forma, a política é tida como não ótima, ou sub-ótima.

O objetivo do agente de AR é representado através da função de recompensa. A função de recompensa deve retornar um valor que representa, numericamente, uma punição ou uma gratificação para cada possível ação a ser executada em cada um dos possíveis estados.

Enquanto a função recompensa representa uma resposta imediata do ambiente à ação executada pelo agente, a função valor representa, para cada ação possível a ser executada em determinado estado, o valor máximo de recompensa acumulada que pode ser recebida ao longo do tempo, até que um estado terminal seja atingido.

Por fim, a função de transição de estados é capaz de retornar o estado futuro, seja qual for o estado atual e a ação executada. Essa função de transição de estados depende das propriedades do ambiente no qual o agente de AR está inserido.

Em um ambiente determinístico, a seleção de uma ação $a_t \in A$ em um estado $s_t \in S$ resulta sempre no mesmo estado futuro $s_{t+1} \in S$ e a transição $T(s_t, a_t, s_{t+1})$ ocorre com probabilidade 1. Para ambientes não-determinísticos, a seleção da ação $a_t \in A$ no estado $s_t \in S$ pode resultar em diferentes estados futuros e a transição $T(s_t, a_t, s_{t+1})$ é representada por uma distribuição de probabilidades que define os efeitos de cada ação a_t sobre o conjunto de estados *S*, que pode ser expressa como a probabilidade de alcançar o estado s_{t+1}, dado o estado atual s_t e considerando que a ação a_t seja executada, $P(s_{t+1} \mid s_t, a_t)$.

2.4 Determinação de uma Política Ótima

Com o agente de AR modelado por um PMD, admitindo que as funções de recompensa e transição de estados são conhecidas e o ambiente seja determinístico, é possível

determinar uma política ótima $\pi : S \rightarrow A$, para selecionar a próxima ação $a \in A$ quando observado o estado $s \in S$, ou seja, $\pi(s) = a$.

Pode-se definir o valor acumulado $V^{\pi}(s_t)$, que utiliza uma política arbitrária π, a partir de um estado inicial s_t, como mostrado na equação (2.1):

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

onde:

- r_{t+i} é a seqüência de recompensas recebidas a partir do estado s_t, utilizando a política π para selecionar ações;
- γ é um fator de desconto que determina o quanto as recompensas futuras serão consideradas, admitindo o intervalo $0 \leq \gamma < 1$.

Para que uma política ótima, denominada π^*, seja determinada, a tarefa do agente de AR é aprender uma política π que maximize o valor acumulado $V^{\pi}(s)$, para qualquer estado $s \in S$, como mostrado na equação (2.2):

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

Dessa forma, denomina-se $V^*(s)$ como sendo o valor acumulado ótimo, aquele que resulta de uma política ótima π^*.

2.5 Métodos para a solução do problema de Aprendizado por Reforço

Segundo Sutton e Barto (1998), existem três classes de métodos para resolver o problema de AR: programação dinâmica, métodos de Monte Carlo e métodos de diferenças temporais.

Os métodos de programação dinâmica têm uma base matemática consistente, embora necessitem de um modelo completo do ambiente, ou seja, o conhecimento das funções de recompensa e transição de estados.

Métodos de Monte Carlo são conceitualmente simples e não requerem um modelo de ambiente, mas não são apropriados para computação iterativa, onde uma política é aprendida passo a passo, enquanto os métodos de diferenças temporais não requerem um modelo de ambiente e são, essencialmente, métodos iterativos.

Sutton e Barto (1998) apresenta uma extensa revisão bibliográfica, descrevendo detalhadamente as três classes de métodos para a solução do problema de AR.

Kaelbling, Littman e Moore (1996) e Ribeiro (1999) também apresentam discussões sobre diversos algoritmos de AR. São discutidos, a seguir, os algoritmos fundamentais para a compreensão desse trabalho.

2.5.1 O Método de Diferenças Temporais – TD

Apresentado por Sutton (1988), o Método de Diferenças Temporais (TD) é fundamentado por uma base matemática consistente. Esse método calcula, de forma iterativa, uma estimativa $\hat{V}^{\pi}$ do valor acumulado V^{π}, selecionando as ações seguindo uma política π. O Método TD exige apenas que o próximo passo s_{t+1} seja observado para atualizar a estimativa $\hat{V}^{\pi}$, sem a necessidade de um modelo do ambiente. A regra de atualização da estimativa $\hat{V}^{\pi}$ do Método TD mais simples, conhecido por TD(0), acontece de acordo com as equações (2.3) e (2.4):

$$\hat{V}^{\pi}_{t+1}(s_t) \leftarrow \hat{V}^{\pi}_{t}(s_t) + \alpha_t \delta^0_t \tag{2.3}$$

$$\delta^0_t = r_t + \gamma \hat{V}^{\pi}_{t}(s_{t+1}) - \hat{V}^{\pi}_{t}(s_t) \tag{2.4}$$

Onde:

- s_t é o estado atual;
- s_{t+1} é o estado futuro;
- $\hat{V}^{\pi}_{t}$ é a estimativa atual do valor acumulado V^{π} na iteração t;
- $\hat{V}^{\pi}_{t+1}$ é a estimativa futura do valor acumulado V^{π} na iteração $t+1$;
- α_t é a taxa de aprendizado na iteração t, sendo $0 < \alpha \leq 1$;
- δ^0_t é chamado de diferença temporal, ou erro TD(0), na iteração t, que representa uma estimativa da diferença entre a estimativa de valor atual $\hat{V}^{\pi}_{t}(s_t)$ e o valor acumulado esperado $r_t + \gamma \hat{V}^{\pi}_{t}(s_{t+1})$;
- r_t é a recompensa recebida na iteração t, quando uma ação a_t, selecionada a partir da política π, é executada no estado s_t, atingindo o estado s_{t+1};
- γ é um fator de desconto que determina o quanto as recompensas futuras serão consideradas, admitindo o intervalo $0 \leq \gamma < 1$.

A Figura 2.1 apresenta o algoritmo TD(0) para calcular a estimativa $\hat{V}^{\pi}$.

Para todo estado *s*, inicialize $V^{\pi}(s)$ com zero
Observe o estado atual s_t
Repita infinitamente:
(1) Selecione uma ação a_t de acordo com a política π
(2) Receba a recompensa imediata r_t
(3) Observe o novo estado s_{t+1}
(4) Compute o erro TD(0) conforme a equação (2.4)
(5) Atualize a estimativa do valor acumulado V^{π} conforme a equação (2.3)
(6) $s_t \leftarrow s_{t+1}$

Figura 2.1 – Algoritmo TD(0)

As equações (2.3) e (2.4) mostram que a estimativa futura $\hat{V}_{t+1}^{\pi}$ é atualizada gradualmente, sendo ponderada com a estimativa atual $\hat{V}_t^{\pi}$ através da taxa de aprendizado α. Em ambientes determinísticos, o valor 1 é atribuído a α, o erro TD(0) δ^0 é calculado e a estimativa $\hat{V}_t^{\pi}$ é atualizada com o valor da estimativa $\hat{V}_{t+1}^{\pi}$. Para o caso de ambientes não-determinísticos, a taxa de aprendizado α deve ter seu valor menor que 1 para que a estimativa futura $\hat{V}_{t+1}^{\pi}$ seja ponderada com a estimativa anterior $\hat{V}_t^{\pi}$. Dessa forma, com a taxa de aprendizado $\alpha < 1$, todas as iterações anteriores são consideradas para calcular a estimativa de valor acumulado $\hat{V}^{\pi}$.

Segundo Watkins e Dayan (1992), a taxa de aprendizado α deve decair ao longo do tempo para satisfazer duas condições de convergência de algoritmos iterativos utilizados em ambientes não-determinísticos.

Uma maneira de se obter uma taxa de aprendizado α que decai ao longo do tempo é através da equação (2.5), como descrito por Mitchell (1997):

$$\alpha_t = \frac{1}{1 + visitas_t(s)} \tag{2.5}$$

Onde $visitas_t(s)$ é o número de visitas ocorridas ao estado *s* até a iteração *t*.

Pode-se expandir o Método TD(0), que calcula a diferença após uma iteração observando o estado futuro s_{t+1}, para uma formulação mais geral, baseando-se nas descrições de Sutton (1988), que considera a influência das diferenças temporais obtidas *n* estados futuros à frente, sendo *n* o número de iterações. A regra de atualização para calcular a estimativa $\hat{V}^{\pi}$, mostrada na equação (2.6), é muito similar à regra da equação (2.3):

$$\hat{V}_{t+1}^{\pi}(s_t) \leftarrow \hat{V}_t^{\pi}(s_t) + \alpha_t \delta_t^n \tag{2.6}$$

No entanto, a diferença temporal δ_t^n é definida para considerar os erros TD(0) dos estados futuros, *n* iterações à frente, conforme a equação (2.7):

$$\delta_t^n = \delta_t^0 + \gamma\delta_{t+1}^0 + \gamma^2\delta_{t+2}^0 + \gamma^3\delta_{t+3}^0 + ... = \delta_t^0 + \sum_{n=1}^{\infty}\gamma^n\delta_{t+n}^0 \qquad (2.7)$$

A partir da definição da diferença temporal δ^n, Sutton (1988) apresentou uma formulação que desconta a influência das diferenças temporais futuras independente do fator γ, utilizando um fator λ, admitindo o intervalo $0 \leq \lambda \leq 1$, originando o Método TD(λ). Enquanto o fator γ representa o desconto de recompensas futuras, o fator λ representa o desconto das diferenças temporais futuras. Dessa forma, é possível definir o erro TD(λ) δ^λ, com base nas descrições de Peng e Williams (1996) e Ribeiro (1999), de acordo com a equação (2.8):

$$\delta_t^\lambda = \delta_t^0 + \gamma\lambda\delta_{t+1}^0 + \gamma^2\lambda^2\delta_{t+2}^0 + \gamma^3\lambda^3\delta_{t+3}^0 + ... = \delta_t^0 + \sum_{n=1}^{\infty}(\gamma\lambda)^n\delta_{t+n}^0 \qquad (2.8)$$

Peng e Williams (1996) demonstra, ainda, que é possível determinar o erro TD(λ) δ_t^λ recursivamente, conforme a equação (2.9):

$$\delta_t^\lambda = \delta_t^0 + \gamma\lambda\delta_{t+1}^\lambda \qquad (2.9)$$

A regra de atualização da estimativa $\hat{V}^\pi$ que utiliza o erro TD(λ) δ^λ, que por sua vez considera as recompensas futuras e as diferenças temporais futuras, descontadas ao longo do tempo pelos fatores γ e λ respectivamente, é definida, então, conforme a equação (2.10):

$$\hat{V}_{t+1}^\pi(s_t) \leftarrow \hat{V}_t^\pi(s_t) + \alpha_t\delta_t^\lambda \qquad (2.10)$$

Entretanto, não é possível implementar diretamente a regra de atualização da equação (2.10), pois ela é não causal, o que significa que as diferenças temporais futuras δ_{t+1}^0, δ_{t+2}^0, δ_{t+3}^0, ..., δ_{t+n}^0 são utilizadas para atualizar a estimativa $\hat{V}_{t+1}^\pi$ na iteração *t*. Para que o cálculo das atualizações possa ser feito iterativamente, utiliza-se o rastro de elegibilidade, discutido inicialmente por Watkins (1989). Esse rastro de elegibilidade é uma variável de memória associada a cada estado $s \in S$. O rastro de elegibilidade pode ser definido por acumulação (*accumulating trace*), conforme a equação (2.11), ou ainda, por substituição (*replacing trace*), conforme a equação (2.12):

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & se\ s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & se\ s = s_t \end{cases} \qquad (2.11)$$

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & se\ s \neq s_t \\ 1 & se\ s = s_t \end{cases} \quad (2.12)$$

Um estudo detalhado sobre rastros de elegibilidade é feito por Sutton e Barto (1998), analisando as vantagens e desvantagens de cada uma das definições anteriores, mostrando que, em alguns casos, o rastro de elegibilidade por substituição apresenta um desempenho significativamente maior em relação ao rastro de elegibilidade por acumulação.

O rastro de elegibilidade armazena a informação sobre quais estados foram visitados recentemente. Determina-se o quão recente um estado *s* foi visitado através de sua elegibilidade *e(s)*, que decai em $\gamma\lambda$ a cada iteração. Dessa forma, a atualização de $\hat{V}^{\pi}_{t+1}$, calculada pelo erro TD(0) δ^0_t, é propagada proporcionalmente aos estados recentemente visitados, conforme mostra a equação (2.13), através do rastro de elegibilidade $e_t(s)$:

$$\hat{V}^{\pi}_{t+1}(s) \leftarrow \hat{V}^{\pi}_{t}(s) + \alpha_t \delta^0_t e_t(s), \forall s \in S \quad (2.13)$$

A prova matemática do algoritmo TD(λ) que calcula a estimativa $\hat{V}^{\pi}$ e converge para V^*, pode ser verificada em Dayan (1992). O algoritmo TD(λ) é apresentado na Figura 2.2.

Para todo estado *s*, inicialize $V^{\pi}(s)$ e *e(s)* com zero
Observe o estado atual s_t
Repita infinitamente:
(1) Selecione uma ação a_t de acordo com a política π
(2) Receba a recompensa imediata r_t
(3) Observe o novo estado s_{t+1}
(4) Compute o erro TD(0) conforme a equação (2.4)
(5) Atualize o rastro de elegibilidade conforme uma das regras (2.11) ou (2.12)
(6) Para todo estado *s*:
  (6.1) Atualize a estimativa do valor acumulado $V^{\pi}(s)$ conforme a equação (2.13)
  (6.2) Compute o decaimento do rastro $e(s) \leftarrow \gamma\lambda e(s)$
(7) $s_t \leftarrow s_{t+1}$

Figura 2.2 – Algoritmo TD(λ)

2.5.2 Q-learning

Walkins (1989) propôs o algoritmo Q-learning, que determina uma função ação-valor *Q*, que considera as transições ocorridas a cada iteração em termos do par estado-ação, em contraste com o Método TD, que considera as transições apenas no conjunto de estados. Dessa forma, o algoritmo Q-learning é capaz de estimar diretamente a função ação-valor ótima Q^*, simplesmente aprendendo a função ação-valor *Q*, independente da política π que está sendo seguida, diferentemente do Método TD, que calcula a estimativa de valor acumulado $\hat{V}^{\pi}$ para uma dada política π.

Admitindo que $V^*(s)$ é o valor acumulado ótimo de s, considerando que a ação a foi selecionada a partir de uma política ótima π^*, pode-se relacionar a função ação-valor ótima $Q^*(s, a)$ e o valor acumulado ótimo $V^*(s)$ conforme a equação (2.14):

$$V^*(s) = \max_a Q^*(s,a) \tag{2.14}$$

Onde $Q^*(s, a)$ representa o máximo valor acumulado de recompensas futuras a ser recebido quando a ação a é selecionada por uma política ótima π^* no estado s. Pode-se, então, definir a política ótima π^* reescrevendo a equação (2.2) em termos de $Q(s, a)$, de acordo com a equação (2.15):

$$\pi^*(s) = \arg\max_a Q(s,a), \forall s \in S \tag{2.15}$$

Logo, é possível aproximar a estimativa $\hat{Q}^*$ iterativamente até Q^*, de maneira muito similar ao Método TD(0), reescrevendo as equações (2.3) e (2.4) em termos da função ação-valor Q, como mostrado nas equações (2.16) e (2.17):

$$\hat{Q}^*_{t+1}(s_t, a_t) \leftarrow \hat{Q}^*_t(s_t, a_t) + \alpha_t \delta^0_t \tag{2.16}$$

$$\delta^0_t = r_t(s_t, a_t) + \gamma \max_a \hat{Q}^*_t(s_{t+1}, a) - \hat{Q}^*_t(s_t, a_t) \tag{2.17}$$

Onde:

- a_t é a ação executada na iteração t;
- $\hat{Q}^*_t$ é a estimativa atual do valor acumulado Q^* na iteração t;
- $\hat{Q}^*_{t+1}$ é a estimativa futura do valor acumulado Q^* na iteração $t+1$;
- α_t é a taxa de aprendizado na iteração t, sendo $0 < \alpha \leq 1$, que decai conforme a regra da equação (2.5);
- δ^0_t é o erro TD(0) na iteração t, reescrito em termos da função ação-valor Q;
- $r_t(s_t, a_t)$ é a recompensa recebida na iteração t, quando uma ação a_t é executada no estado s_t, atingindo o estado s_{t+1};
- γ é um fator de desconto que determina o quanto as recompensas futuras serão consideradas, admitindo o intervalo $0 \leq \gamma < 1$.

A maneira a qual as ações são selecionadas pode ser, por exemplo, através da utilização de uma estratégia de exploração aleatória gulosa, conhecida como ε – *Greedy*,

muito utilizada em implementações do algoritmo Q-learning. Ações selecionadas de maneira aleatória favorecem a exploração do ambiente, enquanto ações selecionadas de maneira gulosa tiram proveito das experiências das iterações anteriores do algoritmo, a explotação. A estratégia ε – *Greedy* determina uma política π que define uma relação entre exploração e explotação do ambiente, através do parâmetro ε, admitindo o intervalo $0 \leq \varepsilon \leq 1$, selecionando ações conforme a equação (2.18):

$$\hat{\pi}_t^*(s_t) = \begin{cases} \arg\max_a \hat{Q}_t^*(s_t, a) & se \;\; p > \varepsilon \\ a_{random} & se \;\; p \leq \varepsilon \end{cases} \tag{2.18}$$

Onde:

- p é um valor determinado aleatoriamente, com distribuição de probabilidade uniforme sobre o intervalo [0, 1];
- a_{random} é uma ação pertencente ao conjunto de ações A, selecionada de maneira aleatória;
- $\hat{\pi}_t^*$ é a estimativa da política ótima, derivada da estimativa $\hat{Q}_t^*$ da função ação-valor ótima Q^*, na iteração t.

Dessa forma, como a estimativa $\hat{Q}^*$ da função ação-valor ótima converge para Q^*, conforme a prova matemática apresentada por Watkins e Dayan (1992), a política ótima π^* é aprendida iterativamente, diretamente através da função ação-valor Q.

O algoritmo Q-learning é mostrado na Figura 2.3.

Para todo estado s e ação a, inicialize $Q(s, a)$ com zero
Observe o estado atual s_t
Repita infinitamente:
(1) Selecione uma ação a_t de acordo com a política derivada de Q, (ex. (2.18))
(2) Receba a recompensa imediata r_t
(3) Observe o novo estado s_{t+1}
(4) Compute o erro TD(0) conforme a equação (2.17)
(5) Atualize a estimativa da função ação-valor Q^* conforme a equação (2.16)
(6) $s_t \leftarrow s_{t+1}$

Figura 2.3 – Algoritmo Q-learning

Apesar de convergirem para uma política ótima, os algoritmos de AR descritos nesse capítulo necessitam de um número de iterações muito elevado e, consequentemente, muito tempo para convergir. O próximo capítulo apresenta técnicas para aceleração do AR.

3 ACELERAÇÃO DO APRENDIZADO POR REFORÇO

Os algoritmos de AR apresentados no capítulo anterior exigem um número de iterações muito grande para convergência. O custo computacional aumenta à medida que o ambiente se torna maior e mais complexo, em termos do conjunto de estados e do conjunto de ações. O aumento do conjunto de estados, do conjunto de ações, ou ainda, do número de agentes presentes no ambiente, também são fatores que elevam o custo computacional dos algoritmos de AR. Logo, para uma convergência mais rápida dos algoritmos, é necessário o uso de técnicas de aceleração do AR.

Sutton e Barto (1998), Bianchi (2004) e Ribeiro (1999) apresentam uma ampla revisão bibliográfica sobre diversas técnicas de aceleração do AR. Esse capítulo descreve apenas as técnicas de aceleração do AR relacionadas ao algoritmo Q-learning, que foram utilizadas no desenvolvimento desse trabalho.

3.1 Aceleração por Abstração

É possível utilizar abstrações para acelerar os algoritmos de AR. A abstração pode ocorrer de forma estrutural, onde estados são agregados para formar regiões maiores, ou de forma temporal, definindo-se macro-ações que incorporam a tarefa de várias ações, reduzindo significativamente o tamanho efetivo do conjunto de estados e do conjunto de ações, resultando em um menor custo computacional.

Este trabalho faz uso de abstrações para reduzir o tamanho do conjunto de estados e do conjunto de ações. Essas abstrações são descritas em detalhes no capítulo 6, onde é definida a arquitetura do sistema de AR utilizado nesse trabalho.

3.2 Aceleração por Generalizações

A aceleração por generalizações ocorre quando a experiência de uma iteração é espalhada para outros estados que não o da iteração atual. Esse espalhamento pode ser tanto temporal, quando a experiência é propagada para os estados visitados recentemente, como espacial, quando a experiência é distribuída a outros estados considerando alguma medida de similaridade.

3.2.1 Q(λ)-learning

Um exemplo de generalização temporal é o algoritmo Q(λ)-learning, uma extensão do algoritmo Q-learning que utiliza a propagação temporal das atualizações existente no algoritmo TD(λ). As duas abordagens mais conhecidas, que combinam os algoritmos Q-learning e TD(λ), são o algoritmo Q(λ) de Watkins (1989) e o Q(λ) Peng e Williams (1996).

Para que as diferenças temporais futuras, descontadas de λ, sejam consideradas no algoritmo Q(λ), são utilizados o erro TD(0) δ^0 do algoritmo Q-learning, de acordo com a equação (2.17), assim como a definição do erro TD(λ) δ^λ da equação (2.8). Então, a regra de atualização do algoritmo Q-learning da equação (2.16) é reescrita, originando a regra de atualização do algoritmo Q(λ), conforme a equação (3.1):

$$\hat{Q}^*_{t+1}(s_t, a_t) \leftarrow \hat{Q}^*_t(s_t, a_t) + \alpha_t \delta^\lambda_t \tag{3.1}$$

Analogamente ao algoritmo TD(λ), não se pode calcular a estimativa $\hat{Q}^*$ iterativamente e, em ambas as abordagens de Watkins (1989) e Peng e Williams (1996), é utilizado o rastro de elegibilidade do algoritmo TD(λ). Porém, esse rastro é modificado para considerar o par estado-ação *(s, a)*. Então, as equações (2.11) e (2.12) são reescritas em termos de *(s, a)*, conforme as equações (3.2) e (3.3):

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & se\ (s,a) \neq (s_t, a_t) \\ \gamma\lambda e_{t-1}(s,a) + 1 & se\ (s,a) = (s_t, a_t) \end{cases} \tag{3.2}$$

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & se\ (s,a) \neq (s_t, a_t) \\ 1 & se\ (s,a) = (s_t, a_t) \end{cases} \tag{3.3}$$

Pela regra da equação (2.14), entende-se que a estimativa de valor acumulado $\hat{V}^*$ é, na verdade, o valor acumulado V^π que segue a estimativa de política ótima $\hat{\pi}^*$, que pode não representar uma política gulosa. Por esse motivo, a abordagem de Watkins (1989) reinicializa o rastro de elegibilidade sempre que uma ação é selecionada aleatoriamente, enquanto a variante de Peng e Williams (1996) não diferencia uma ação aleatória de uma ação selecionada seguindo uma política, não reinicializando o rastro de elegibilidade, pois assume que a política utilizada é uma política gulosa, cujas ações são selecionadas através da estratégia ε *– Greedy*.

Como conseqüência, de acordo com Sutton e Barto (1998), para uma política fixa não gulosa, a estimativa $\hat{Q}^*$ do algoritmo Q(λ) de Peng e Williams (1996) não converge nem para

Q^{π}, tampouco para Q^{*}, mas para algo híbrido entre as duas. No entanto, Sutton e Barto (1998) ainda afirma que, para uma política que se torne gulosa ao longo do tempo, o método de Peng e Williams (1996) pode convergir para Q^{*}, além de apresentar um desempenho significativamente melhor que o algoritmo Q(λ) de Watkins (1989).

A regra de atualização para o algoritmo Q(λ) de Watkins (1989) é apresentada na equação (3.4):

$$\hat{Q}^{*}_{t+1}(s_t, a_t) \leftarrow \hat{Q}^{*}_{t}(s_t, a_t) + \alpha_t \delta_t^{0} e_t(s,a), \forall (s,a) \in S \tag{3.4}$$

O algoritmo Q(λ) de Watkins (1989) é mostrado na Figura 3.1.

Para todo estado *s* e ação *a*, inicialize *e(s, a)* e *Q(s, a)* com zero
Observe o estado atual s_t
Repita infinitamente:
- (1) Selecione uma ação a_t de acordo com a política derivada de *Q*, (ex. (2.18))
- (2) Receba a recompensa imediata r_t
- (3) Observe o novo estado s_{t+1}
- (4) Compute o erro TD(0) conforme a equação (2.17)
- (5) Atualize o rastro $e(s_t, a_t)$ conforme uma das regras (3.2) ou (3.3)
- (6) Para todo par estado-ação *(s, a)*:
  - (6.1) Atualize a estimativa da função ação-valor Q^{*} conforme a equação (3.4)
  - (6.2) Se $a_t = a^{*}$: compute o decaimento do rastro: $e(s, a) \leftarrow \gamma\lambda e(s, a)$
  - (6.3) Senão: $e(s, a) \leftarrow 0$
- (7) $s_t \leftarrow s_{t+1}$

Figura 3.1 – Algoritmo Q(λ) de Watkins (1989)

Para o caso da abordagem de Peng e Williams (1996), é assumido que uma política gulosa é utilizada para determinar as ações e demonstrado que, a partir das equações (2.8), (2.9) e considerando a equação (2.14), pode-se definir o erro TD(λ) δ^{λ} iterativamente, como mostrado na equação (3.5):

$$\delta_t^{\lambda} = r_t(s_t, a_t) + \gamma \max_a \hat{Q}^{*}_{t}(s_{t+1}, a) - \hat{V}^{*}_{t}(s_t) \tag{3.5}$$

Dessa forma, no algoritmo Q(λ) de Peng e Wiiliams (1996), a atualização da estimativa $\hat{Q}^{*}$ ocorre através de duas regras de atualização. Para todos os pares estado-ação *(s, a)*, uma das regras de atualização utiliza o erro TD(λ) δ^{λ}, determinado iterativamente pela equação (3.5), para atualizar as estimativas $\hat{Q}^{*}(s,a)$, fazendo uso do rastro de elegibilidade *e(s, a)*, conforme mostra a equação (3.6):

$$\hat{Q}^{*}_{t+1}(s_t, a_t) \leftarrow \hat{Q}^{*}_{t}(s_t, a_t) + \alpha_t \delta_t^{\lambda} e_t(s,a), \forall (s,a) \in S \tag{3.6}$$

A outra regra de atualização do algoritmo Q(λ) de Peng e Wiiliams (1996), mostrada na equação (3.7), utiliza o erro TD(0) δ^{0} para atualizar a estimativa $\hat{Q}^{*}(s_t, a_t)$:

$$\hat{Q}^*_{t+1}(s_t, a_t) \leftarrow \hat{Q}^*_{t+1}(s_t, a_t) + \alpha_t \delta^0_t \tag{3.7}$$

A Figura 3.2 apresenta o algoritmo Q(λ) de Peng e Williams (1996).

Para todo estado *s* e ação *a*, inicialize *e(s, a)* e *Q(s, a)* com zero
Observe o estado atual s_t
Repita infinitamente:
- (1) Selecione uma ação a_t de acordo com a política derivada de *Q*, (ex. (2.18))
- (2) Receba a recompensa imediata r_t
- (3) Observe o novo estado s_{t+1}
- (4) Compute o erro TD(0) conforme a equação (2.17)
- (5) Compute o erro TD(λ) conforme a equação (3.5)
- (6) Para todo par estado-ação *(s, a)*:
  - (6.1) Compute o decaimento do rastro: $e(s, a) \leftarrow \gamma\lambda e(s, a)$
  - (6.2) Atualize a estimativa da função $Q^*(s, a)$ conforme a equação (3.6)
- (7) Atualize a estimativa da função $Q^*(s_t, a_t)$ conforme a equação (3.7)
- (8) Atualize o rastro $e(s_t, a_t)$ conforme uma das regras (3.2) ou (3.3)
- (9) $s_t \leftarrow s_{t+1}$

Figura 3.2 – Algoritmo Q(λ) de Peng e Wiiliams (1996)

3.2.2 QS-learning

O algoritmo QS-learning, proposto em Ribeiro e Szepervári (1996), é uma abordagem do algoritmo Q-learning combinada com generalizações espaciais, que espalha a atualização da estimativa $\hat{Q}^*(s,a)$ para outros pares estado-ação *(x, u)* não envolvidos na iteração atual *t*, de acordo com a similaridade entre pares estado-ação. Para tal, a similaridade é determinada através de uma função de espalhamento $\sigma(x,u,s,a)$, sendo $0 \leq \sigma(x,u,s,a) \leq 1$.

Caso a similaridade ocorra apenas no conjunto de estados, a função $\sigma(x,u,s,a)$ é definida conforme a equação (3.8). Ainda, se a similaridade ocorrer apenas no conjunto de ações, a função $\sigma(x,u,s,a)$ é definida de acordo com a equação (3.9):

$$\sigma(x,u,s,a) = g_x(x,s)\partial(u,a) \tag{3.8}$$

$$\sigma(x,u,s,a) = \partial(x,s)g_u(u,a) \tag{3.9}$$

Onde $\partial(.,.)$ é o Delta de Kronecker, podendo assumir os valores $\partial(i,j) = 1$ se $i = j$, ou $\partial(i,j) = 0$ se $i \neq j$.

Por último, quando a similaridade ocorre em ambos os conjuntos, a função de espalhamento $\sigma(x,u,s,a)$ é definida de acordo com a equação (3.10):

$$\sigma(x,u,s,a) = g_{(x,u)}((x,s),(u,a)) \tag{3.10}$$

A função $g_x(x,s)$ determina o grau de similaridade entre estados, enquanto a função $g_u(u,a)$ determina essa proporção entre ações e a função $g_{(x,u)}((x,s),(u,a))$ entre pares

estado-ação. Por simplificação, a partir desse momento, será considerado que as similaridades ocorrem apenas no conjunto de estados, conforme a equação (3.8) e a função $g_x(x,s)$ será descrita apenas como $g(x,s)$. Dessa maneira, pode-se definir a função $g(x,s)$ de similaridade entre estados conforme a equação (3.11):

$$g(x,s_t) = \tau^d, \forall x \in S \tag{3.11}$$

Onde:

- s_t é o estado que está sendo visitado;
- x é qualquer estado pertencente ao conjunto de estados, cuja proporção de similaridade com o estado atual s_t é verificada pela função g;
- d é um valor que quantifica a similaridade entre os estados x e s_t, considerando, por exemplo, a vizinhança entre esses estados;
- τ pode ser uma constante ou uma variável cujo valor decai ao longo do tempo.

Ribeiro, Pegoraro e Reali-Costa (2002) apresenta testes e comparações de uma variante do algoritmo QS para um valor fixo para $\tau = 0.7$, assim como para um valor de $\tau = 0.7$ que decai linearmente em relação ao número de iterações. A prova matemática apresentada por Ribeiro e Szepesvári (1996) demonstra que o algoritmo QS converge para valores ótimos desde que o valor de τ decaia, no pior caso, à mesma velocidade que a taxa de aprendizado α.

A regra de atualização e o erro TD(0) δ^0 do algoritmo QS podem, então, ser definidos com base nas equações (2.16) e (2.17) do algoritmo Q-learning, conforme é mostrado nas equações (3.12) e (3.13):

$$\hat{Q}^*_{t+1}(x,u) \leftarrow \hat{Q}^*_t(x,u) + \sigma(x,u,s_t,a_t)\alpha_t\delta^0_t \tag{3.12}$$

$$\delta^0_t = r_t(s_t,a_t) + \gamma \max_a \hat{Q}^*_t(s_{t+1},a) - \hat{Q}^*_t(x,u) \tag{3.13}$$

Pode-se observar que, caso a função $g(x,s)$ não considere nenhuma similaridade entre os estados, seu valor será zero para qualquer outro estado x que não seja o próprio estado s_t, resultando em uma função de espalhamento $\sigma(x,u,s,a)$ também igual a zero. Nesse caso, o algoritmo QS, descrito na Figura 3.3, torna-se idêntico ao algoritmo Q-learning.

Para todo estado s e ação a, inicialize $Q(s, a)$ com zero
Observe o estado atual s_t
Repita infinitamente:
(1) Selecione uma ação a_t de acordo com a política derivada de Q, (ex. (2.18))
(2) Receba a recompensa imediata r_t
(3) Observe o novo estado s_{t+1}
(4) Para todo par estado-ação (x,u):
(4.1) Compute o erro TD(0) conforme a equação (3.13)
(4.2) Atualize a estimativa da função ação-valor Q^* conforme a equação (3.12)
(5) $s_t \leftarrow s_{t+1}$

Figura 3.3 – Algoritmo QS

3.3 Uso de heurísticas para aceleração do aprendizado por reforço

Outra forma de utilizar o conhecimento sobre o domínio para acelerar os algoritmos de AR é através do uso de heurísticas, como proposto por Bianchi (2004). As heurísticas podem ser definidas por um especialista, a partir de conhecimento *a priori* sobre o domínio. Outra possibilidade é a utilização de métodos de extração de conhecimento sobre o domínio para definição da heurística, inclusive durante o aprendizado, que Bianchi (2004) denominou de métodos de "Heurística a partir de X".

Essa técnica pode ser definida como uma maneira de se resolver um PMD utilizando uma função heurística $H : S \times A \rightarrow \Re$ para influenciar a seleção das ações durante o aprendizado. Bianchi (2004) demonstra que utilização de heurísticas em algoritmos de AR acelera o aprendizado quando a heurística é capaz de direcionar a seleção das ações ao objetivo do agente. Do contrário, caso a heurística não seja adequada, o resultado é um atraso no aprendizado, que não impede os algoritmos de convergir para uma política ótima.

3.3.1 A função heurística *H*

A função heurística $H_t(s_t, a_t)$ determina o quão desejável é a seleção da ação a_t quando no estado s_t. Essa função modifica as estratégias de seleção das ações dos algoritmos de AR, determinando que, para cada estado s_t, a seleção da ação a_t sugerida pela heurística pode direcionar o agente ao seu objetivo. Dessa forma, a função heurística H representa uma política heurística $\pi^H(s_t) = a_t$.

Utilizando a estratégia de exploração $\in$ – *Greedy*, descrita na equação (2.18), pode-se exemplificar a influência da função heurística H na seleção das ações modificando essa estratégia $\in$ – *Greedy*, conforme a equação (3.14):

$$\hat{\pi}_t^*(s_t) = \begin{cases} \arg\max_a \left[F_t(s_t, a_t) \rhd\lhd \xi H_t(s_t, a_t)^\beta \right] & se \;\; p > \varepsilon \\ a_{random} & se \;\; p \leq \varepsilon \end{cases} \tag{3.14}$$

onde as mudanças em relação à estratégia $\in$ *– Greedy* da equação (2.18) são:

- $F : S \times A \to \Re$ é uma estimativa de qualquer função que represente o valor acumulado máximo de recompensa. Caso $F_t(s_t, a_t) \equiv \hat{Q}_t^*(s_t, a_t)$, por exemplo, tem-se um algoritmo similar ao Q-learning;
- $H : S \times A \to \Re$ é a função heurística propriamente dita;
- $\rhd\lhd$ é uma função matemática qualquer (adição, por exemplo), capaz de operar números reais e resultar em um conjunto ordenado, que suporte a operação de maximização;
- ξ e β são variáveis reais que ponderam a influência da função heurística;

Embora possa ser utilizada qualquer função que opere números reais em $\rhd\lhd$, visto que as funções *F* e *H* assumem valores reais, Bianchi (2004) diz que, para o caso da operação de adição, a análise dos valores de *H* que influenciam a seleção das ações pode ser feita de maneira similar à análise feita em algoritmos de busca informada. Já para o caso da operação de multiplicação, se a função *F* for multiplicada por uma heurística positiva, não se pode afirmar se a ação apontada pela heurística terá sua importância reduzida ou aumentada, pois, caso o valor de *F* seja negativo, a ação terá sua importância diminuída.

Bianchi (2004) demonstra que a convergência dessa formulação é garantida e as provas de convergência dos algoritmos de AR existentes continuam válidas.

3.3.2 Definição da função heurística *H*

Embora a função heurística *H* influencie apenas a seleção das ações, existem alguns fatores que restringem a definição dos valores dessa função heurística, que pode ser não-estacionária ou estacionária, para assegurar a convergência dos algoritmos de AR.

Para definir esses valores da função heurística *H*, Bianchi (2004) definiu a regra mostrada na equação (3.15):

$$H_t(s_t, a_t) = \begin{cases} \max_a F_t(s_t, a) - F_t(s_t, a_t) + \eta & se \;\; a_t = \pi^H(s_t) \\ 0 & caso \; contrário \end{cases} \tag{3.15}$$

Onde $a_t = \pi^H(s_t)$ é a ação sugerida pela política heurística e η é um valor pequeno o suficiente para que a função heurística continue sugerindo a seleção da ação a_t.

No entanto, é necessário que limites máximos e mínimos dos valores da função heurística *H* sejam estipulados, do contrário, o algoritmo pode não convergir caso a política heurística π^H não coincida com a política ótima π^*.

Bianchi (2004) define o erro L_H, causado pelo uso de heurísticas em algoritmos de AR na aproximação da função *F* para uma política ótima π^*, conforme a equação (3.16):

$$L_H(s_t) = F_t(s_t, \pi^*(s_t)) - F_t(s_t, \pi^H(s_t)), \forall s_t \in S \tag{3.16}$$

Dessa forma, caso a política heurística π^H coincida com a política ótima π^*, o erro L_H é igual a zero e o algoritmo converge para π^*. No entanto, caso a política heurística π^H não coincida com a política ótima π^*, haverá algum erro de aproximação. Restringindo os valores de heurística entre $h_{\min} \le H_t(s_t, a_t) \le h_{\max}$, os valores de recompensa entre $r_{\min} \le r(s_t, a_t) \le r_{\max}$, o fator de desconto γ entre $0 \le \gamma < 1$ e definindo a função $\triangleright\triangleleft$ como adição, Bianchi (2004) prova que o erro na aproximação de *F* é limitado superiormente conforme a equação (3.17):

$$L_H(s_t) \le \xi \left[h_{\max}^{\beta} - h_{\min}^{\beta} \right], \forall s_t \in S \tag{3.17}$$

Bianchi (2004) prova que os valores máximo e mínimo que estimativa $\hat{Q}^*$ pode atingir são os valores mostrados nas equações (3.18) e (3.19):

$$\max \hat{Q}_t^*(s_t, a_t) = \frac{r_{\max}}{1-\gamma} \tag{3.18}$$

$$\min \hat{Q}_t^*(s_t, a_t) = \frac{r_{\min}}{1-\gamma} \tag{3.19}$$

Considerando $F_t(s_t, a_t) \equiv \hat{Q}_t^*(s_t, a_t)$, a partir da equação (3.15) é possível determinar os valores h_{min} e h_{max} conforme as equações (3.20) e (3.21), respectivamente:

$$h_{\min} = 0 \tag{3.20}$$

$$h_{\max} = \max_{a_t} \hat{Q}_t^*(s_t, a_t) - \hat{Q}_t^*(s_t, \pi^H(s_t)) + \eta \tag{3.21}$$

A heurística terá seu valor máximo quando $\max \hat{Q}_t^*(s_t, a_t)$ e $\min \hat{Q}_t^*(s_t, a_t)$ se encontrarem no mesmo estado s_t. Dessa forma, Bianchi (2004) define h_{max} em função dos

valores de recompensa r_{min} e r_{max}, conforme a equação (3.22) e redefine o erro de aproximação de acordo com a equação (3.23):

$$h_{\max} = \frac{r_{\max}}{1-\gamma} - \frac{r_{\min}}{1-\gamma} + \eta \tag{3.22}$$

$$L_H(s_t) = \xi\left[\frac{r_{\max} - r_{\min}}{1-\gamma} + \eta\right] \tag{3.23}$$

Até o momento, foi considerada uma função heurística H não-estacionária, cujos valores $H_t(s_t, a_t)$ são atualizados a cada iteração do algoritmo de AR. Entretanto, uma função heurística estacionária $H(s_t, a_t)$ também pode ser definida. De acordo com Bianchi (2004), para que o algoritmo de AR seja capaz de convergir, os valores da função heurística $H(s_t, a_t)$ devem ser anulados ou superados pelos valores da estimativa $\hat{Q}^*(s_t, a_t)$.

É considerado o pior caso aquele em que a política heurística sugere ações que resultem sempre nas menores recompensas r_{min}. Bianchi (2004) define o valor máximo de heurística h_{max} com base na equação (3.19), que determina o menor valor de recompensa acumulada possível, conforme a equação (3.24).

$$h_{\max} \leq \left|\frac{r_{\min}}{1-\gamma}\right| \tag{3.24}$$

Caso $|r_{\min}| > r_{\max}$, então h_{max} também deve obedecer à equação (3.25) para que o valor $H(s_t, a_t)$ seja, no mínimo, anulado pelo valor da estimativa $\hat{Q}^*(s_t, a_t)$.

$$h_{\max} \leq \frac{r_{\max}}{1-\gamma} \tag{3.25}$$

Bianchi (2004) propôs alguns algoritmos de AR combinados com a aceleração por heurísticas, dentre eles, uma variante do algoritmo Q-learning, utilizada nesse trabalho e descrita a seguir.

3.3.3 Q-learning Acelerado por Heurísticas - HAQL

Bianchi (2004) propôs uma extensão do algoritmo Q-learning, capaz de tirar proveito de conhecimento sobre o domínio para acelerar a convergência desse algoritmo, denominada "Q-learning Acelerado por Heurísticas" (*Heuristically Accelerated Q-learning* – HAQL).

Nesse algoritmo, a função F representa a estimativa de função valor ótimo $\hat{Q}^*$ e a função matemática $\triangleright\triangleleft$ representa a operação de adição, enquanto as variáveis ξ e β assumem

valor igual a 1. A estratégia de exploração é uma modificação da estratégia $\in$ – *Greedy* utilizada no Q-learning, que passa a considerar o valor da função heurística *H* na seleção das ações através de uma somatória simples, conforme mostra a equação (3.26):

$$\hat{\pi}_t^*(s_t) = \begin{cases} \arg\max_a \left[\hat{Q}^*{}_t(s_t, a_t) + H_t(s_t, a_t)\right] & se \;\; p > \varepsilon \\ a_{random} & se \;\; p \leq \varepsilon \end{cases} \tag{3.26}$$

A regra de atualização do algoritmo HAQL continua idêntica à regra de atualização do algoritmo Q-learning, assim como o cálculo do erro TD(0) δ^0. A Figura 3.4 apresenta o algoritmo HAQL, proposto por Bianchi (2004), muito similar ao algoritmo Q-learning da Figura 2.3.

Para todo estado *s* e ação *a*, inicialize *Q(s, a)* com zero
Observe o estado atual s_t
Repita infinitamente:
(1) Selecione uma ação a_t de acordo com a política heurística derivada de *Q*, (eq. (3.26))
(2) Receba a recompensa imediata r_t
(3) Observe o novo estado s_{t+1}
(4) Compute o erro TD(0) conforme a equação (2.17)
(5) Atualize a estimativa da função ação-valor Q^* conforme a equação (2.16)
(6) Atualize o valor da heurística $H_t(s_t, a_t)$, caso *H* seja não-estacionária
(7) $s_t \leftarrow s_{t+1}$

Figura 3.4 – Algoritmo HAQL

Em comparação com o algoritmo Q-learning, o algoritmo HAQL difere apenas na estratégia utilizada para a seleção das ações e na atualização dos valores da função heurística *H* a cada iteração, caso essa função seja não-estacionária.

Esse capítulo apresentou algumas das técnicas existentes que visam acelerar a convergência dos algoritmos de AR, assim como os algoritmos de AR que utilizam tais técnicas de aceleração, que foram utilizadas no desenvolvimento deste trabalho.

4 FUTEBOL DE ROBÔS

O domínio do Futebol de Robôs apresenta um ambiente real, dinâmico e não-determinístico. No entanto, é um ambiente de horizonte finito, onde as tomadas de decisão devem acontecer o quanto antes, caracterizando a importância em se utilizar técnicas que aceleram a convergência dos algoritmos de AR. Esse ambiente é completamente observável, mas com incerteza, visto que ruídos dos sensores são inerentes ao sistema.

4.1 Arquitetura de um sistema simulado – Categoria *Simurosot*

O objetivo da categoria de simulação *Simurosot* é permitir que pesquisadores desenvolvam algoritmos de controle e estratégias de sistemas multi-agentes sem a necessidade de um complexo e custoso sistema real de Futebol de Robôs. Os pesquisadores são encorajados a desenvolver e testar seus algoritmos nessa plataforma de simulação para, eventualmente, participarem da categoria *Mirosot*, na qual o sistema é exposto às condições de um ambiente real.

A categoria *Simurosot* oferece duas configurações de ambiente para simulação, o ambiente *Simurosot Large League*, que simula a categoria *Mirosot Large League* e o ambiente *Simurosot Middle League*, que simula a categoria *Mirosot Middle League*. Esse trabalho apresenta as características da categoria *Simurosot Middle League* sob a qual o sistema de AR foi desenvolvido. A Figura 4.1 apresenta a tela do programa servidor do simulador *Simurosot*.

Figura 4.1 – O simulador *Simurosot*

O programa servidor da categoria *Simurot Middle League* simula o campo de jogo, que mede de 220 x 180 cm, a bola (de golfe e de cor laranja) e os cinco robôs de cada time, em forma de cubos com tamanho máximo de 7,5 cm de aresta, conforme as regras da categoria *Mirosot Middle League*.

O servidor do simulador *Simurosot* disponibiliza as informações de posição e orientação dos robôs e da bola, assim como as extremidades do campo, as delimitações dos gols e das áreas, através de variáveis de programação. As informações contidas nessas variáveis são utilizadas pela estratégia de jogo e o algoritmo que controla a movimentação dos cinco robôs do time. A movimentação dos robôs acontece através da atribuição de valores às variáveis de velocidade dos motores de cada robô. Cada equipe desenvolve o próprio código de estratégia e algoritmo de controle, gerando um arquivo de biblioteca dinâmica (*dynamic-link library – DLL*) que é carregado pelo programa servidor. Com as estratégias carregadas no *Simurosot*, é iniciado um jogo, quem tem a duração de dois tempos de cinco minutos cada.

Segundo a FIRA (2007), a plataforma de simulação *Simurosot* foi desenvolvida para reproduzir, o mais fielmente possível, os aspectos e comportamentos físicos de um robô real da categoria *Mirosot*. Os robôs YSR_A, fabricados pela *Yujin Robotics* (2007), foram utilizados para a modelagem dinâmica e cinemática dos robôs virtuais. Os robôs YSR_A são dotados de duas rodas, cada qual com um motor para acioná-la, simetricamente posicionadas perpendicularmente ao eixo central do robô. Essa configuração física das rodas dos robôs YSR_A resulta em um modelo cinemático conhecido como diferencial.

Embora o simulador *Simurosot* utilize o modelo físico de um robô real, os motores dos robôs virtuais são idênticos e respondem igualmente e de forma linear às variações de velocidade. Não há incerteza inerente ao sistema, como por exemplo, atrito e escorregamento das rodas em relação ao campo, ruído e imprecisão de posição e orientação dos robôs e da bola. O campo é plano, uniforme e sem imperfeições.

No entanto, o simulador *Simurosot* reproduz um ambiente com os problemas típicos de controle de robôs móveis. É fornecido um algoritmo de controle simples para a movimentação e posicionamento dos robôs, que calcula as velocidades de cada roda conforme mostram as equações (4.1) e (4.2):

$$v_r = \left[v_c \left(\frac{1}{1 + e^{-3 \cdot d}} - 0.3 \right) \right] + K_a \Delta\theta \qquad (4.1)$$

$$v_l = \left[v_c \left(\frac{1}{1+e^{-3 \cdot d}} - 0.3 \right) \right] - K_a \Delta\theta \qquad (4.2)$$

Onde:

- v_r e v_l são, respectivamente, as velocidades das rodas direita e esquerda, cujos valores são limitados entre 0 e 127 pelo algoritmo de controle;
- v_c é um valor fixo que define a velocidade média de deslocamento do robô;
- d é a distância euclidiana entre a posição do robô e a posição desejada;
- K_a é uma constante que influencia diretamente a velocidade angular do robô;
- $\Delta\theta$ é a diferença entre o ângulo de direção do robô e o ângulo do robô em relação à posição desejada.

Apesar de apresentar oscilação no posicionamento dos robôs, as equações de controle (4.1) e (4.2) oferecem as condições básicas para pesquisadores que queiram focar os estudos apenas na estratégia do time, sem que seja necessário desenvolver um algoritmo de controle para a movimentação e posicionamento dos robôs.

4.2 Arquitetura de um sistema real – Categoria *Mirosot*

A categoria *Mirosot* propõe aos pesquisadores o desafio de se desenvolver uma arquitetura completa de um time de Futebol de Robôs para atuar em um ambiente real. Essa categoria possui três configurações de ambiente, *Mirosot Large League*, *Mirosot Middle League*, e *Mirosot Small League*. As categorias diferem apenas nas dimensões do campo e no número de robôs por time. Esse trabalho foi desenvolvido sob a arquitetura e as regras da categoria *Mirosot Small League*.

A arquitetura proposta pela FIRA para a categoria *Mirosot Small League* pode ser vista na Figura 4.2. O campo mede 150 x 130 cm e, como já descrito na seção anterior, os robôs têm o tamanho limitado por um cubo com arestas de tamanho máximo de 7,5 cm.

O sistema de cada time é composto por uma câmera posicionada acima do campo para capturar a imagem, um sistema de visão computacional para o reconhecimento dos objetos, um sistema de estratégia do time, um sistema de navegação e controle e três robôs, que recebem informações provindas do computador, por radiofreqüência, através de um protocolo de comunicação.

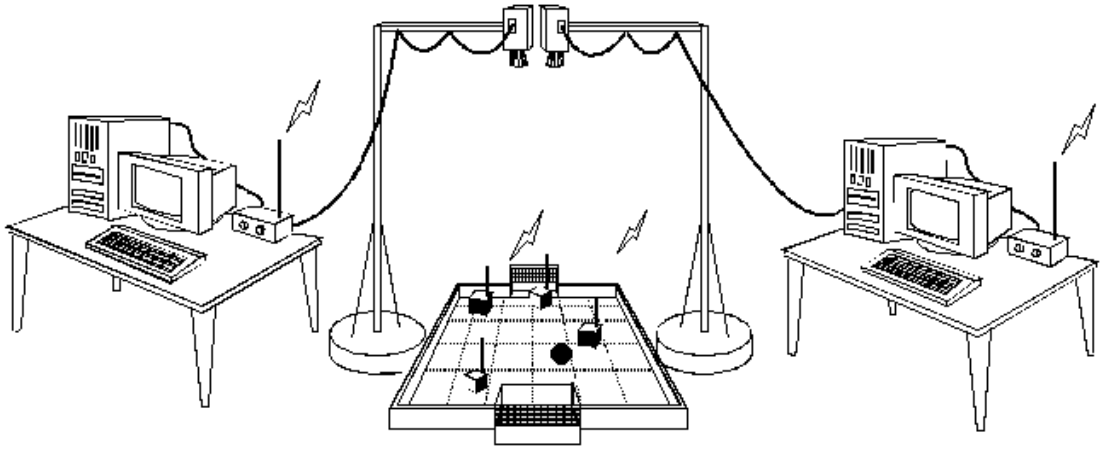

Figura 4.2 – Arquitetura proposta pela FIRA para a categoria *Mirosot Small League*

A seguir, são apresentados alguns módulos desenvolvidos para resolver os problemas de visão computacional, estratégia, navegação e controle, além de alguns trabalhos que descrevem características de projeto de robôs para a categoria *Mirosot Small League*.

4.2.1 Sistemas de visão computacional

O sistema de visão computacional de um time de futebol de robôs da categoria *Mirosot* deve ser capaz de, a partir da imagem do campo inteiro capturada pela câmera, reconhecer todos os objetos pertencentes ao domínio do Futebol de Robôs. Esses objetos são: a bola, que na imagem aparece representada pela figura geométrica de um círculo preenchido pela cor laranja, e os robôs, do próprio time e do time adversário. Cada robô deve ser dotado de uma etiqueta colorida para que possa ser feita a distinção entre os times. De acordo com as regras da categoria *Mirosot*, cada robô deve possuir uma área mínima de 12,25 cm^2, independente da forma geométrica, preenchida por uma cor, azul ou amarela, para representar o time. A Figura 4.3 exemplifica um modelo de etiqueta que pode ser utilizado para a distinção dos times da categoria *Mirosot*.

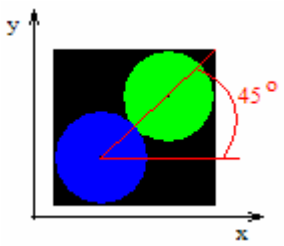

Figura 4.3 – Etiqueta de um robô da categoria *Mirosot*

Pode-se utilizar uma cor secundária para diferenciar robôs do mesmo time, desde que essa cor secundária não seja a cor que representa o time adversário, a cor da bola, a cor branca ou a cor preta.

Dadas as restrições impostas pela regra da categoria *Mirosot*, para participar das competições cada equipe desenvolve seu próprio algoritmo para resolver o problema de reconhecimento dos objetos. Esse algoritmo deve ser rápido o suficiente para não comprometer o desempenho do sistema em tempo real e tolerante a ruídos e variação de intensidade luminosa.

Sistemas de visão computacional para o Futebol de Robôs podem considerar apenas informações de cor. Bianchi e Reali-Costa (2000) propuseram um algoritmo que percorre a imagem e, ao detectar um pixel de cor especificada, traça um segmento de reta interno para os pixels de mesma cor no sentido horizontal. A partir do ponto médio desse segmento horizontal é traçado, novamente, um segmento interno de reta para os pixels dessa mesma cor especificada, mas no sentido vertical. O ponto médio desse segmento vertical é, então, considerado o centro do objeto da cor especificada. Esse algoritmo foi desenvolvido visando o reconhecimento de objetos em forma de círculo.

A proposta de Bianchi e Reali-Costa (2000) foi estendida no trabalho de Martins, Tonidandel e Bianchi (2006a). A varredura da imagem é feita por amostragem, onde, a cada linha, são analisados os pixels distantes em um número configurável um do outro, ao invés de analisar todos os pixels de uma linha. Essa amostragem diminui consideravelmente o tempo de processamento. Também é feito um processamento em cruz onde os segmentos de reta horizontal e vertical são traçados várias vezes, de acordo com um número de vezes configurável. Dessa forma, não apenas objetos representados por círculos podem ter seus centros determinados, mas todo e qualquer objeto, independente da forma geométrica pela qual ele é representado na imagem.

O trabalho de Grittani, Gallinelli e Ramírez (2000) também se baseia apenas em informações de cor, assim como o trabalho de Weiss e Hildebrand (2004), que utiliza essas informações de cor para reduzir a quantidade de informações contida em cada quadro de imagem através de um filtro de pontos de relevância.

Outros sistemas consideram a forma geométrica dos objetos para detectá-los na imagem, técnica geralmente utilizada em sistemas de visão local. O trabalho de Gonner, Rous e Kraiss (2000), por exemplo, detecta a bola através de sua forma geométrica projetada na imagem, um círculo, mas ainda utiliza as informações de cor para reconhecer os robôs.

Independente da técnica utilizada para resolver o problema, o sistema de visão computacional deve ser capaz de determinar posição e orientação dos robôs e posição da bola com a máxima precisão e mínimo tempo de processamento possível, pois o sucesso do sistema de estratégia nas tomadas de decisão depende diretamente das informações vindas do sistema de visão computacional.

4.2.2 Sistemas de estratégia

Responsável pelo comportamento do time, o sistema de estratégia determina as ações de movimentação dos robôs de um time baseando-se na observação de posicionamento atual dos robôs e da bola, fornecido pelo sistema de visão computacional. O sistema de estratégia deve ser capaz de tomar as decisões por todos os robôs de um time em poucos milissegundos para que o sistema possa funcionar em tempo real.

Diversas técnicas de IA são aplicadas no desenvolvimento de um sistema de estratégia para o Futebol de Robôs, técnicas as quais determinam a complexidade da estratégia. Essas técnicas podem ser desde tomadas de decisões simples, até técnicas de aprendizado de máquina, onde o sistema aprende a interagir com o ambiente, seja por experimentação, por observação ou ainda recebendo exemplos de como atuar nesse ambiente.

As abordagens mais comuns na literatura são sistemas de estratégia integrados a sistemas de navegação e controle, responsáveis pela determinação das trajetórias dos robôs e correção do erro na execução dessas trajetórias.

Faria, Teizen e Romero (2004) desenvolveram uma estratégia baseada em campos potenciais para determinar as ações de movimentação dos robôs utilizando o ambiente de simulação *SimuroSot*. Dessa forma os robôs são capazes de alcançar um ponto final desejado, que geram um campo potencial de atração, desviando de obstáculos, que geram campos potencias de repulsão.

Han et al. (2002) apresentam uma estratégia hierárquica de quatro camadas. A camada de mais baixo nível está dividida em comportamentos, programados manualmente com funções específicas de atacante, defensor, goleiro e uma função que representa o comportamento de deslocar-se para uma área livre para receber um passe. A camada intermediária acima da camada de mais baixo nível é responsável por selecionar o comportamento de cada robô, considerando algumas percepções do estado atual, como posição atual do robô e da bola, por exemplo. A camada intermediária mais acima, por sua

vez, modifica a prioridade de cada robô para servir de referência na próxima escolha dos comportamentos dos robôs. Por último, a camada de mais alto nível fornece informações de possíveis problemas mecânicos e situacionais para a camada de mudança de prioridade. A estratégia ainda conta com quatro diferentes formações de posicionamento dos robôs em relação ao campo.

A abordagem de Egly, Novak e Weber (2004) utiliza lógica *fuzzy* como a base de um sistema hierárquico de três camadas para a tomada de decisões. Dessa maneira é possível dividir o objetivo global, vencer o jogo, em tarefas mais simples. A camada mais acima, chamada *Strategy*, define o comportamento principal (ofensivo ou defensivo) para o time, visando atingir o objetivo global. A camada intermediária, chamada *Task Distribution*, atribui o papel de cada robô baseada no comportamento principal definido pela camada *Strategy*. A camada mais abaixo, chamada *Action*, depende do papel de cada robô. As ações possíveis representam comportamentos como, por exemplo, "Chutar no gol". Esse sistema funciona como uma máquina de estados onde as transições de estado são baseadas nas regras do sistema *fuzzy*.

Whiteson et al. (2005) apresentam uma comparação de algumas técnicas para evolução de robôs que jogam futebol. Uma das técnicas, por exemplo, é composto por uma árvore de decisão onde cada nó representa uma rede neural responsável por uma tarefa, como passe ou interceptação, por exemplo. Outra técnica comparada por Whiteson et al. (2005) utiliza o conceito de Aprendizado em Camadas introduzido por Stone (1998).

As estratégias descritas, invariavelmente, utilizam abstração de informações de observação do ambiente e de ações de comportamento, facilitando a inserção de conhecimento sobre o domínio. Tais abstrações também tornam o sistema mais rápido e eficiente, pois o sistema de tomada de decisão utiliza informações de mais alto nível sobre o ambiente e tem o número de comportamentos possíveis reduzido.

Então, os sistemas de estratégia visam selecionar as melhores movimentações para os robôs de um time. Entretanto, o sucesso dessa tomada de decisões depende tanto da precisão das informações do sistema de visão computacional, como de um sistema de navegação e controle.

4.2.3 Sistemas de navegação e controle

Sistemas de navegação e controle devem ser capazes de guiar os robôs por caminhos que resultem em posição e orientação finais definidas pelo sistema de estratégia, com o mínimo de erro, além de evitar colisões com os robôs do time adversário.

Novak e Seyr (2004) propuseram um sistema de planejamento de trajetória baseado no modelo cinemático diferencial de um robô da categoria *Mirosot*. O sistema é dividido em camadas. A camada de alto nível calcula a trajetória, a qual é composta por pontos de intersecção. São calculadas, então, as velocidades linear e angular do robô. A camada de baixo nível controla a velocidade dos motores, garantindo que a trajetória desejada seja seguida. Dois algoritmos são utilizados para determinar a trajetória. O primeiro algoritmo gera uma trajetória até determinada posição sem preocupar-se com a orientação do robô ao atingir a posição desejada. O segundo algoritmo utiliza o primeiro algoritmo para calcular um ponto de intersecção e, em seguida, traça um arco circular até a posição final desejada. Dessa forma pode-se alcançar a posição final com a orientação desejada. Segundo Novak e Seyr (2004), o sistema tem um custo computacional baixo e é simples de ser implementado.

O trabalho de Kim, Park e Kim (2001) propõe um método de navegação que utiliza funções não-lineares de segunda ordem. O ajuste do raio da circunferência que o robô deve seguir e a direção do obstáculo a ser desviado resulta em um sistema que evita colisões com obstáculos, mesmo móveis, e alcança a posição final desejada.

Seyr e Jakubek (2005) abordam o problema de seguir uma trajetória definida com um sistema de controle preditivo não-linear baseado no algoritmo de Gauss-Newton, combinado com um sistema de controle linear modelado em espaço de estados para o modelo cinemático diferencial, demonstrando a importância de um sistema de controle capaz de considerar as estimativas de posição futuras. Em Seyr, Jakubek e Novak (2005) o mesmo problema de controle foi abordado utilizando-se redes neurais para o sistema de controle preditivo não-linear.

4.2.4 Robôs da categoria *Mirosot*

Sistemas de navegação e controle são geralmente desenvolvidos considerando o modelo cinemático e/ou dinâmico dos robôs utilizados por cada equipe. O modelo cinemático mais utilizado pelas equipes é o modelo diferencial, considerando-se a restrição de tamanho imposta pela regra da categoria *Mirosot* e dada a sua facilidade de implementação, cujo

modelo matemático facilita o desenvolvimento do algoritmo de controle. Esse modelo cinemático define um robô simétrico, com duas rodas, cada qual com seu motor de acionamento, capaz de atacar a bola utilizando tanto sua parte frontal quanto a parte traseira.

Características mecânicas desejáveis para robôs voltados às competições de futebol são baixo peso e baixo centro de gravidade, além de alto torque e bom atrito com a superfície para evitar derrapagens e ter aceleração e velocidade suficientes para acompanhar o dinamismo de uma partida.

A parte eletrônica de um robô deve ser capaz de receber dados do computador via radiofreqüência, interpretar esses dados, acionar os motores e controlar a velocidade dos mesmos. A recepção dos dados deve ser rápida e robusta o suficiente para não tornar-se o gargalo de tempo do sistema, além de ser imune a ruídos. Todo o circuito deve ser projetado minimizando o consumo total de energia, visto que a bateria deve fornecer a energia consumida pela parte eletrônica e respeitar as dimensões máximas impostas pela regra.

Novak (2004) descreve um robô, denominado *Roby-Go*, desenvolvido para a categoria *Mirosot*. São citadas algumas características desejáveis importantes, como um único estágio de engrenagens de redução e motores com *encoders* de alta resolução. O robô é modular e tem a parte eletrônica responsável pela etapa de potência dos motores separada da unidade de controle para possibilitar modificações futuras em sua arquitetura. A unidade de controle é composta por um microcontrolador operando a 20 MHz e recebe os dados do computador através de um módulo de radiofreqüência que opera em uma freqüência de 433 MHz. Essa unidade de controle gera sinais PWM (*Pulse Width Modulation*) para o controle de rotação dos motores e recebe as informações dos *encoders* internos dos motores para efetuar o controle de velocidade, cujas equações da malha de controle são desenvolvidas com base no modelo cinemático diferencial do robô. A integração dos módulos é feita através de uma rede CAN (*Controller Area Network*).

O robô *Tinyphoon*, desenvolvido por Novak e Mahlknecht (2005), é baseado na arquitetura do robô *Roby-Go*, utilizando, inclusive, a mesma malha de controle dos motores. A principal diferença se encontra na parte eletrônica, a qual possui um módulo de visão local, composto por uma câmera de vídeo e um DSP (*Digital Signal Processor*) para tornar o robô completamente autônomo, sem depender de um computador externo para enviar informações. Além do módulo de visão local, outras melhorias foram feitas, como a comunicação de dados

com o computador, que é realizada a uma freqüência de 2,4 GHz e a uma taxa de 1 Mbps. A unidade de controle utiliza um microcontrolador operando a 40 MHz.

Os robôs desenvolvidos para partidas de futebol são os atuadores de um sistema autônomo de controle com objetivo definido (vencer uma partida de futebol). O projeto desses atuadores pode envolver as mais variadas tecnologias e utilizar qualquer solução de *hardware* e *software*, desde que sejam respeitadas as regras impostas pela categoria *Mirosot*.

Esse capítulo descreveu o domínio do Futebol de Robôs e apresentou uma breve revisão bibliográfica sobre algumas abordagens para solucionar os problemas de visão computacional, estratégia, navegação, controle e projeto de robôs móveis para uma arquitetura real de Futebol de Robôs da categoria *Mirosot*. Com base nessas informações, o próximo capítulo descreve os sistemas de visão computacional, navegação e robôs, desenvolvidos para que os algoritmos de AR pudessem ser aplicados no domínio do Futebol de Robôs real.

5 O TIME DE FUTEBOL DE ROBÔS DESENVOLVIDO

Para que o objetivo desse trabalho fosse atingido, foi necessário o desenvolvimento de um time completo de Futebol de Robôs da categoria *Mirosot*. A seguir, são descritos os sistemas de visão computacional, navegação, controle, os robôs móveis e o protocolo de comunicação entre robôs e computador.

5.1 O sistema de visão computacional desenvolvido

Feitas as considerações iniciais sobre sistemas de visão computacional para a categoria *Mirosot* no capítulo anterior, o sistema de visão computacional desenvolvido, descrito em Martins, Tonidandel e Bianchi (2006b), segue as restrições impostas pela regra da categoria *Mirosot* e resolve os problemas de detecção e reconhecimento dos objetos.

5.1.1 A Transformada de Hough

A Transformada de Hough (TH), segundo McLaughLin e Alder (1998), é uma técnica de segmentação de imagens muito conhecida, utilizada para a detecção de objetos a partir do ajuste de modelos de uma classe de objetos. Essa técnica exige que uma classe de objetos seja determinada, a qual deve ser capaz de descrever todas as possíveis combinações de valores dos parâmetros que definem o objeto referido. Entretanto, imagens capturadas em tempo real, como as utilizadas no domínio do Futebol de Robôs, apresentam ruídos, os quais não são modelados, nem tolerados pela parametrização da classe de determinado objeto. A parametrização de uma classe de objetos define a forma desse objeto, portanto, variações de cor na imagem, ou até mesmo nos objetos não afetam o desempenho e a eficiência da TH. As informações relevantes para a TH, para que seja possível detectar objetos em uma imagem, são as bordas desses objetos encontradas nessa imagem. Logo, a TH exige um pré-processamento da imagem, o qual deve ser capaz de extrair as bordas dos objetos contidos nessa imagem.

Para esse trabalho o círculo foi escolhido como a forma geométrica a ser parametrizada. A motivação em reconhecer círculos se dá pelo fato que são uma estrutura geométrica muito comum no domínio do Futebol de Robôs, onde todos os objetos pertencentes a esse domínio podem ser representados por círculos. Por exemplo, a bola é uma esfera, cuja projeção na imagem capturada será representada por um círculo. Para a distinção

entre times são utilizadas etiquetas com as cores pré-determinadas pela regra, como descrito anteriormente. O modelo de etiqueta utilizado nesse trabalho segue o padrão mostrado na Figura 4.3, onde os círculos possuem o mesmo diâmetro da bola utilizada em um jogo de futebol de robôs, dispostos a 45 graus, tendo como referência o eixo *x*, que é a direção da frente do robô.

Círculos são parametrizados pela tripla (x_c, y_c, r), a qual define um conjunto de pontos eqüidistantes em r do ponto central representado pela coordenada cartesiana (x_c, y_c).

Essa relação de distância é expressa pelo Teorema de Pitágoras, em coordenadas cartesianas, conforme a equação (5.1):

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \tag{5.1}$$

Entretanto, segundo Hearn e Baker (1997), esse não é o melhor método para parametrizar um círculo. Pode-se parametrizar um círculo utilizando o plano de coordenadas polares, resultando nas equações (5.2) e (5.3):

$$x = x_c + r\cos\theta \tag{5.2}$$

$$y = y_c + r\ sen\theta \tag{5.3}$$

Dessa maneira, com valores conhecidos para a tripla (x_c, y_c, r), variando-se o ângulo θ por todo intervalo de 0 a 360 graus, pode-se traçar a circunferência completa.

Mas o objetivo da TH é justamente o contrário do que as equações (5.2) e (5.3) mostram, é determinar os parâmetros da tripla (x_c, y_c, r) a partir de um círculo, pois a única informação provinda da imagem são os pontos de borda, que podem ou não fazer parte da borda desse círculo. O espaço de parâmetros, ou histograma da imagem, também chamado de espaço de Hough, é tridimensional. A necessidade de efetuar uma busca pelos valores dos parâmetros da tripla (x_c, y_c, r) em um espaço tridimensional torna a técnica da TH computacionalmente complexa. Para resolver esse problema e aumentar a robustez do método original, várias técnicas foram propostas, como TH com segmentação e análise de regiões de interesse, TH Probabilística e TH com busca exaustiva combinada com regras de parada, de acordo com Stricker e Leonardis (1995).

Como o foco da análise do problema é a aplicação da TH no domínio do Futebol de Robôs, onde os objetos estão constantemente se movimentando, mas apenas em duas dimensões, não existirá variação de profundidade dos objetos na imagem, o que possibilita

determinar um valor fixo para o raio *r*. Logo, o espaço de parâmetros torna-se bidimensional e computacionalmente viável para uma aplicação em tempo real, pois o espaço de busca torna-se bidimensional. Esse espaço de parâmetros, com raio *r* fixo é, então, representado pela tupla (x_c, y_c).

Detectar círculos de raio fixo em uma imagem que contém apenas pontos de borda de coordenadas (x, y) utilizando a TH consiste em determinar quais desses pontos pertencem à borda do círculo com centro em (x_c, y_c) e raio *r*. O algoritmo da TH determina para cada ponto de borda (x, y) da imagem um conjunto de possíveis centros (x_c, y_c) no espaço de Hough, conjunto o qual será definido iterativamente pela variação do ângulo θ, modificando as equações (5.2) e (5.3), resultando nas equações (5.4) e (5.5):

$$x_c = x - r\cos\theta \tag{5.4}$$

$$y_c = y - r\,sen\theta \tag{5.5}$$

A Figura 5.1 demonstra a execução do algoritmo descrito por Pistori, Pistori e Costa (2005) para três pontos da borda de um círculo da imagem à esquerda, enquanto o respectivo espaço de Hough gerado é mostrado à direita. Os três círculos de raio *r* desenhados no espaço de Hough, a partir de três pontos (x, y) de borda do círculo de centro (x_c, y_c) da imagem, se intersectam em apenas um ponto, justamente o ponto central de mesmas coordenadas (x_c, y_c) do círculo presente na imagem.

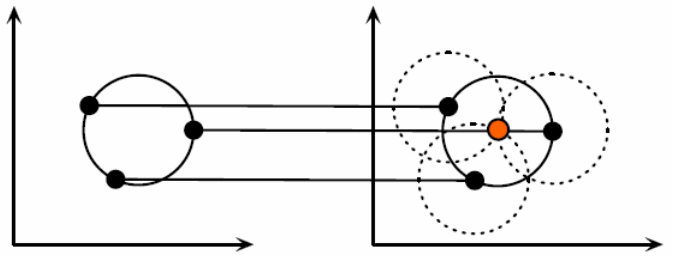

Figura 5.1 – Exemplo de geração de pontos no espaço de Hough

Cada ponto de borda da imagem gera um círculo no espaço de Hough e cada ponto de borda desse círculo no espaço de Hough recebe um voto, o que significa que esse ponto tem seu valor incrementado. Quanto maior o número de votos recebidos por um ponto, maior a probabilidade de esse ponto representar o centro de um círculo. Esses pontos de maior probabilidade, ou com maior número de votos, são os máximos relativos do espaço de Hough.

O sistema de visão computacional implementado possui basicamente sete etapas de processamento, que são elas: aquisição da imagem, subtração de fundo, conversão de cores

para escala de cinza, aplicação do filtro de bordas proposto por Canny (1986), geração do espaço de Hough, determinação de pontos com alta probabilidade de serem centros de círculos na imagem e reconhecimento dos objetos (robôs e bola).

5.1.2 Sistema de aquisição de imagens

O sistema de aquisição de imagens consiste em uma câmera analógica com saída de vídeo composto e uma placa de aquisição de vídeo capaz de adquirir até 30 quadros por segundo a uma resolução máxima de 640 x 480 pontos e profundidade de 24 bits de cores.

5.1.3 Subtração de fundo

Como citado anteriormente, apenas as bordas da imagem são relevantes para TH. Cada ponto de borda representa uma iteração do algoritmo da TH. Para otimizar o tempo desse algoritmo utilizou-se um método simples de subtração de fundo, descrito por Piccardi (2004), que calcula a diferença entre a imagem capturada e uma imagem do fundo, sem os objetos móveis. A imagem de fundo é atualizada ao longo do tempo, segundo o método conhecido como *Running Average*. Essa atualização acontece de acordo com a equação (5.6):

$$B_{i+1} = \alpha F_i + (1-\alpha) B_i \qquad (5.6)$$

A imagem de fundo é representada por B, enquanto a imagem capturada é representada por F. Uma taxa de aprendizado α é utilizada para determinar o quão rápido objetos estáticos se tornam parte da imagem de fundo.

Apesar de simples, esse método é eficaz para essa aplicação, porque o fundo não sofre alterações significativas. O resultado é uma imagem que contém apenas os objetos móveis, que irá resultar apenas em bordas relevantes, as bordas dos objetos, para a TH. O resultado da subtração de fundo pode ser visto na figura 5.2.

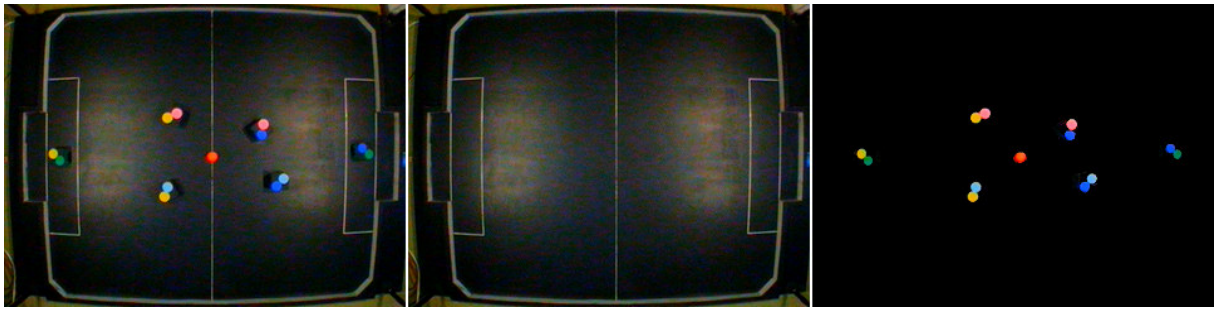

Figura 5.2 – Exemplo de subtração de fundo

5.1.4 Conversão de cores para escala de cinza

Imagens coloridas demandam um maior tempo de processamento, pelo fato de possuírem três canais de informação de cor. Como a TH não necessita de informações de cor, para otimizar o tempo de processamento do sistema a imagem é convertida para escala de cinza. A imagem resultante contém apenas um canal de informação referente à variação de iluminação, a qual é suficiente para a determinação das bordas na imagem.

5.1.5 O filtro de bordas Canny

Existem diversas técnicas capazes de extrair informações de bordas em uma imagem, como as descritas por Forsyth e Ponce (2003, p.165) e por Ziou e Tabbone (1998). O presente trabalho utiliza uma técnica muito conhecida para detecção de bordas, o filtro Canny, proposto por Canny (1986), para a detecção de bordas da imagem, resultando em uma imagem binária. Esse filtro foi escolhido por se tratar de um filtro ótimo linear.

A Figura 5.3 demonstra o resultado da detecção de bordas com o filtro Canny a partir de uma imagem resultante da subtração do fundo, convertida para escala de cinza.

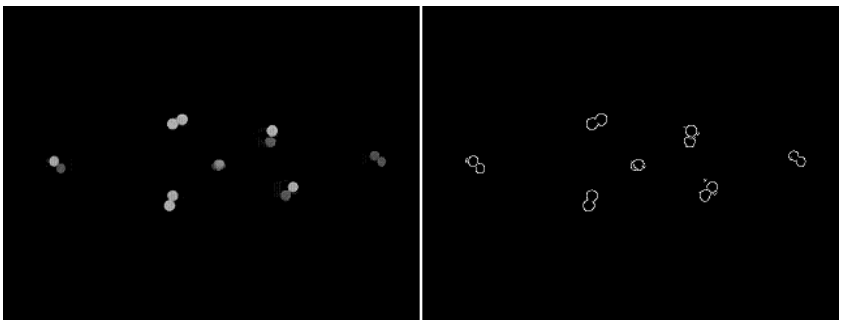

Figura 5.3 – Resultado da aplicação do filtro Canny

5.1.6 Geração do espaço de Hough

A partir da imagem binária resultante do filtro Canny pode-se gerar o espaço de Hough. A geração do espaço de Hough a partir do algoritmo descrito por Pistori, Pistori e Costa (2005) é eficaz, mas não é eficiente. Apesar de gerar o espaço de Hough corretamente, esse algoritmo efetua diversas iterações redundantes para gerar os pontos de possíveis centros de círculos. Essa redundância acontece no laço referente à variação do ângulo θ. O algoritmo gera pontos com precisão de várias casas decimais, além de gerar 360 pontos de possíveis centros para cada ponto de borda encontrado. Entretanto, as imagens digitais são quadriculares (conhecidas como *raster images*), ou seja, sua precisão de pontos (resolução) é

representada por números inteiros, onde cada ponto é conhecido como pixel. Logo, a precisão de casas decimais é algo irrelevante para a determinação de coordenadas de pontos. Ao desconsiderar a precisão de casas decimais, nota-se que determinados intervalos de valores de θ irão resultar em valores iguais de pontos de coordenadas (x, y), ou seja, um mesmo pixel, caracterizando a redundância. Para resolver o problema de redundância observado, utilizou-se um algoritmo de desenho de círculos proposto por Bresenham (1965), também descrito em Hearn e Baker (1997). Esse algoritmo tira vantagem da simetria de pontos em um círculo.

Basicamente o algoritmo determina os pontos de apenas um octante e, por simetria, em tempo constante, determina os pontos dos outros sete octantes, não havendo repetição de pontos já desenhados. Dessa maneira é possível ter um algoritmo de geração do espaço de Hough funcional, para que possa ser utilizado em aplicações em tempo real, otimizando o tempo de processamento necessário.

Com o espaço de Hough gerado, detectar possíveis centros (x_c, y_c) de círculos de raio fixo r em uma imagem se resume a encontrar os pontos de máximos relativos, os mais votados, nesse espaço de Hough. Segundo McLaughLin e Alder (1998), existem diversas técnicas que foram desenvolvidas com o propósito de resolver o problema de interpretação do espaço de Hough.

Essa etapa é a que apresenta maiores problemas na detecção de círculos, pois como todo e qualquer ponto de borda gera um conjunto de pontos (círculo) que são votados no espaço de Hough, pode haver votos falsos, ocasionando em máximos relativos falsos. Um ponto recebe um voto quando alguma circunferência gerada por um ponto de borda passa por ele. Podem ocorrer, ainda, máximos relativos cuja proximidade de suas coordenadas seja de poucos pixels. Pelo fato de se trabalhar com imagens quadriculares, essa proximidade pode representar pontos com alta probabilidade de serem centros de um mesmo círculo. Outro problema é determinar dinamicamente o número de votos mínimo para que se possa considerar tal ponto como sendo o centro de um círculo.

Ao mesmo tempo em que o espaço de Hough é gerado, ou seja, a cada iteração do algoritmo, é verificado se o ponto votado ultrapassou um determinado número de votos mínimo, definido previamente. Se tal ponto ultrapassou esse limiar, ele é armazenado, inserido em um vetor, uma única vez. Ao final da geração do espaço de Hough, o parâmetro de número de votos de cada ponto armazenado no vetor é atualizado, pois esse ponto pode ter recebido mais votos após ter ultrapassado o limiar de número de votos mínimo.

5.1.7 Determinação de círculos a partir do espaço de Hough

Para garantir que todos os pontos que representam centros reais de círculos na imagem sejam inseridos no vetor, é definido um baixo número de votos mínimo. A aplicação da subtração de fundo resulta em uma saída do filtro Canny com poucas bordas além daquelas dos objetos que se pretende reconhecer. Mas, pelo fato de se ter um baixo número de votos mínimo, haverá pontos referentes a falsos máximos relativos presentes nesse vetor. O espaço de Hough gerado para uma imagem de teste pode ser obesrvado na Figura 5.4, na qual pode ser verificada, também, a importância da subtração de fundo para evitar que falsos máximos relativos sejam gerados.

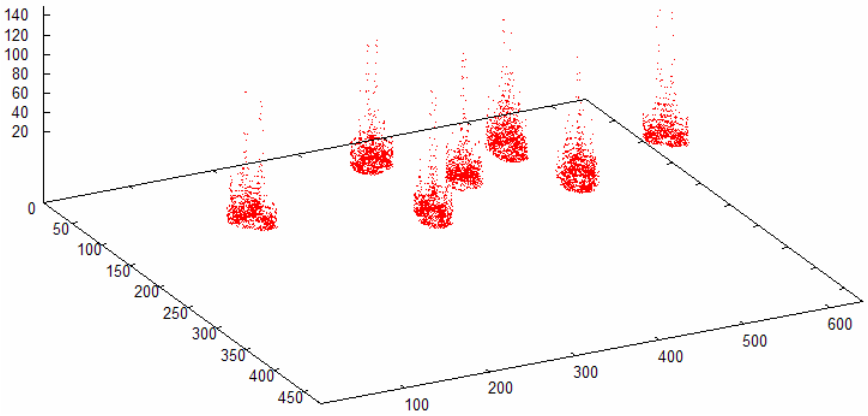

Figura 5.4 – O espaço de Hough

O primeiro passo para separar os falsos máximos relativos de pontos de centros de círculo reais é a ordenação dos pontos pelo número de votos recebidos com o algoritmo *quicksort*. Após a ordenação, o ponto máximo global, presente na primeira posição do vetor, é considerado um centro de círculo. O ponto que possui o segundo valor máximo é comparado com o ponto anterior. Para tal comparação é calculada a distância euclidiana entre os pontos.

Um outro parâmetro manual determina se pode haver sobreposição de círculos ou não. Caso possa haver sobreposição, então a distância euclidiana deve ser maior ou igual ao valor do raio r. Caso a sobreposição não seja uma característica desejável, então a distância euclidiana deve ser maior ou igual a $2r$.

Caso essas restrições sejam satisfeitas, encontrou-se um novo ponto de centro de círculo. O algoritmo continua suas iterações até que seja encontrado o número máximo de círculos determinado, ou caso todos os pontos do vetor sejam verificados. Dessa forma, é feita uma busca exaustiva apenas pelos pontos que excederam o número de votos mínimo, tendo

como critério de parada o número máximo de círculos a serem encontrados, resultando em um vetor com pontos distantes o suficiente para poderem representar círculos distintos.

Cada ponto que representa um centro de círculo é inserido em um novo vetor, o vetor de centros. Apesar de aumentar a certeza sobre a possibilidade de um ponto desse vetor de centros representar um centro de círculo real na imagem, um número de votos mínimo muito baixo ainda pode resultar em falsos máximos relativos no vetor. O importante é ter um número de votos mínimo capaz de garantir que todo ponto que representa um círculo real esteja presente no vetor de centros e que cada círculo seja representado por um e somente um ponto presente nesse vetor.

5.1.8 Reconhecimento dos objetos

O vetor de centros é utilizado para reconhecer os objetos. Essa etapa é a única que considera informações de cor e iluminação da imagem, pois para reconhecer os objetos com sucesso é necessário distingui-los, o que pode ser feito somente a partir das cores principais, pré-determinadas pela regra, diferentes para cada time, assim como as cores secundárias, para diferenciar os robôs de um mesmo time. Para a calibração das cores é utilizado o sistema já existente da equipe de Futebol de Robôs do Centro Universitário da FEI, proposto por Bianchi et al. (2004).

O problema pode ser descrito como uma árvore, onde a raiz é um ponto do vetor de centros de cor primária, que representa um time, tendo como nós filhos os outros pontos do vetor de centros. Um nó, ou seja, um ponto do vetor de centros é filho da raiz caso a distância entre esse ponto e o ponto da raiz seja pequena o suficiente para considerar que não há espaço entre as duas circunferências.

Ao percorrer o vetor de centros, é determinada a primeira raiz para se resolver o problema, verificando se esse centro é de um círculo cuja cor seja primária e também se esse centro já pertence a algum objeto reconhecido. Com a raiz determinada, a árvore é construída, determinando seus filhos.

Depois de construída a árvore, é feita uma busca em largura (visto que a árvore tem apenas um nível) para verificar se existem nós filhos válidos, ou seja, com alguma cor secundária, que representa um dos robôs do time. A restrição do problema está no fato de que cada raiz pode ter apenas um nó filho válido. Caso a restrição seja satisfeita, é reconhecido o robô de cor primária representada pela raiz e cor secundária representada por esse nó filho e

esses pontos passam a representar um objeto já reconhecido. A Figura 5.5(a) exemplifica uma árvore com o ponto presente na posição 0 do vetor e seus nós filhos. Essa árvore irá resultar no reconhecimento de um robô com a cor primária representada pela raiz e cor secundária do nó filho cuja posição no vetor é cinco, pois os outros nós filhos não representam nenhuma outra cor secundária.

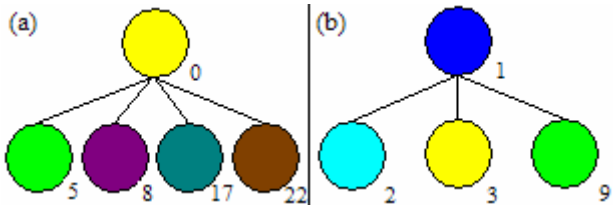

Figura 5.5 – Exemplo de árvores com raízes de cor primária e nós filhos (círculos próximos)

Tendo as informações de centro de cada círculo da etiqueta e a disposição dos círculos conhecida, o algoritmo determina a posição global (*x, y*) do robô na imagem e seu ângulo de direção em relação ao eixo *x*. Entretanto, pode acontecer de robôs estarem próximos, ou até mesmo encostados, fazendo com que uma raiz tenha mais de um nó filho com alguma cor secundária, como a árvore da Figura 5.5(b). Nesse caso nenhum robô é reconhecido e os pontos do vetor de centros ainda não representam objetos reconhecidos. O algoritmo continua a percorrer o vetor de centros até encontrar outra raiz. Ao determinar uma nova raiz, outra árvore é construída. Não há critério de parada do algoritmo, ele percorre o vetor de centros por três vezes construindo árvores com raízes e nós filhos que não representam um objeto já reconhecido.

O motivo pelo qual o algoritmo é executado por três vezes é para resolver os problemas em que alguma raiz possua mais de um nó filho de cor secundária. Pela disposição dos círculos nas etiquetas dos robôs, observou-se que o pior caso seria uma raiz com três nós filhos válidos, como pode ser observado na Figura 5.6.

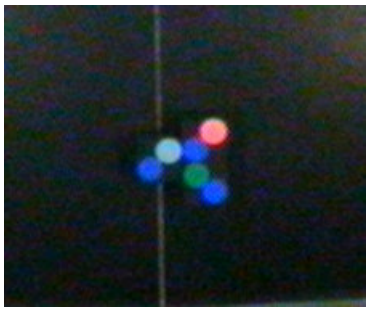

Figura 5.6 – Raiz com três nós filhos válidos

Na primeira iteração, raízes com mais de um nó filho não são reconhecidas como nenhum objeto. Na segunda iteração, as raízes que tinham dois nós filhos válidos são

resolvidas, pois uma nova árvore é construída e um dos nós filhos da iteração anterior não fará mais parte da nova árvore por ter sido resolvido na primeira iteração e já representar um objeto. Por fim, a terceira iteração do algoritmo resolve o problema de uma raiz com três nós filhos válidos.

O reconhecimento da bola é uma exceção. Como a bola é representada por um círculo de cor laranja e essa cor não pode ser utilizada em nenhum outro objeto, todo e qualquer círculo cuja cor seja laranja é considerado uma bola. Não há necessidade de se construir uma árvore de busca de soluções para reconhecer bolas.

Poderia ser implementado um critério de parada quando todos os objetos fossem detectados, mas é desejável que o sistema seja capaz de reconhecer mais de uma bola, assim como um número maior ou menor de robôs presentes no campo.

Outro critério de parada possível seria quando todos os pontos do vetor fossem reconhecidos como objetos, diminuindo, em algumas situações, o número de iterações do algoritmo para resolver as restrições. No entanto, por existirem falsos máximos relativos no vetor, não se pode reconhecer todo círculo presente no vetor como um objeto.

Essas duas etapas, de determinação de centros de círculos e reconhecimento dos objetos, tornam a interpretação do espaço de Hough tolerante a ruídos, robusta a falsos máximos relativos e eficaz no reconhecimento de todos os objetos.

5.1.9 Resultados obtidos

A implementação da TH foi feita em linguagem C++ e utilizou-se a biblioteca de visão computacional OpenCV (INTEL, 2006). Os resultados apresentados foram obtidos em um computador equipado com um processador Intel Pentium 4 *Hyper Threading*, com velocidade de processamento de 3,2 GHz. O programa foi executado em sistema operacional Windows XP, configurado para prioridade em tempo real.

Como a intenção é utilizar a TH para uma aplicação em tempo real, a avaliação de desempenho considera como sendo o tempo máximo aceitável para a execução do algoritmo o intervalo de tempo de uma captura de imagem. Os sistemas de captura comumente utilizados no futebol de robôs, como o descrito nesse trabalho, são capazes de adquirir imagens a 30 quadros por segundo. Logo, a janela de tempo máxima disponível para processamento é de 33 milissegundos. Os tempos descritos a seguir, na Tabela 5.1, foram obtidos pela média dos

tempos observados nas 30 imagens reais capturadas, nas duas resoluções utilizadas, 640 x 480 pixels e 320 x 240 pixels.

O tempo foi medido pela média da execução das etapas em um laço de 500 vezes para assegurar uma medição de tempo mais precisa. Os parâmetros configuráveis, tais quais, limiares de histerese do filtro Canny, raio da circunferência e número de votos mínimo, foram fixados igualmente para todas as imagens testadas, salvo alterações para as diferentes resoluções.

Tabela 5.1 – Tempos de execução do sistema de visão

Função	Tempo de execução (ms) 320×240	Tempo de execução (ms) 640×480
Subtração de fundo	0,8412	5,1667
Conversão de cor + filtro Canny	1,6163	7,8435
Geração do espaço de Hough	1,4621	6,3683
Determinação de centros de círculo	0,0334	0,0267
Reconhecimento dos objetos	0,0031	0,0023
Total	4,0674	19,4083

Todas as etapas, em ambas as resoluções, apresentaram desvio padrão da ordem de 10^{-6}. A diferença entre a soma das etapas e o tempo total é pequena. Pode-se considerar tal diferença como erro de arredondamento. Qualquer que seja a resolução, o sistema implementado é capaz de reconhecer os objetos, não apenas em tempo real, mas disponibilizando, no mínimo, 13 milissegundos da janela de tempo máxima disponível para outros processamentos, como estratégia e controle dos robôs.

Como já citado anteriormente, para efetuar os testes, todos os parâmetros ajustáveis do sistema foram fixados e utilizados igualmente em todas as imagens. Para imagens com resolução de 640 x 480, o raio *r* foi fixado em 8 pixels, enquanto o número de votos mínimo fixado em 16. Já para imagens com resolução de 320 x 240, o raio *r* foi fixado em 4 pixels e o número de votos mínimo em 10. Os limiares de histerese do filtro Canny foram mantidos fixos para todas as imagens, nas duas resoluções, em valores de 75 para o limiar baixo e 150 para o limiar alto. Tais valores foram estipulados empiricamente, após algumas observações. A intenção foi determinar os limiares do filtro Canny e o número de votos mínimo baixos o suficiente para que todos os pontos que representassem centros reais na imagem fossem detectados. A Figura 5.4 mostra que muitos outros pontos, referentes a falsos máximos relativos são gerados no espaço de Hough. Mesmo assim, o sistema mostrou-se robusto,

detectando todos os objetos (seis robôs, três de cada time, e uma bola) presentes nas 30 imagens utilizadas nos testes, em ambas as resoluções.

No entanto, após os testes observou-se que existe um caso muito específico, praticamente impossível de ocorrer em uma partida de Futebol de Robôs, em que o sistema pode vir a falhar. A disposição dos robôs na imagem é tal que até mesmo a visão humana é incapaz de distingui-los com absoluta certeza. A seqüência de imagens da Figura 5.7 exemplifica duas situações em que o sistema pode vir a falhar, não detectando os robôs na imagem. Cada linha representa um caso com imagem real, seguida do filtro Canny, o espaço de Hough, os círculos detectados e, por fim, os objetos reconhecidos.

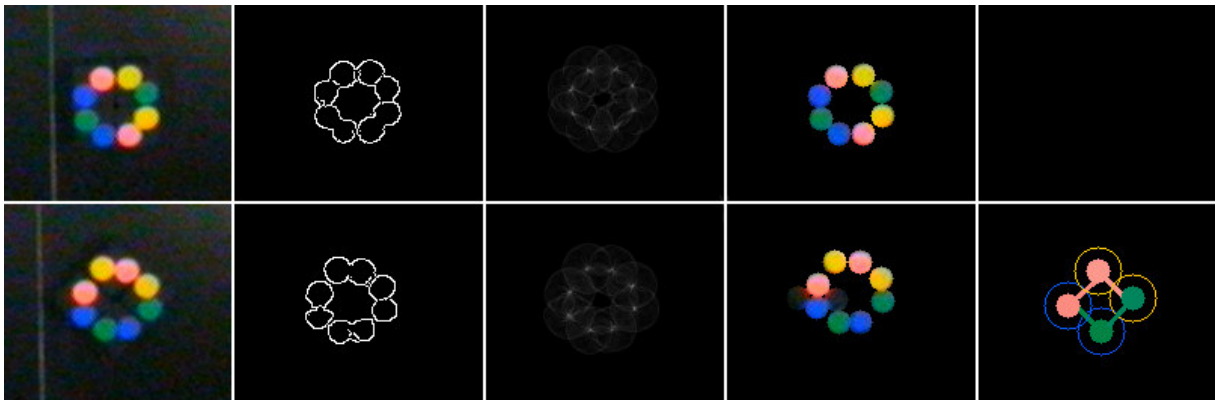

Figura 5.7 – Caso especial em que o sistema pode não detectar os robôs

No primeiro caso especial os robôs não podem ser reconhecidos, pois toda circunferência de cor primária tem dois nós filhos de cor secundária próximos o suficiente para que não seja possível resolver o problema. No segundo caso um dos robôs foi ligeiramente deslocado, resultando em uma circunferência de cor primária com apenas um nó filho de cor secundária próximo, obtendo êxito no reconhecimento.

A utilização da TH mostrou-se eficiente e funcional na detecção de todos os círculos presentes nas imagens do domínio do Futebol de Robôs testadas. O sistema implementado mostrou-se robusto e tolerante a ruídos, além de tolerante a variação de cor, pois apenas considera a forma dos objetos, utilizando informações de cor apenas para distinguir os mesmos. Assim, as cores podem ser calibradas em um intervalo de valores muito mais aberto, em relação a sistemas de reconhecimento baseados em cores. O método utilizado para a interpretação do espaço de Hough é capaz de diferenciar falsos máximos relativos de pontos de centro de círculos reais da imagem.

Os tempos medidos, juntamente com os testes de reconhecimento dos objetos, demonstram que é possível utilizar o sistema descrito em tempo real, porque supera as

necessidades de tempo, precisão e robustez exigidas de um sistema de visão computacional para o reconhecimento dos objetos em um ambiente de Futebol de Robôs.

5.2 Sistema de navegação e controle utilizado

O sistema de navegação utilizado nesse trabalho é o mesmo sistema que compõe o time de Futebol de Robôs do Centro Universitário da FEI, desenvolvido pelo aluno José Ângelo Gurzoni Jr., integrante da equipe de Futebol de Robôs do Centro Universitário da FEI.

O sistema de navegação deve determinar os pontos pelos quais o algoritmo de controle deve deslocar os robôs. São definidas zonas de desvio em torno de cada obstáculo, definidas por um quadrilátero cujo centro coincide com o centro do obstáculo, para que os pontos gerados resultem em uma trajetória capaz de atingir o ponto final desejado evitando colisões com outros robôs. É verificado se não há intersecção entre a reta de trajetória e as retas dos quadriláteros das zonas de desvio. Caso haja intersecção, é calculada a curva de Bezier de grau cúbico.

Curvas de Bezier de grau cúbico, descritas em detalhes por Azevedo e Conci (2003), são curvas polinomiais que requerem quatro pontos de referência, conforme mostrado na equação (5.7), sendo que a curva gerada passa apenas pelo primeiro e pelo quarto pontos:

$$p(t) = at^3 + bt^2 + ct + p_0, para\ \ 0 < t < 1 \qquad (5.7)$$

Onde:

- p é um ponto de coordenadas cartesianas (x, y)
- $c = (p_1 - p_0)$
- $b = 3(p_2 - p_1) - c$
- $a = p_3 - p_0 - b - c$
- $t = 0$ em p_0 e $t = 1$ em p_3

As curvas de Bezier podem representar polinômios de maior grau, com o número n de pontos de controle. No entanto, o polinômio seria de grau n – 1, exigindo um custo computacional muito alto. O cálculo de curvas de Bezier de grau cúbico tem um custo computacional baixo, motivo pelo qual são utilizadas nesse trabalho.

Considerando que os pontos p_0 e p_3 são, respectivamente, os pontos inicial e final da trajetória, os pontos de controle p_1 e p_2 são definidos pelas extremidades das zonas de desvio. O ponto p_1 é definido pela extremidade da zona de desvio da primeira intersecção. A partir de p_1 uma nova reta é traçada até o ponto final da trajetória. Caso haja outra intersecção com uma segunda zona de desvio, então o ponto p_2 é definido pela extremidade dessa segunda zona de desvio. Com os quatro pontos definidos, é calculada a curva de Bezier de grau cúbico, conforme mostra a Figura 5.8.

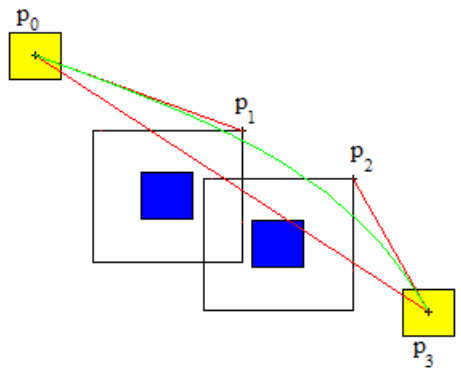

Figura 5.8 – Curva de Bezier de grau cúbico

No entanto, apenas dois pontos de controle podem não ser suficientes para o desvio dos obstáculos. Para esse problema, a trajetória a ser definida é segmentada em curvas parciais, onde o ponto final p_3 de uma curva coincide com o ponto inicial q_0 da curva seguinte. Mas a composição de curvas parciais pode apresentar descontinuidade entre as curvas, exigindo movimentos bruscos do robô para seguir a trajetória, o que não é possível na prática, pois o robô não responde a variações bruscas de velocidade dada a sua inércia. A Figura 5.9 exemplifica um caso onde duas curvas parciais compõem uma trajetória com descontinuidade.

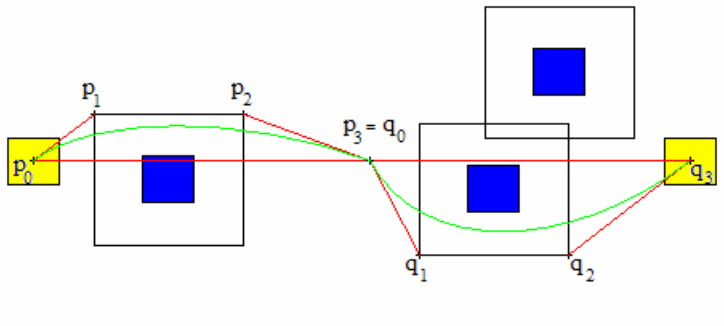

Figura 5.9 – Trajetória com descontinuidade entre as curvas de Bezier

Para criar continuidade entre as curvas é necessário que a inclinação entre os dois últimos pontos p_2 e p_3 de uma curva e os dois primeiros pontos q_0 e q_1 seja constante. Isso

significa que a primeira derivada deve ser constante. Dessa forma, as equações (5.8) e (5.9) garantem que haja continuidade entre as curvas de Bezier.

A equação (5.8) define que a primeira derivada seja constante entre as retas, enquanto a equação (5.9) faz o mesmo, mas de forma simplificada, utilizando as coordenadas dos pontos das curvas:

$$\frac{\partial}{\partial t} p(1) = -\frac{\partial}{\partial t} q(0) \tag{5.8}$$

$$p_3 - p_2 = q_1 - q_0 \tag{5.9}$$

A Figura 5.10 demonstra um caso onde há continuidade entre as curvas de Bezier.

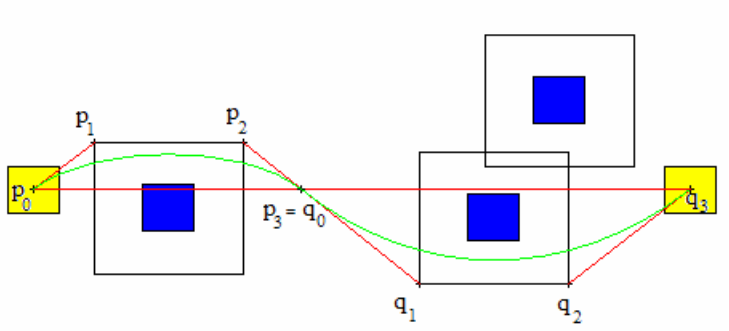

Figura 5.10 – Trajetória com continuidade entre as curvas de Bezier

Então, as curvas são calculadas e, caso haja descontinuidade, é calculada a primeira derivada, a partir da equação (5.9), para definir os pontos final p_3 e inicial q_0 das curvas para que a trajetória seja contínua. Dessa forma, são gerados os pontos que são utilizados pelo algoritmo de controle para a movimentação dos robôs através dos pontos definidos pela trajetória, que considera tanto os obstáculos para evitar colisão, como a orientação desejada do robô ao chegar à posição final da trajetória.

O algoritmo de controle, que segue a trajetória definida pelo sistema de navegação, utilizado para os robôs reais é o mesmo algoritmo disponibilizado pelo simulador *Simurosot*, descrito anteriormente. No entanto, o intervalo das velocidades v_r e v_l foi redefinido para que a menor velocidade atribuída a um motor seja capaz de vencer a inércia do robô. Os valores de velocidade calculados pelas equações (4.1) e (4.2) são interpolados com o intervalo de velocidades válidas para os robôs reais, entre 40 e 127. Com as velocidades v_r e v_l redefinidas para todos os robôs, o computador envia um pacote de dados contendo essas velocidades.

5.3 Protocolo de comunicação

O pacote de dados enviado pelo computador aos robôs segue uma formatação conforme o protocolo proposto por Martins, Tonidandel e Bianchi (2005). Esse protocolo utiliza o conceito de *broadcast*, onde todos os robôs recebem todos os dados e cada robô interpreta o bloco de dados referente ao seu endereço. A Figura 5.11 exemplifica o pacote de dados utilizado nesse trabalho, ajustado para encapsular informações de velocidade dos motores de cinco robôs. O pacote tem tamanho de 12 bytes, composto por 10 bytes de dados, encapsulados por um byte de início, para indicar o começo do pacote e um byte de finalização, para indicar o fim do pacote de dados.

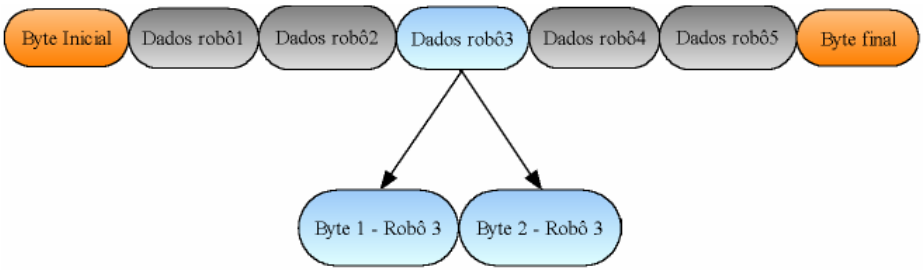

Figura 5.11 – Pacote de dados do protocolo proposto

Os 10 bytes de dados recebidos pelo robô são armazenados e, de acordo com o endereço atribuído ao robô, os 2 bytes referentes a esse endereço são interpretados. Na Figura 5.11, por exemplo, o robô cujo endereço é 3 interpreta o quinto e sexto bytes de dados do pacote recebido. Cada um desses bytes representa um valor de velocidade entre 40 e 127, sendo o bit mais significativo referente ao sentido de rotação do motor, 0 indica sentido horário, enquanto 1 indica sentido anti-horário. O byte 1 do robô de endereço 3 (quinto byte de dados do pacote) contém as informações referentes ao motor esquerdo, enquanto o byte 2 (sexto byte do pacote de dados) contém as informações do motor direito.

5.4 Cube^3 – O robô desenvolvido

Os robôs desenvolvidos para esse trabalho são baseados no projeto de Martins (2004). Foram feitas algumas mudanças para adequar os novos modelos de motores, bateria e módulo de radiofreqüência, originando a terceira versão de *hardware* e *software*. A seguir, uma breve descrição sobre os aspectos principais do robô é apresentada.

O robô Cube^3 tem a parte eletrônica separada em duas placas, a placa de controle e a placa de alimentação e etapa de potência dos motores. A separação do circuito elétrico é

fundamental para o aproveitamento do espaço disponível e para diminuir a incidência de ruído na parte do circuito que utiliza sinais digitais.

A seguir, é descrito, em detalhes, cada módulo do robô Cube^3.

5.4.1 Placa de controle do robô Cube^3

O circuito responsável pela comunicação com o computador, processamento das informações de velocidade recebidas, atribuição e controle dessas velocidades, compõe a placa de controle do robô. É utilizado um microcontrolador de 8 bits que opera a uma freqüência de 10 MHz, modelo PIC 16F876A da fabricante *Microchip* (2007), como o núcleo de processamento do robô Cube^3.

Para a comunicação com o computador, o microcontrolador utiliza um módulo de radiofreqüência que opera na faixa de freqüência entre 2,4 GHz e 2,5 GHz e é capaz de transmitir e receber dados à taxa de 1 Mbps modulados em GFSK e com checagem de erro CRC de 16 bits, modelo TRW-24G, da fabricante *Wenshing* (2007).

A placa de controle ainda é composta por uma chave de *reset* do microcontrolador e 4 chaves do tipo *DIP-Switch*, sendo uma chave para a seleção entre dois canais de freqüência de operação e três chaves para determinação do endereço de cada robô, possibilitando até 7 endereços diferentes, visto que o endereço zero executa uma rotina de auto-teste do robô. No entanto, como o pacote de dados foi ajustado para cinco robôs, os endereços 6 e 7 não são utilizados.

Dois leds indicadores mostram o estado de funcionamento do robô. Um led pisca intermitentemente para indicar que o *software* do microcontrolador está sendo executado, enquanto o outro led acende quando um pacote de dados, enviado pelo computador, é recebido pelo robô.

A conexão com a placa de potência é feita através de um conector de 10 pinos. Dois desses pinos são utilizados para alimentar a placa de controle com uma tensão de 3,3 V, enquanto quatro pinos levam os sinais de controle do sentido de rotação dos dois motores. Outros dois pinos transmitem o sinal de freqüência fixa de 10 KHz e largura de pulso variável PWM (*pulse width modulation*), gerado pelo microcontrolador para controlar a velocidade dos motores. Por fim, dois pinos trazem a informação de sensores de corrente elétrica dos motores.

Um circuito integrado de portas lógicas inversoras *schmitt-trigger* 7414 é utilizado pelo microcontrolador como filtro para os sinais provindos dos sensores de corrente dos motores e para os sensores de odometria. Esses sensores de odometria capturam 128 pulsos por revolução através de um disco *encoder* estampado na roda.

Ao receber, através do módulo de radiofreqüência, um pacote de dados enviado pelo computador, o microcontrolador verifica quais são as velocidades dos motores de acordo com o endereço configurado nas chaves correspondentes. Após a interpretação dos bytes de dados, é calculada a largura de pulso do sinal PWM a ser enviada aos motores para que as velocidades sejam atingidas. Os sinais PWM são transmitidos para os motores, assim como o sinal de sentido de rotação. O controle de velocidade que utiliza a malha de realimentação dos sensores de odometria não foi implementado para esse trabalho.

A Figura 5.12 apresenta o esquema elétrico da placa de controle do robô Cube[3].

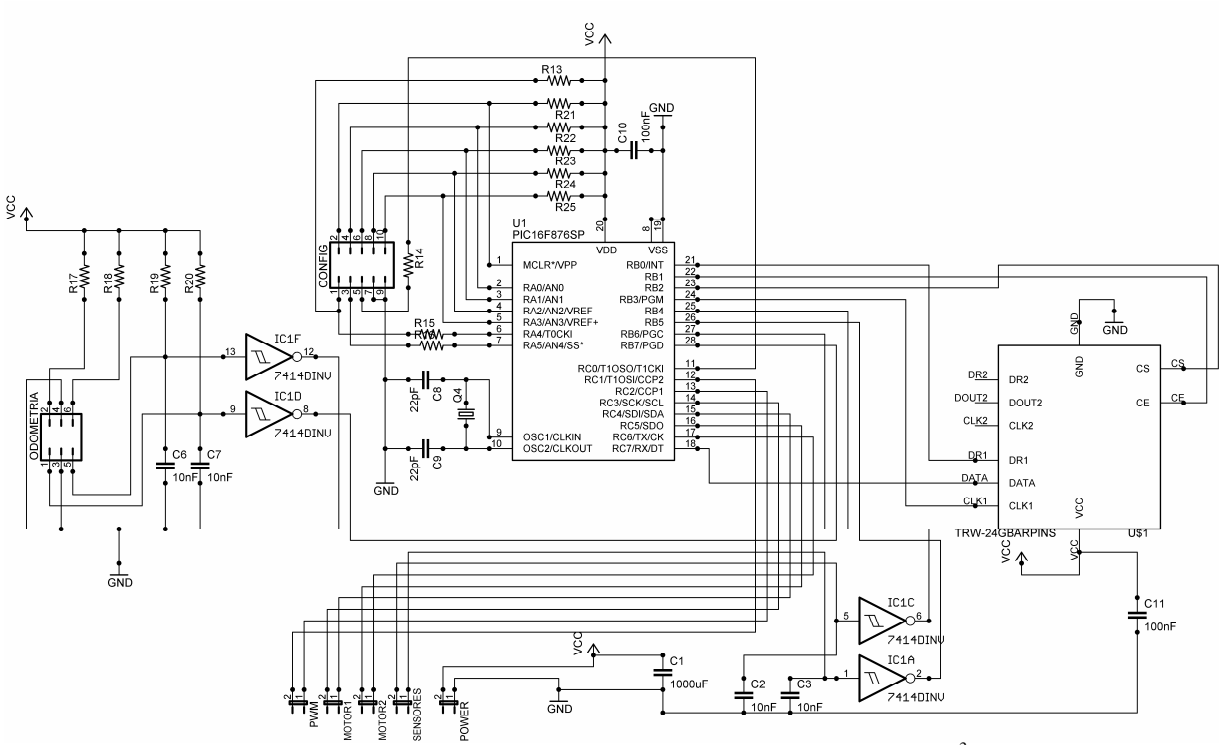

Figura 5.12 – Esquema elétrico da placa de controle do robô Cube[3]

5.4.2 Placa de potência do robô Cube[3]

A placa de potência do robô Cube[3] é responsável por alimentar a placa de controle, servir de interface entre o microcontrolador e os motores, além de fornecer a potência elétrica exigida pelos mesmos.

A bateria de lithium-polímero utilizada fornece 7,4 V de tensão. Essa tensão é reduzida em duas etapas, através dos reguladores de tensão LM7805 e LM117T3.3, para que a tensão de 3,3 V da placa de controle seja obtida. Um led indica quando a tensão de 3,3 V está disponível e uma chave interrompe o fornecimento de energia da bateria para todo o circuito do robô, desligando-o.

Um circuito integrado L298N é utilizado como interface entre os sinais digitais vindos do microcontrolador da placa de controle e os motores. Esse circuito integrado é composto por duas pontes H completas, capazes de acionar os motores aplicando diretamente a tensão nominal de 7,4 V fornecida pela bateria. O circuito integrado L298N eleva a tensão do sinal PWM de 3,3 V, proveniente do microcontrolador, para 7,4 V e é capaz de inverter o fluxo de corrente elétrica nos motores conforme os sinais digitais de sentido de rotação enviados pelo microcontrolador. Diodos *Schottky* de chaveamento rápido BYV27200 são utilizados para proteger o circuito integrado L298N contra a corrente elétrica inversa geradas pelos motores.

Os motores utilizados são modelos 2619006SR, da fabricante FTB (2007). Têm tensão nominal de 6 V e caixa de engrenagens de redução embutida com relação de 8:1. O robô é capaz de atingir uma velocidade de até 1,5 m/s.

A Figura 5.13 apresenta o esquema elétrico da placa de potência do robô Cube[3].

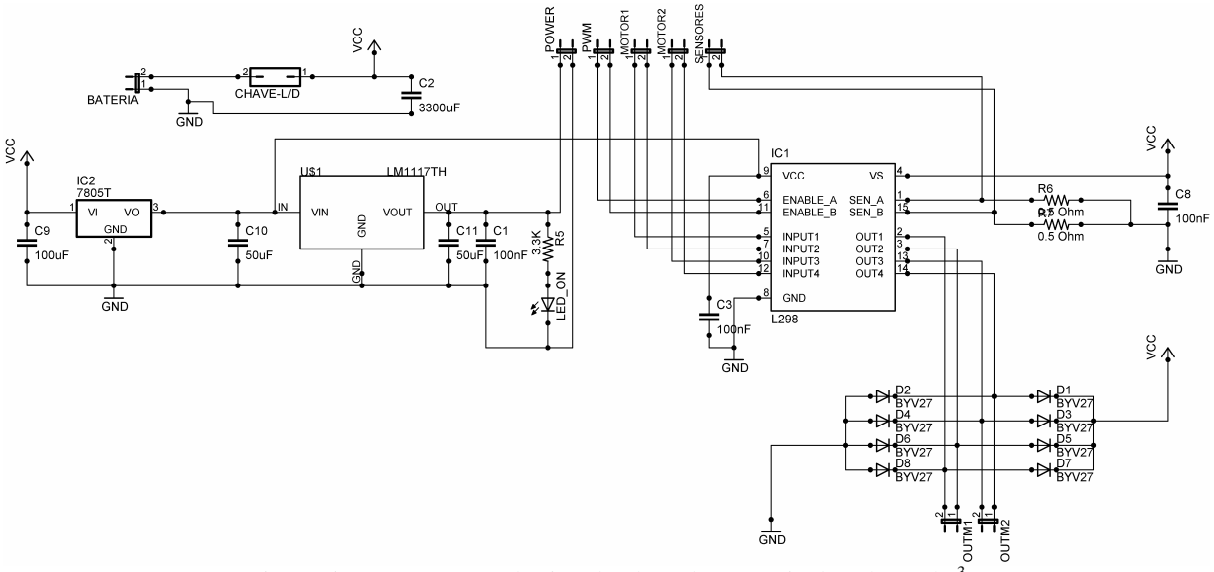

Figura 5.13 – Esquema elétrico da placa de potência do robô Cube[3]

A estrutura mecânica e as rodas, confeccionadas em alumínio, foram projetadas pelo aluno de iniciação científica Fernando Perez Tavarez, estudante de engenharia mecânica e integrante da equipe de Futebol de Robôs do Centro Universitário da FEI.

A Figura 5.14 apresenta as placas de potência e controle descritas anteriormente, enquanto a Figura 5.15 apresenta os três robôs Cube^3 construídos para esse trabalho.

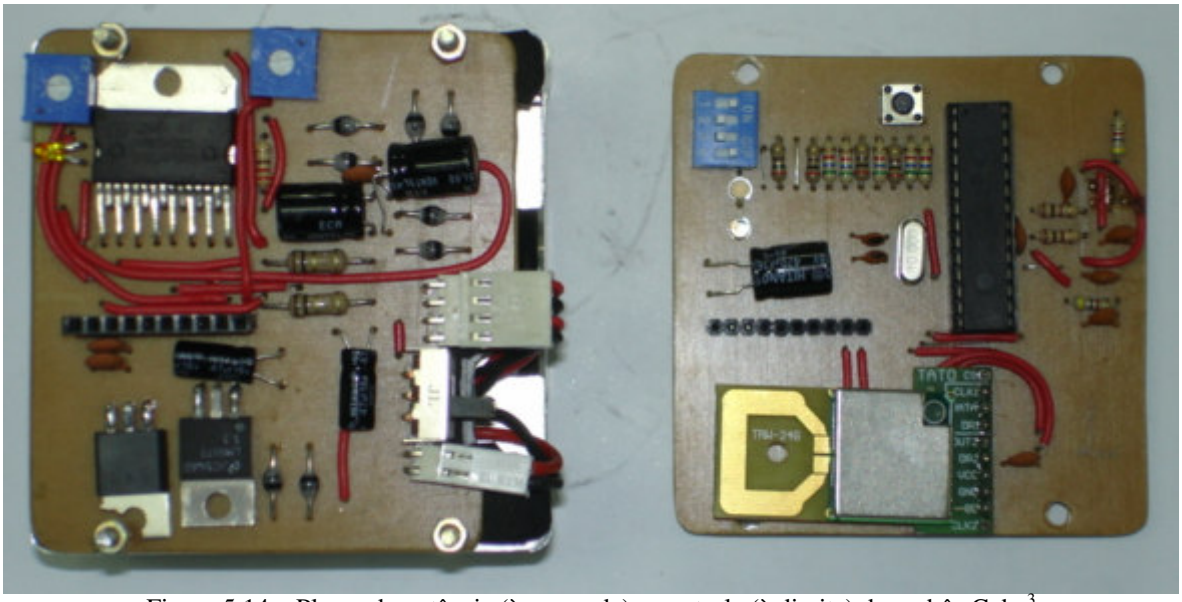

Figura 5.14 – Placas de potência (à esquerda) e controle (à direita) dos robôs Cube^3

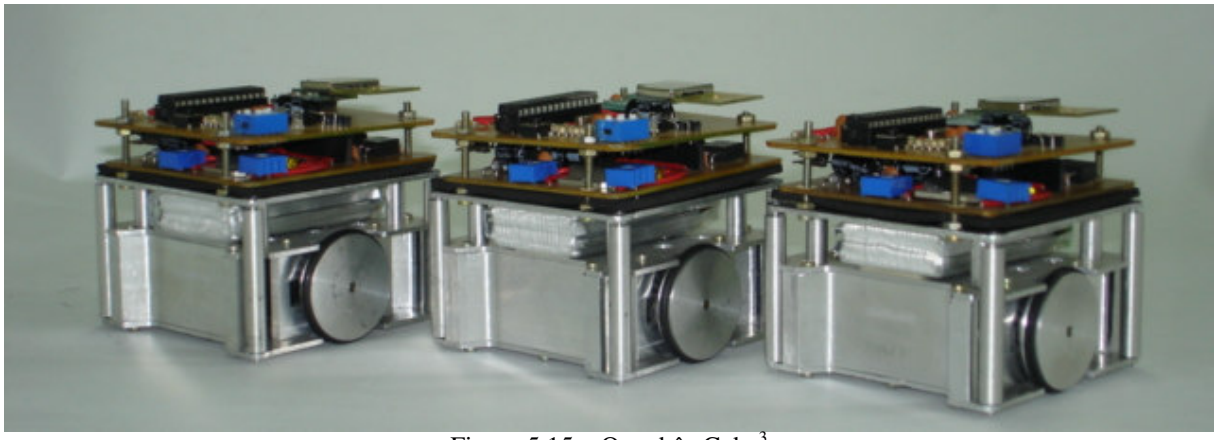

Figura 5.15 – Os robôs Cube^3

Esse capítulo apresentou o sistema de visão e os robôs desenvolvidos para esse trabalho, assim como o sistema de navegação utilizado. Esses módulos compõem o time de Futebol de Robôs da categoria *Mirosot* utilizado em conjunto com o sistema de AR descrito no próximo capítulo para a realização dos experimentos.

6 APRENDIZADO POR REFORÇO NO DOMÍNIO DE FUTEBOL DE ROBÔS

Nos capítulos anteriores foi feita uma revisão bibliográfica sobre AR e algumas técnicas de aceleração do AR. Também foi apresentado o domínio do Futebol de Robôs, contexto no qual está inserido o sistema de AR. O presente capítulo apresenta a proposta dessa dissertação, definindo o sistema de AR utilizado no domínio do Futebol de Robôs.

6.1 Introdução

Littman (1994) propôs um jogo de Markov (jogo modelado por um PMD) inserido em um ambiente de Futebol de Robôs muito simplificado, onde apenas dois jogadores, denominados A e O, competem entre si em um campo de futebol representado por uma grade de 5 x 4 regiões, conforme pode ser visto na Figura 6.1. Nesse ambiente, uma célula da grade pode ser ocupada por apenas um agente e a bola, representada por um círculo, está sempre de posse de um dos dois agentes. Na Figura 6.1, por exemplo, o jogador O está com a posse de bola.

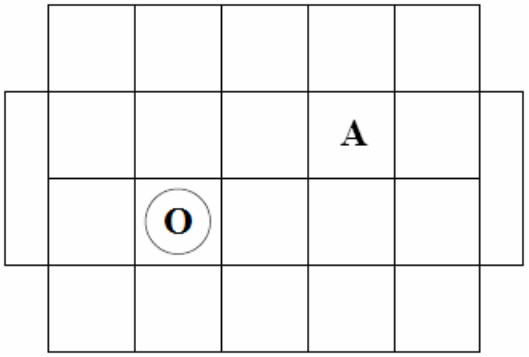

Figura 6.1 – Ambiente proposto por Littman (1994)

O conjunto de estados é definido pela posição dos jogadores A e O, enquanto o conjunto de ações é composto por cinco ações. Cada ação indica uma direção de deslocamento do agente que a executou, podendo ser “Norte”, “Sul”, “Leste” e “Oeste”. Existe, ainda, uma ação que mantém o agente parado.

Quando um jogador executa uma ação cujo deslocamento seria para fora da grade, ele permanece na posição em que estava antes de executar a ação. Caso o jogador esteja com a posse da bola, a mesma se desloca junto com esse jogador e a posse da bola só é perdida quando for executada uma ação cujo deslocamento seria para dentro da célula onde se encontra o jogador adversário. Cada jogador marca um ponto quando está com a posse de bola e executa uma ação que o desloca para dentro do gol adversário.

Bianchi (2004) utilizou esse ambiente simplificado de Futebol de Robôs, proposto por Littman (1994), para efetuar experimentos comparativos de algoritmos de AR com e sem a utilização de heurísticas para acelerar o aprendizado. Porém, não foram feitos experimentos em um ambiente simulado de Futebol de Robôs mais complexo, tampouco em um ambiente real.

A proposta de pesquisa dessa dissertação é uma extensão do trabalho de Bianchi (2004). Essa proposta tem o objetivo comparar os algoritmos Q-learning, Q(λ) e QS, com e sem a utilização de heurísticas para a aceleração do aprendizado, em um ambiente simulado de Futebol de Robôs mais complexo, o simulador *Simurosot*, além de analisar a influência do uso de heurísticas em um ambiente real da categoria *Mirosot*.

6.2 Arquitetura do sistema de Aprendizado por Reforço

A abordagem do problema de aprendizado foi feita com o intuito de manter o mesmo nível de abstração do conjunto de estados e conjunto de ações do ambiente proposto por Littman (1994). No entanto, foram feitas algumas adaptações na definição do conjunto de estados, conjunto de ações e da grade de regiões que representa o campo (abstrações citadas na seção 3.1). Essas adaptações se justificam pelo fato de que os ambientes da categoria *Simurosot* e *Mirosot* são mais complexos, dinâmicos e não-determinísticos.

No decorrer do texto, o agente representa o jogador A, que deve aprender através dos algoritmos de AR, enquanto o oponente representa o jogador O, que, para esse trabalho, é um agente robótico que seleciona as ações de modo aleatório.

6.2.1 Conjunto de estados

A primeira consideração a ser feita sobre o ambiente de Futebol de Robôs das categorias *Simurosot* e *Mirosot* em relação ao ambiente proposto por Littman (1994) é que ambos os jogadores e a bola podem ocupar qualquer região distinta um do outro, ou ainda, todos podem estar em uma mesma região. Outra observação é que a bola é livre durante todo o jogo, pois nenhum agente pode prender a bola a si mesmo.

Isso implica em um aumento do número de variáveis que determinam o estado do ambiente, pois é necessário considerar as posições de ambos os jogadores e da bola, resultando na tupla $\langle x_a, y_a, x_b, y_b, x_o, y_o \rangle$, representando as coordenadas (x, y) da região onde estão, respectivamente, o agente (x_a, y_a), a bola (x_b, y_b) e o oponente (x_o, y_o).

A representação do campo em regiões, que pode ser vista na Figura 6.2, é definida por uma grade de 7 x 5 regiões. O aumento do número de colunas se deu pelo fato de que os jogadores podem deslocar-se para dentro dos gols mesmo sem a bola, sendo que as regiões dos gols são estados terminais apenas quando a bola entra em um dos dois gols. Em relação ao número de linhas, foi adicionada uma linha que considera os limites inferior e superior dos gols como sendo os limites verticais das regiões dessa linha. Excetuando as regiões da linha citada, as outras áreas foram divididas simetricamente.

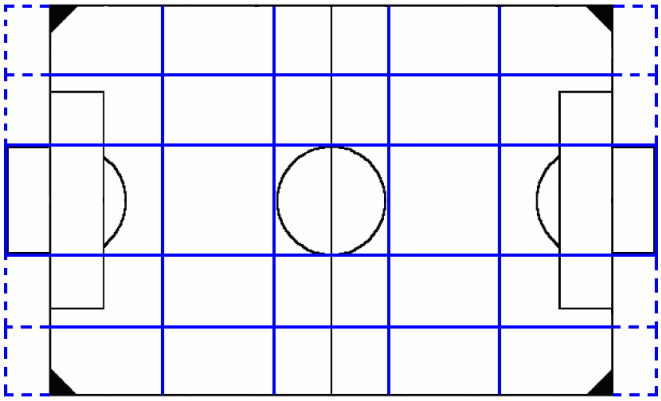

Figura 6.2 – Campo de jogo dividido em 7 x 5 regiões

Como as dimensões do campo são conhecidas, as fronteiras de cada região da grade são definidas virtualmente. Um jogador, ou a bola, está posicionado em uma região discreta quando sua posição em relação ao campo estiver dentro das fronteiras dessa região.

Apesar de o campo ser dividido em 7 x 5 regiões, as 8 regiões pontilhadas, mostradas na Figura 6.2, não fazem parte do conjunto de estados porque estão fora do campo e nenhum jogador ou a bola podem alcançar essas posições. Então, jogadores e bola podem ocupar 27 regiões distintas, resultando em um conjunto de 19.683 estados, tendo como estados terminais todos aqueles onde a bola estiver nas regiões que representam os gols.

6.2.2 Conjunto de ações

Outra consideração a ser feita sobre os ambientes das categorias *Simurosot* e *Mirosot* em relação ao ambiente proposto por Littman (1994) é que a bola desloca-se livremente pelo campo. Então, o conjunto de ações do ambiente de Littman (1994) foi estendido para que os agentes possam manipular a bola.

As sete ações que compõem esse conjunto de ações são, na verdade, macro-ações programadas manualmente, que consideram tanto as informações das regiões de posição, como as informações de posição e orientação dos agentes e da bola dentro dessas regiões.

Todas essas ações utilizam o sistema de navegação e controle para determinação da trajetória, desvio de obstáculos e controle de velocidade e posicionamento dos agentes em ambos os ambientes simulado (*Simurosot*) e real (*Mirosot*). A seguir, as ações são apresentadas.

A ação "*Stop*". Quando a ação "*Stop*" é executada, o agente simplesmente pára na posição em que ele se encontra. Não é feito nenhum cálculo de trajetória e o algoritmo de controle apenas atribui velocidade zero aos motores das rodas dos robôs.

A ação "*Move Up*". O ponto central da região imediatamente acima à qual o jogador se encontra é calculado através das fronteiras virtuais dessa região e é definido como o ponto final da trajetória. O algoritmo de controle, então, calcula as velocidades dos motores de cada roda do robô, para que o mesmo chegue a esse ponto final. Caso a região imediatamente acima seja fora do campo de jogo, o movimento do jogador é interrompido e ele permanece parado até a execução da próxima ação.

A ação "*Move Down*". Analogamente à ação "*Move Up*", é calculado o ponto central da região imediatamente abaixo àquela em que o jogador se encontra e esse ponto é definido como o final da trajetória. Da mesma maneira, caso a região imediatamente abaixo seja fora do campo de jogo, o movimento do jogador é interrompido, permanecendo parado até que a próxima ação seja executada.

A ação "*Move Forward*". Essa ação considera o lado para o qual o jogador deve fazer os gols. A região horizontal imediatamente mais próxima ao gol adversário, em relação à região na qual o jogador se encontra, tem seu ponto central calculado e definido como o ponto final da trajetória. Caso essa região seja para fora do campo de jogo, o jogador tem seu movimento interrompido e assim permanece até que a próxima ação seja executada.

A ação "*Move Backward*". Análoga à ação "*Move Forward*", essa ação calcula o ponto central da região horizontal imediatamente mais próxima ao gol do próprio jogador, em relação à região em que o mesmo se encontra, definindo esse ponto central como sendo o final da trajetória. Novamente, caso a região de destino seja para fora do campo de jogo, o jogador tem o movimento interrompido e continua parado até que a próxima ação seja executada.

A ação "*Get Ball*" é uma das duas ações que foram inseridas no conjunto de ações para que os jogadores possam manipular a bola. Essa ação considera tanto a posição dos jogadores e da bola na grade de regiões, como a posição e orientação em relação ao campo.

Figura 6.3 – Exemplo de execução da ação "*Get Ball*"

São considerados o lado para onde o jogador deve fazer os gols e as posições, em relação ao campo, da bola e do jogador que está executando essa ação. É definido como ponto final da trajetória o ponto imediatamente atrás da bola. O resultado dessa ação é que o jogador que a executa vai em direção à bola e posiciona-se na linha horizontal entre a bola e seu próprio gol, sendo que a trajetória é calculada pelo sistema de navegação e o deslocamento para os pontos da trajetória é feito pelo algoritmo de controle. A Figura 6.3 exemplifica a execução da ação "*Get Ball*", onde o agente (robô de cores amarelo e rosa), que deve fazer gols para a direita, está posicionado imediatamente atrás da bola, na linha horizontal entre a bola e seu próprio gol.

A possibilidade de execução da ação "*Get Ball*" foi restringida às situações em que o jogador se encontra na mesma região da bola ou em alguma região vertical, horizontal ou diagonalmente vizinha à região em que a bola se encontra. Caso o jogador esteja em alguma das demais regiões, seus movimentos são interrompidos e ele permanece parado até que uma nova ação seja executada.

A outra ação inserida no conjunto de ações para que os jogadores possam manipular a bola é a ação "*Kick to Goal*". Essa ação também considera tanto a posição dos jogadores e da bola na grade de regiões, como a posição e orientação em relação ao campo.

Muito similar à ação "*Get Ball*", a ação "*Kick to Goal*" define que o jogador que a executa deve empurrar a bola em direção ao gol adversário. No entanto, por se tratar de uma ação que exige uma proximidade da bola, a possibilidade de execução dessa ação foi restringida às situações em que o jogador que a executa está na mesma região em que se

encontra a bola. Para todas as outras situações em que o jogador tentar executar a ação "*Kick to Goal*" e não estiver na mesma região da bola, seus movimentos são interrompidos e ele permanece parado até a execução de uma nova ação.

Para que o jogador seja capaz de empurrar a bola em direção ao gol adversário, ele deve se deslocar até a bola. É determinado o ângulo de impacto do jogador com a bola. Esse ângulo é calculado por trigonometria, através da posição da bola e do ponto central da linha vertical que delimita o gol. Com o ponto de impacto (a posição da bola) e o ângulo de impacto definidos, o sistema de navegação determina os pontos da trajetória, que são utilizados pelo algoritmo de controle para deslocar o robô.

6.2.3 Função de recompensa

Com os conjuntos de estados e ações definidos, é possível definir as recompensas recebidas pelo agente. O objetivo do agente para ganhar uma partida de futebol é marcar mais gols que o oponente. A partir desse objetivo, foram definidas as recompensas de modo que o mínimo de conhecimento sobre o domínio fosse inserido na função de recompensa, evitando limitar as escolhas do agente e tornando o aprendizado mais genérico.

O agente recebe uma recompensa de valor 1000 quando ocorre um gol a seu favor, enquanto recebe uma recompensa de valor -1000 quando sofre um gol. Esses valores foram os mesmos utilizados por Bianchi (2004).

Sempre que o agente executa uma ação a_t que não resulta na transição do estado s_t para um estado terminal, é recebida uma recompensa de valor -10. Essa recompensa é importante para assegurar que o agente sempre terá de executar uma seqüência de ações que resultem na recompensa de valor 1000, significando que o agente sempre irá tentar fazer gols e evitando que o agente aprenda a ficar parado, por exemplo.

Definiu-se, também, uma recompensa de valor -50 quando a ação selecionada for impossível de ser executada. Uma ação é impossível de ser executada quando a movimentação resultante da execução dessa ação levaria o agente para fora do campo de jogo. Foi necessária a definição dessa recompensa para que o agente aprenda rápido a evitar colisões violentas com as paredes do campo, o que poderia danificar o robô.

6.2.4 Função heurística

Para os algoritmos que utilizam a técnica de aceleração do aprendizado pelo uso de heurísticas, definiu-se o valor da heurística estacionária de acordo a equação (3.24), considerando o valor de recompensa mínimo como sendo -10, resultando em um valor h = 100. O valor -50 não foi considerado como r_{min} porque apenas alguns estados podem receber essa recompensa específica, enquanto o valor -10 é uma recompensa que todos os estados podem receber. Dessa forma é garantido que os valores que as estimativas $\hat{Q}^*(s,a)$ podem alcançar, no pior caso, anulam a influência do valor da heurística.

A função heurística, então, é mostrada na equação (6.1):

$$H(s_t, a_t) = \begin{cases} 100 & se\ a_t = \pi^H(s_t) \\ 0 & caso\ contrário \end{cases} \tag{6.1}$$

A heurística utilizada foi definida *a priori* através de conhecimento sobre o domínio. A política heurística indica que o agente deve preferir selecionar a ação "*Kick to Goal*" em relação às outras, quando estiver na mesma região da bola. Então, $H(s_t, a_t) = 100$ quando a ação a_t for "*Kick to Goal*" e quando, no estado s_t, as posições $x_a = x_b$ e $y_a = y_b$, significando que o agente e a bola estão na mesma região.

6.2.5 Transição de estados

A partir de um estado s_t, o agente deve selecionar uma ação a_t de acordo com a política utilizada $\pi(s_t) = a_t$. Quando o agente executa a ação a_t selecionada, o ambiente reage a essa ação e um novo estado s_{t+1} é alcançado. No entanto, como o ambiente é não-determinístico, a transição de estados ocorre por uma distribuição de probabilidades sobre o conjunto de estados S.

Então, considera-se que ocorreu uma transição de estados quando alguma das variáveis da tupla $\langle x_a, y_a, x_b, y_b, x_o, y_o \rangle$ que representa o estado sofre alteração. Essa alteração pode ocorrer pela execução de alguma ação por parte dos jogadores, ou ainda, pelo dinamismo do ambiente, pois se a bola estiver em movimento, assim ela continua, mesmo que os jogadores parem. No entanto, pode acontecer de a bola parar e os jogadores executarem ações que não alterem as variáveis da tupla $\langle x_a, y_a, x_b, y_b, x_o, y_o \rangle$. Nesse caso, após um segundo sem que as variáveis sejam alteradas e aconteça uma transição de estados naturalmente, o sistema força o agente a selecionar uma nova ação.

Essa consideração em que o sistema força a seleção de uma nova ação pelo agente é importante porque o agente só recebe uma recompensa, atualiza os valores da estimativa $\hat{Q}^*$ e seleciona uma nova ação após a transição de estados. Do contrário, o jogo poderia travar caso as variáveis de estado da tupla $\langle x_a, y_a, x_b, y_b, x_o, y_o \rangle$ não sofressem alteração.

6.3 Algoritmos de Aprendizado por Reforço implementados

Utilizando a arquitetura de AR descrita nesse capítulo, foram implementados quatro algoritmos de AR já conhecidos na literatura. São eles, o Q-learning, Q(λ), QS e HAQL, apresentados nos capítulos 2 e 3.

Além desses algoritmos, esse trabalho propõe dois novos algoritmos de AR, o algoritmo Q(λ) acelerado por heurísticas (*Heuristically Accelerated Q(λ)* – HAQ(λ)) e o algoritmo QS acelerado por heurísticas (*Heuristically Accelerated QS* – HAQS).

A seguir, são descritos alguns detalhes importantes na implementação dos algoritmos Q-learning, Q(λ) e QS e são apresentados os novos algoritmos HAQ(λ) e HAQS.

6.3.1 Parâmetros dos algoritmos de AR

Os parâmetros comuns aos algoritmos de AR foram mantidos sempre iguais para todos os seis algoritmos Q-learning, Q(λ), QS, HAQL, HAQ(λ) e HAQS.

A taxa de aprendizado α inicia com valor igual a 1 e decai conforme a regra apresentada na equação (2.7). No entanto, após sete visitas ao par estado-ação *(s, a)*, quando o valor de α decai para menos de 0,125, seu valor é mantido em 0,125 para que o agente continue aprendendo e considerando a mesma importância para todas as experiências após a sétima visita ao par *(s, a)*.

A taxa de exploração/explotação ε recebeu o valor 0,2, enquanto o fator de desconto de recompensas futuras γ foi estipulado em 0,9, ambos valores utilizados por Littman (1994) e Bianchi (2004).

6.3.2 Q-learning e HAQL

O algoritmo Q-learning implementado nesse trabalho segue exatamente as descrições feitas na seção 2.6.2. Já o algoritmo HAQL foi implementado conforme descrito na seção 3.3.3 e utiliza as heurísticas definidas conforme a seção 6.2.4.

6.3.3 Q(λ)-learning

Algumas peculiaridades de implementação da abordagem de Peng e Williams (1996) para o algoritmo Q(λ), descritas por Wiering e Schmidhuber (1998), são determinantes para o desempenho e a complexidade do algoritmo.

É utilizada uma lista duplamente ligada L, onde são armazenados os pares estado-ação (x, u) que foram visitados pelo menos uma vez. É computado o rastro de elegibilidade e apenas para os pares estado-ação que constam na lista ligada L. Caso o rastro de elegibilidade $e(x, u)$ alcance um valor menor que um limiar definido por $l \geq 0$, o par (x, u) é removido da lista L. Para garantir que um par (x, u) não seja inserido duas vezes na lista ligada, variáveis binárias *visitado*(x, u) são utilizadas para indicar que o par já foi visitado anteriormente e já está inserido na lista ligada L.

As operações de inserção e remoção dos pares na lista L têm um custo $O(1)$, pois cada par (x, u) possui um ponteiro para sua posição na lista. A remoção de pares (x, u) da lista L com rastro de elegibilidade $e(x, u) < l$ pode aumentar a velocidade do algoritmo significativamente, de acordo com Wiering e Schmidhuber (1998).

Para todo estado s e ação a, inicialize $e(s, a)$ e $Q(s, a)$ com zero
Observe o estado atual s_t
Repita infinitamente:
- (1) Selecione uma ação a_t de acordo com a política derivada de Q, (ex. (2.20))
- (2) Receba a recompensa imediata r_t
- (3) Observe o novo estado s_{t+1}
- (4) Compute o erro TD(0) conforme a equação (2.19)
- (5) Compute o erro TD(λ) conforme a equação (3.5)
- (6) Para todo par estado-ação (x, u) *da lista ligada* L:
  - (6.1) Compute o decaimento do rastro: $e(x, u) \leftarrow \gamma\lambda e(x, u)$
  - (6.2) Atualize a estimativa da função $Q^*(x, u)$ conforme a equação (3.6)
  - (6.3) Se $e(x, u) < l$:
    - (6.3a) $L \leftarrow L \setminus (x, u)$
    - (6.3b) *visitado*$(x, u) \leftarrow 0$
- (7) Atualize a estimativa da função $Q^*(s_t, a_t)$ conforme a equação (3.7)
- (8) Atualize o rastro $e(s_t, a_t)$ por substituição, conforme a equação (3.3)
- (9) Se *visitado*$(s, a) = 0$:
  - (9.1) *visitado*$(s, a) \leftarrow 1$
  - (9.2) $L \leftarrow L \cup (s, a)$
- (10) $s_t \leftarrow s_{t+1}$

Figura 6.4 – Algoritmo Q(λ) implementado nesse trabalho

A Figura 6.4 apresenta o algoritmo Q(λ) proposto por Peng e Williams (1996) com as modificações de implementação sugeridas por Wiering e Schmidhuber (1998). Com a utilização de rastros de elegibilidade por substituição (*replacing eligibility traces*) é possível estipular o número máximo de pares estado-ação que a lista L pode conter, de acordo com os

valores $\gamma\lambda$ e l. Logo, o número de atualizações a cada iteração do algoritmo torna-se gerenciável. Em relação ao valor de λ, demonstrações empíricas de Sutton (1988) apontam para valores de λ próximos de zero. Por isso, definiu-se o valor λ = 0,3. Ao limiar l foi atribuído o valor 0,000001. Dessa forma, como o valor $\gamma\lambda$ = 0,27, a cada 10 iterações um par estado-ação é removido da lista, pois o valor do rastro de elegibilidade irá decair abaixo do limiar l, resultando em uma lista L de tamanho máximo de 10 pares estado-ação.

6.3.4 QS-learning

Seguindo a descrição de Ribeiro, Pegoraro e Reali-Costa (2002), que também utilizou o ambiente simplificado de Futebol de Robôs proposto por Littman (1994), foram definidos os parâmetros para o algoritmo QS. A similaridade ocorre apenas no conjunto de estados, de acordo com a equação (3.8). A variável τ, que deve decair, no mínimo, à mesma velocidade que a taxa de aprendizado α recebe o valor inicial 0,7, o mesmo valor definido em Ribeiro, Pegoraro e Reali-Costa (2002). Foi necessário determinar uma taxa para a variável τ decair mais rapidamente que α, conforme descrita na equação (6.2).

$$\tau_t = \left[0.7 - 0.1 visitas_t(x,u)\right]^d \qquad (6.2)$$

Dessa forma, o valor de τ decai a zero a uma taxa maior que α, assegurando a convergência do algoritmo, lembrando que α nunca será menor que 0.125.

A similaridade entre estados é definida com base na similaridade descrita em Ribeiro, Pegoraro e Reali-Costa (2002). Na Figura 6.5, o agente é representado por A, enquanto a bola é representada por *b* e o oponente por O. O espalhamento de uma experiência ocorre em duas etapas. A primeira etapa espalha a experiência atual pela similaridade através da vizinhança, variando-se a posição do oponente para as oito regiões vizinhas, totalizando nove atualizações. A segunda etapa espalha as nove atualizações feitas pela similaridade através da vizinhança utilizando o conceito de similaridade através da simetria do campo de jogo, ilustrada pela linha pontilhada na Figura 6.5. O espalhamento pela similaridade através da simetria também é ilustrado na Figura 6.5 pelas posições simétricas do agente A', da bola *b'* e do oponente O', que representam o complemento das posições reais e do espalhamento pela vizinhança. Ações também são complementadas quando necessário. Para o exemplo dado na Figura 6.5, caso o agente A tenha selecionado a ação "*Move Up*", o complemento dessa ação para a posição do agente A' será "*Move Down*".

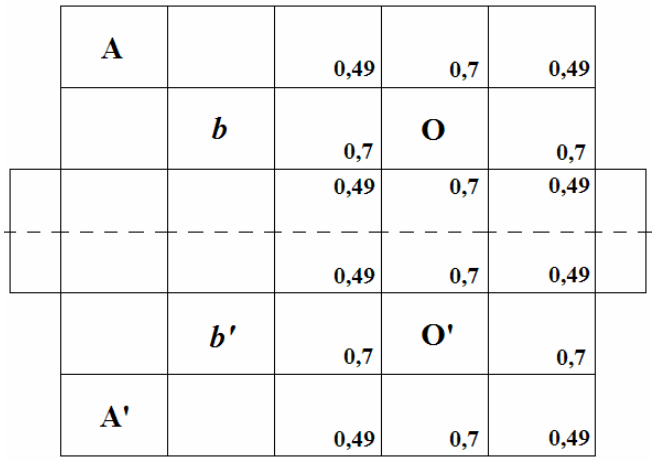

Figura 6.5 – Espalhamento por similaridade no algoritmo QS

Considerando a equação (3.11), para cada par estado-ação *(x, u)*, a quantização de similaridade *d* entre os estados assume valor 0 quando $x = s$. Assume valor 1 quando a posição do oponente em *x* for uma região horizontal ou verticalmente vizinha à sua posição em *s*. Quando a posição do oponente em *x* for uma região diagonalmente vizinha à sua posição em *s*, então *d* assume valor 2. Para os demais casos, *d* assume valor infinito.

O algoritmo QS descrito na seção 3.2.2 verifica todos os pares estado-ação existentes para espalhar a experiência de uma iteração através da similaridade entre o par (s_t, a_t) e os demais pares *(x, u)*. Essa característica pode comprometer o desempenho do algoritmo, visto que, a cada iteração, todos os pares estado-ação devem ser atualizados de acordo com a função $\sigma(x,u,s,a)$.

No entanto, para evitar que seja necessário percorrer todos os pares *(x, u)* possíveis, definiu-se manualmente os nove pares *(x, u)* a serem atualizados pela simetria através da vizinhança e os nove pares *(x, u)* complementares, seguindo as regras de quantização de similaridade da equação (3.11). Dessa forma, a única diferença entre o algoritmo QS implementado nesse trabalho e o algoritmo QS descrito na seção 3.2.2 está no quarto passo do algoritmo da Figura 3.3, onde não são percorridos todos os pares estado-ação *(x, u)*, mas apenas nove pares pela similaridade através da vizinhança do oponente e nove pares pela similaridade através da simetria do campo de jogo.

6.3.5 HAQ(λ)-learning

O algoritmo HAQ(λ) proposto nesse trabalho é uma variante do algoritmo Q-learning que combina a utilização das generalizações temporais do Método TD(λ) com o uso de heurísticas para a aceleração do aprendizado.

Para a seleção das ações é utilizada a estratégia de exploração do ambiente ε – *Greedy* modificada, herdada do algoritmo HAQL, descrita na equação (3.26). Dessa forma, tanto a abordagem de Watkins (1989) como a abordagem de Peng e Williams (1996) podem ser consideradas no novo algoritmo HAQ(λ), modificando apenas a estratégia de exploração do ambiente para considerar a política heurística.

O algoritmo HAQ(λ) implementado nesse trabalho difere do algoritmo Q(λ) descrito na Figura 6.4 apenas pela substituição da estratégia de exploração do ambiente do passo (1), que segue a equação (2.20), pela estratégia de exploração descrita na equação (3.26).

6.3.6 HAQS-learning

O algoritmo HAQS é uma variante do algoritmo Q-learning que faz uso de generalizações espaciais e heurísticas para acelerar o aprendizado. Baseado no algoritmo QS descrito na seção 3.2.2, o HAQS utiliza a estratégia de exploração do ambiente ε – *Greedy* modificada, descrita na equação (3.26). Dessa forma, a política heurística é considerada na seleção das ações do algoritmo HAQS.

7 EXPERIMENTOS E RESULTADOS

Os experimentos foram realizados em duas etapas: na primeira etapa foi feita uma comparação entre os seis algoritmos implementados – Q-learning, Q(λ), QS, HAQL, HAQ(λ) e HAQS – no ambiente de Futebol de Robôs simulado *Simurosot*; na segunda etapa foi realizada a comparação no ambiente real de Futebol de Robôs da categoria *Mirosot*.

Ambas as etapas de experimentos utilizaram a arquitetura do sistema de AR descrita no capítulo 6. A maneira a qual a função de recompensa foi definida determina que o objetivo do agente seja aprender a maximizar o saldo de gols ao longo do tempo. Por esse motivo, os gráficos dos resultados obtidos apresentam os valores acumulados ao longo do tempo.

Todos os experimentos foram efetuados com um agente aprendiz jogando contra um oponente aleatório. Considerando a tupla $\langle x_a, y_a, x_b, y_b, x_o, y_o \rangle$ que define um estado, as 35 regiões que representam o campo de jogo e o conjunto de 7 ações possíveis para o agente, o sistema implementado exige aproximadamente 4 MB de memória RAM.

O tempo consumido por cada iteração dos algoritmos de AR foi menor que 1 milissegundo, rápido o suficiente para não comprometer o desempenho do sistema.

As duas etapas de experimentos são descritas a seguir.

7.1 Experimentos em ambiente simulado *Simurosot*

Os experimentos em ambiente simulado foram realizados com o simulador *Simurosot* em dois microcomputadores Pentium D 3,4 GHz com 1 GB de memória RAM. O aprendizado ocorreu em 5 rodadas de 500 jogos. Cada jogo tem a duração de 5 minutos, independente do número de gols que podem ocorrer durante esse tempo. Cada rodada consumiu, em média, 72 horas de simulação ininterrupta para cada algoritmo de AR, visto que, a cada gol e a cada término de jogo, a simulação parava e era necessário recomeçá-la. Para a automação da configuração do programa servidor do simulador *Simurosot* e reinício dos jogos após um gol ou o término de um jogo, foi utilizado o programa *Automate*, o qual possibilitou que as simulações fossem feitas sem a necessidade de supervisão humana.

O conjunto de ações foi previamente testado e observou-se que, apesar das oscilações apresentadas pelo sistema de controle utilizado, todas as ações cumprem o papel para o qual foram programadas. As ações de movimentação entre as regiões, quando executadas, sempre

deslocam o robô para a região correta, lembrando que, tão logo ocorra uma mudança de região, o estado do ambiente é alterado e uma nova ação é selecionada. A ação "*Get Ball*" sempre é capaz de posicionar o robô atrás da bola, desde que o estado não seja alterado e outra ação não seja selecionada antes de o robô chegar até a posição determinada. No entanto, devido ao não-determinismo do ambiente, a ação "*Kick to Goal*" apresenta probabilidade 0,8 de sucesso ao ser executada, pois o ponto de contato com a bola sofre variações, mesmo quando robô e bola estão sempre na mesma posição antes da execução da ação.

Os gráficos das Figuras 7.1 e 7.2 apresentam as curvas com média e desvio padrão de 5 rodadas dos algoritmos Q-learning e HAQL ao longo de 500 jogos. Esses gráficos apresentam, respectivamente, o número acumulado de gols marcados e o número acumulado de gols sofridos pelo agente.

Pelo gráfico da Figura 7.1 pode-se observar que, ao final de 500 jogos, a utilização da heurística na estratégia de seleção das ações do algoritmo HAQL resultou no aumento do número de gols marcados pelo agente. Esse aumento era esperado, visto que a heurística utilizada indica ao agente a execução da ação "*Kick to Goal*" sempre que estiver na mesma região da bola, o que leva o agente a ter probabilidade 0,8 de marcar um gol.

Já o gráfico da Figura 7.2 mostra uma redução do número de gols sofridos pelo agente do algoritmo HAQL em relação ao algoritmo Q-learning. Apesar de a heurística utilizada estar diretamente relacionada aos gols marcados, como o agente do HAQL marca gols mais rapidamente que o agente do Q-learning, as recompensas recebidas por esses gols são propagadas a outros pares estado-ação e o agente do HAQL aprende a aproximar-se da bola e empurrá-la em direção ao gol adversário antes que o agente do Q-learning aprenda a fazer o mesmo. Dessa forma, mesmo que no HAQL o agente não marque o gol, a bola permanece mais tempo perto do gol adversário e o agente do HAQL sofre menos gols em relação ao agente do Q-learning.

Como a heurística influencia tanto o número de gols marcados, como sofridos, o gráfico da Figura 7.3 apresenta o saldo de gols acumulado ao longo de 500 jogos. Esse gráfico evidencia a influência positiva da heurística. Pode-se observar que o agente do HAQL tem saldo de gols acumulado positivo desde o início do aprendizado, enquanto o agente do Q-learning começa a apresentar um saldo de gols acumulado positivo apenas após 200 jogos.

Os algoritmos QS e HAQS também foram comparados. O gráfico da Figura 7.4 apresenta a média e desvio padrão de 5 rodadas do número acumulado de gols marcados pelo agente ao longo dos 500 jogos. Já o gráfico da Figura 7.5 mostra a média e desvio padrão do número de gols sofridos pelo agente. Novamente, pode-se observar que a heurística influencia positivamente no desempenho do algoritmo HAQS em relação ao algoritmo QS, tanto em relação ao número de gols marcados, como em relação ao número de gols sofridos pelo agente. As observações feitas para os algoritmos Q-learning e HAQL sobre a influência da heurística também podem ser feitas para os algoritmos QS e HAQS.

O gráfico da Figura 7.6 apresenta a média e desvio padrão do saldo de gols acumulado em 5 rodadas, ao longo de 500 jogos dos algoritmos QS e HAQS, onde a influência da heurística se torna mais evidente, assim como no caso dos algoritmos Q-learning e HAQL. No entanto, há uma melhora sutil de desempenho do algoritmo QS em relação ao Q-learning no início do aprendizado, visto que o QS alcança um saldo positivo de gols após 150 jogos.

Ao verificar os gráficos das Figuras 7.3 e 7.6, percebe-se uma semelhança entre os algoritmos Q-learning e QS, assim como HAQL e HAQS. Essa semelhança, também observada em Ribeiro, Pegoraro e Reali-Costa (2002), se deve ao fato de que o espalhamento das experiências, utilizada nos algoritmos QS e HAQS, ocorre apenas no início do aprendizado, onde o agente tem pouca experiência e poucas recompensas significativas (gols marcados e sofridos) recebidas. Como a função de espalhamento decai rapidamente para manter o critério de convergência, os algoritmos QS e HAQS passam a se comportar exatamente como os algoritmos Q-learning e HAQL, respectivamente, em pouco tempo de aprendizado.

Por fim, foram comparados os algoritmos Q(λ) e HAQ(λ). O gráfico da Figura 7.7 apresenta média e desvio padrão do número de gols marcados pelo agente em 5 rodadas desses algoritmos, também ao longo de 500 jogos. A Figura 7.8 apresenta o gráfico de média e desvio padrão para o número de gols sofridos pelo agente. Como nas comparações anteriores, a heurística influenciou positivamente tanto o número de gols marcados, como o número de gols sofridos pelo agente.

O gráfico da Figura 7.9 apresenta média e desvio padrão do saldo de gols acumulado ao longo dos mesmos 500 jogos para os algoritmos Q(λ) e HAQ(λ). Nota-se que o algoritmo Q(λ) supera o desempenho dos algoritmos Q-learning e QS ao acelerar o aprendizado com a propagação das experiências por generalizações temporais. No entanto, o Q(λ) não é capaz de

superar os algoritmos HAQL e HAQS que utilizam a aceleração do aprendizado utilizando conhecimento sobre o domínio através de heurísticas.

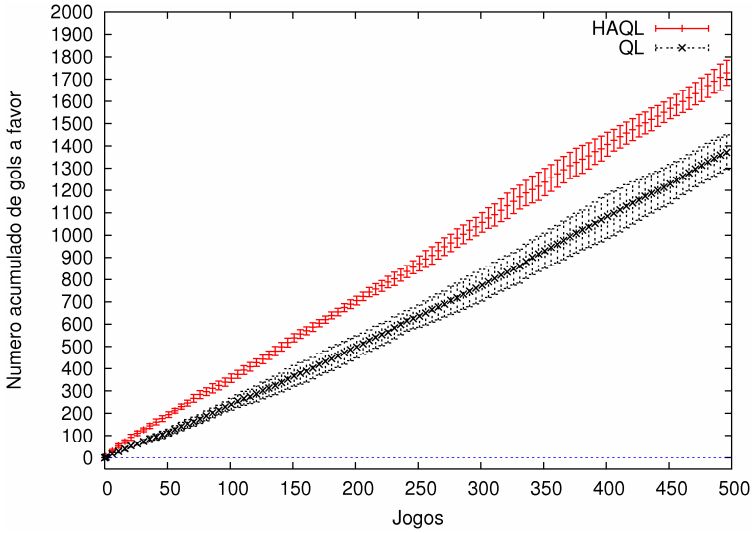

Figura 7.1 – Número acumulado de gols a favor em 500 jogos dos algoritmos Q-learning e HAQL

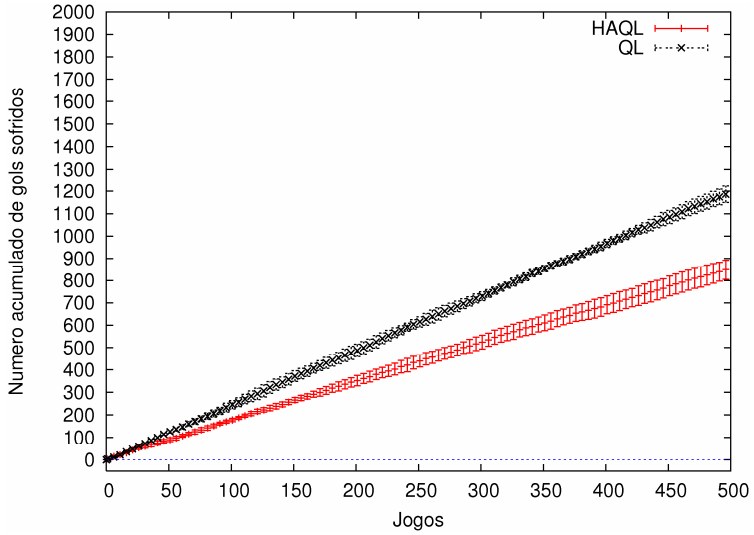

Figura 7.2 – Número acumulado de gols sofridos em 500 jogos dos algoritmos Q-learning e HAQL

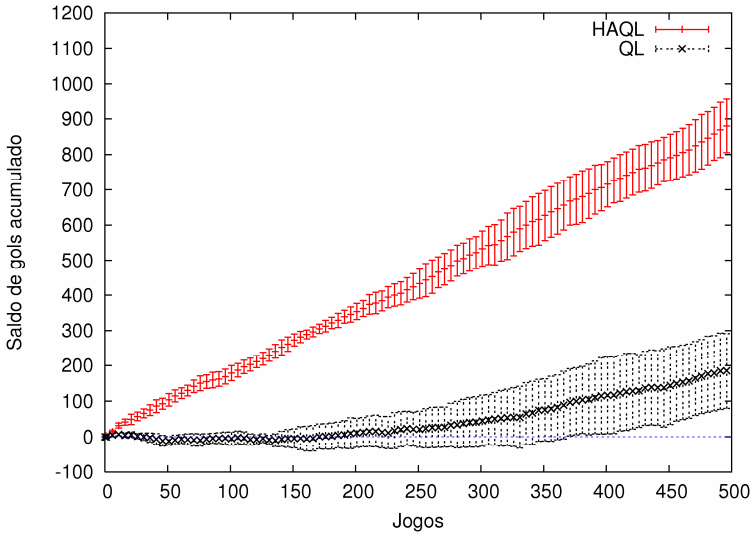

Figura 7.3 – Saldo de gols acumulado em 500 jogos dos algoritmos Q-learning e HAQL

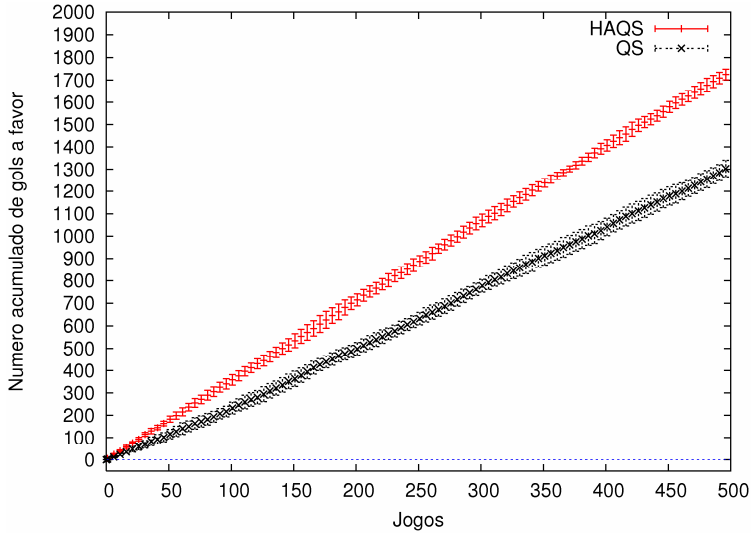

Figura 7.4 – Número acumulado de gols a favor em 500 jogos dos algoritmos QS e HAQS

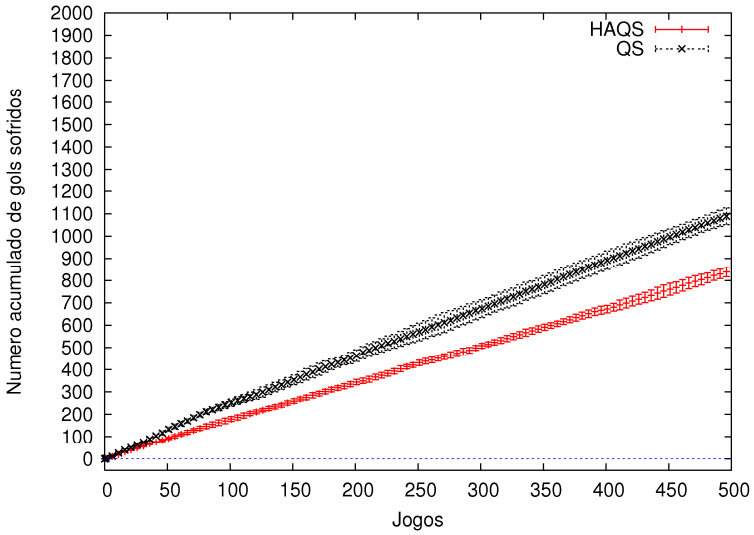

Figura 7.5 – Número acumulado de gols sofridos em 500 jogos dos algoritmos QS e HAQS

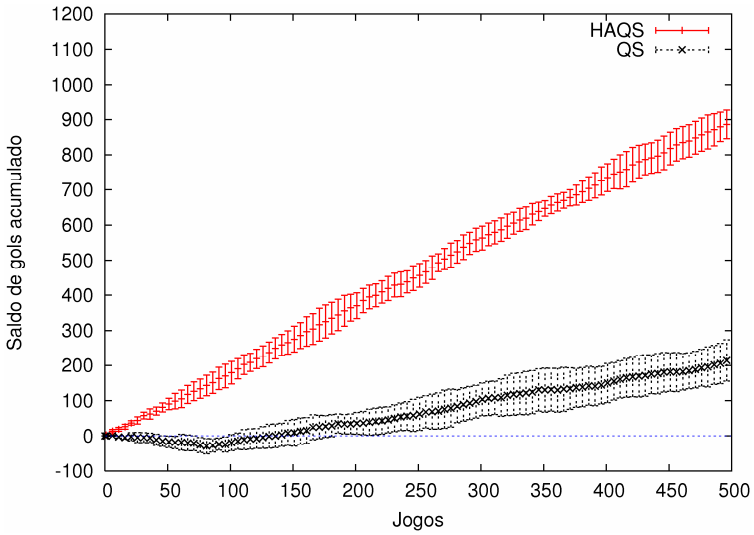

Figura 7.6 – Saldo de gols acumulado em 500 jogos dos algoritmos QS e HAQS

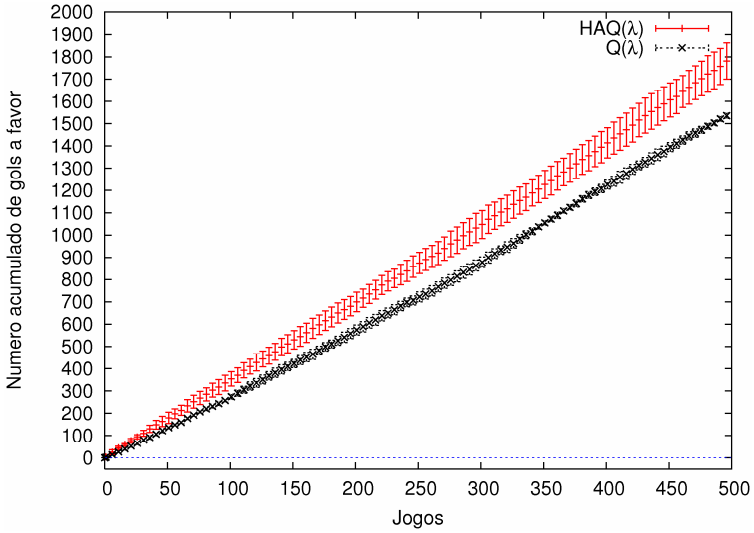

Figura 7.7 – Número acumulado de gols a favor em 500 jogos dos algoritmos Q(λ) e HAQ(λ)

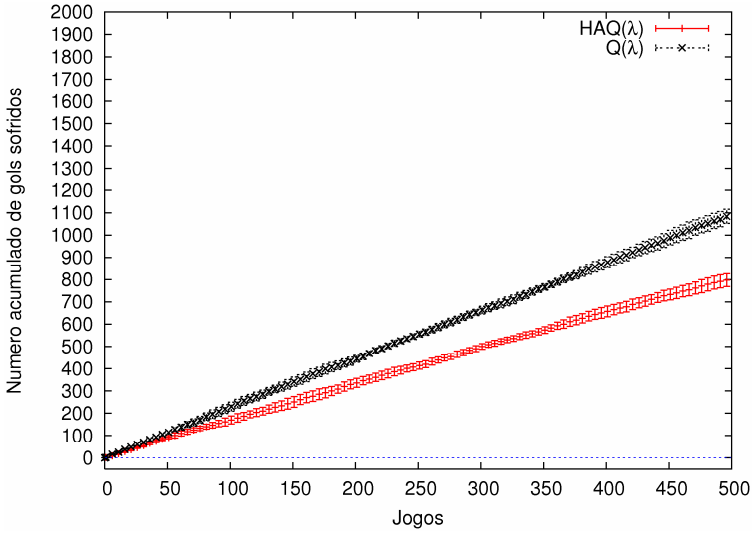

Figura 7.8 – Número acumulado de gols sofridos em 500 jogos dos algoritmos Q(λ) e HAQ(λ)

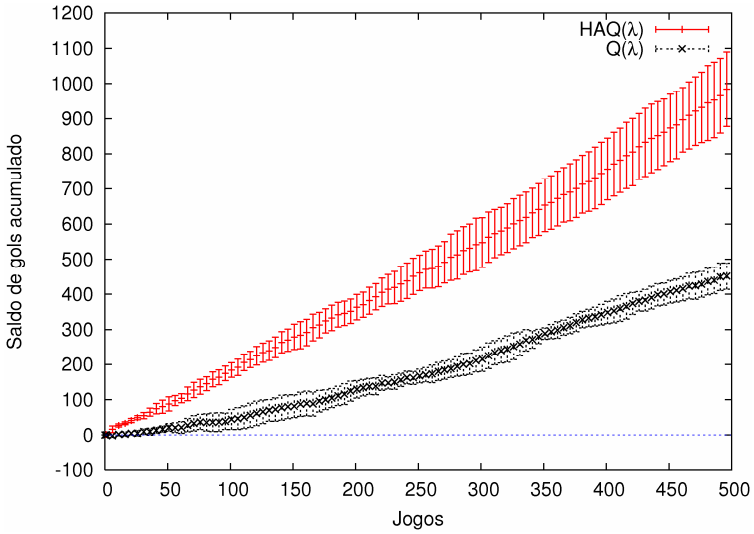

Figura 7.9 – Saldo de gols acumulado em 500 jogos dos algoritmos Q(λ) e HAQ(λ)

Duas observações importantes podem ser feitas para todos os experimentos simulados efetuados, de acordo com os resultados apresentados nos gráficos das Figuras 7.1 a 7.9.

A primeira observação é que a utilização de heurísticas melhora significativamente o desempenho dos algoritmos, resultando em um acúmulo positivo no saldo de gols desde o início do aprendizado, enquanto os algoritmos que não utilizam heurísticas apresentam acúmulo negativo no saldo de gols no início do aprendizado. A heurística é capaz de direcionar o agente ao seu objetivo quando o mesmo não tem experiência suficiente, enquanto sem a heurística o agente acaba por ter um comportamento excessivamente exploratório no início do aprendizado.

A segunda observação é que, aproximando-se por retas as curvas dos algoritmos dos gráficos das Figuras 7.3, 7.6 e 7.9, nota-se uma inclinação maior nas retas que correspondem aos algoritmos que utilizam heurísticas, mostrando que o saldo de gols aumenta a uma taxa maior quando heurísticas são utilizadas, confirmando a aceleração do aprendizado. A aceleração do aprendizado também pode ser verificada da mesma maneira para os outros gráficos apresentados, sendo que o número de gols marcados aumenta a uma taxa maior nos algoritmos que utilizam heurísticas e o número de gols sofridos aumenta a uma taxa menor quando a heurística é utilizada.

Em relação às generalizações temporais em comparação com as generalizações espaciais, um ponto a ser considerado é o fato de que os algoritmos Q(λ) e HAQ(λ) espalham a recompensa recebida por um rastro dos últimos pares estado-ação visitados, representando uma seqüência de diferentes posições dos jogadores e da bola e diferentes ações (diferentes experiências) que visam maximizar a recompensa. Já os algoritmos QS e HAQS espalham uma mesma posição do agente e da bola e uma mesma ação selecionada pelo agente (apenas uma experiência) por estados cuja posição do oponente tenha alguma similaridade com sua posição original.

Dessa forma, a propagação das recompensas descontadas ao longo do tempo nos algoritmos QS e HAQS será muito mais próxima à propagação que ocorre nos algoritmos Q-learning e HAQL. Esse fato também justifica a semelhança entre os algoritmos Q-learning e QS, assim como entre os algoritmos HAQL e HAQS, além de justificar o motivo pelo qual o algoritmo Q(λ) supera em eficiência os algoritmos Q-learning, QS e o algoritmo HAQ(λ) também supera os algoritmos HAQL e HAQS, conforme pôde ser observado nos gráficos apresentados.

7.2 Experimentos em ambiente real *Mirosot*

Os experimentos em ambiente real utilizaram todo o sistema real do time de Futebol de Robôs da categoria *Mirosot*, descrito no capítulo 5. Foi utilizado um computador Pentium 4 *Hyper Threading* 3,2 GHz, com 512 MB de memória RAM, equipado com um cartão PCMCIA de captura de vídeo, capaz de capturar imagens a 30 quadros por segundo. O sistema de controle utilizado com os robôs reais foi o mesmo utilizado no ambiente simulado.

Cada jogo também tem 5 minutos de duração, no entanto, os experimentos em ambiente real exigiram muito mais tempo se comparados aos experimentos em ambiente simulado, pois era necessário posicionar a bola no centro do campo de jogo a cada gol e posicionar os robôs a cada início de jogo, manualmente. Logo, foi necessária supervisão humana durante todos os jogos em ambiente real.

Os algoritmos utilizados nos experimentos em ambiente real foram o Q(λ) e HAQ(λ), pois foram os algoritmos que apresentaram o melhor desempenho em ambiente simulado sem a utilização de heurística e com o uso da mesma. Foram feitas duas rodadas de 100 jogos para o algoritmo Q(λ) em ambiente real. O gráfico da Figura 7.10 apresenta os resultados de média e desvio padrão dessas duas rodadas, comparando com os resultados obtidos para o algoritmo Q(λ) em 5 rodadas em ambiente simulado.

Apesar de a curva do algoritmo Q(λ) em ambiente real apresentar uma média de apenas duas rodadas cujo desempenho é ligeiramente superior à curva do algoritmo Q(λ) em ambiente simulado, não é possível assegurar que o desempenho do algoritmo Q(λ) seja melhor em ambiente simulado, ou ainda, que seja melhor em ambiente real.

No entanto, verificando-se os gráficos das Figuras 7.11 e 7.12, que representam, respectivamente, o número acumulado de gols marcados e o número acumulado de gols sofridos pelo agente, ao longo de 100 jogos, pode-se observar que houve maior ocorrência de gols, tanto marcados como sofridos, no ambiente real. Esse aumento no número de gols pode ser justificado pelo fato de que o ambiente simulado modela um campo de jogo com dimensões de 220 x 180 cm, enquanto o ambiente real é composto por um campo de jogo com dimensões de 150 x 130 cm. Dessa forma, as dimensões reduzidas do campo em ambiente real aumentam as chances de ocorrência de gols, tanto a favor, como contra o agente.

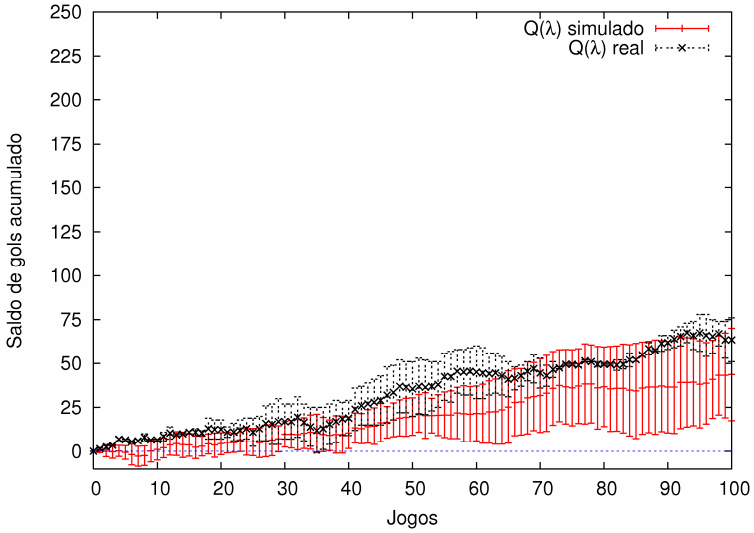

Figura 7.10 – Saldo de gols acumulado em 100 jogos do algoritmo Q(λ) em ambiente simulado e real

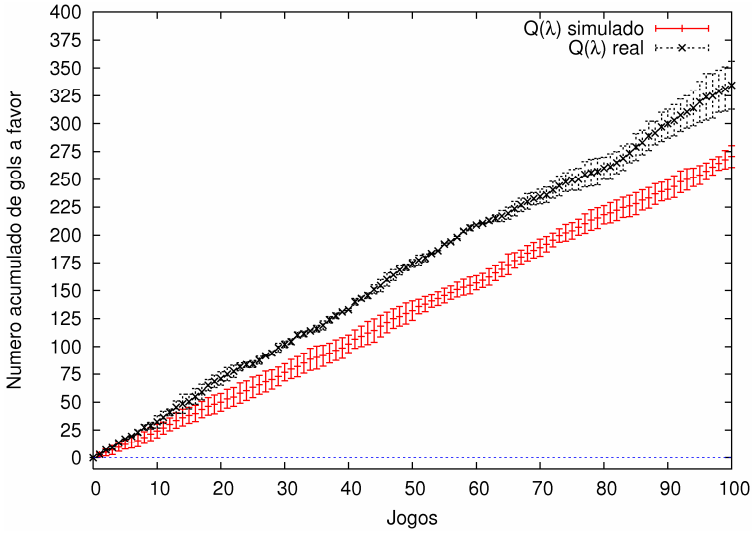

Figura 7.11 – Número acumulado de gols a favor em 100 jogos do Q(λ) em ambiente simulado e real

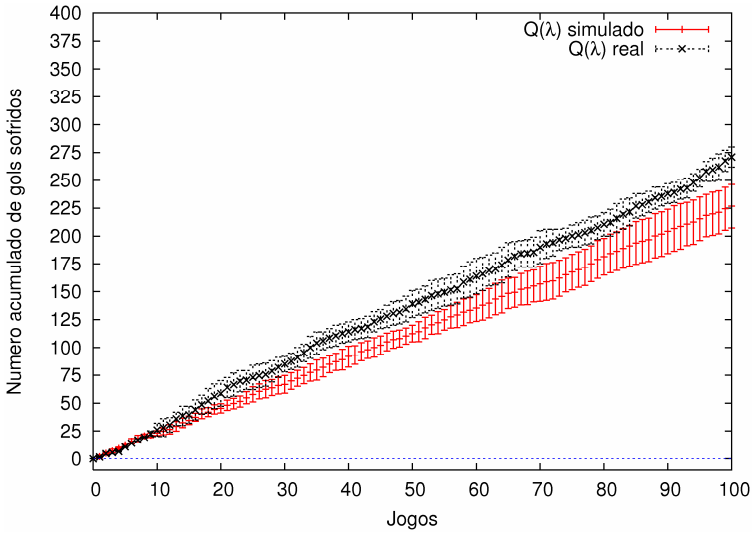

Figura 7.12 – Número acumulado de gols sofridos em 100 jogos do Q(λ) em ambiente simulado e real

7.2.1 Transferência de conhecimento através de heurísticas

Diferentemente de técnicas que utilizam a transferência de conhecimento adquirido em AR através da função valor, como as descritas em Banerjee e Stone (2007) e Taylor e Stone (2005), para a comparação do algoritmo HAQ(λ) com os resultados do algoritmo Q(λ) em ambiente real, foram definidas heurísticas a partir da política aprendida pelo agente em ambiente simulado. Dessa forma, o conhecimento adquirido pelo agente em ambiente simulado é transferido através de heurísticas para o ambiente real.

A heurística em ambiente simulado era fixa e definia que a ação "*Kick to Goal*" deveria ser executada quando o agente estivesse na mesma região da bola. Para o ambiente real, a função heurística é definida conforme a regra da equação (7.1):

$$H(s_t, a_t) = \begin{cases} 100 & se\ a_t = \max_a Q^{Sim}(s_t, a) \\ 0 & caso\ contrário \end{cases} \tag{7.1}$$

Onde Q^{Sim} é a função ação-valor aprendida após 500 jogos do algoritmo HAQ(λ) em ambiente simulado. Então, no ambiente real, para cada estado possível do ambiente, a função heurística sugere a ação que resultaria no maior valor acumulado de recompensas caso o agente estivesse em ambiente simulado, conforme a função ação-valor aprendida após 500 jogos de simulação.

Com a heurística definida para o ambiente real, foram efetuadas duas rodadas de experimentos do algoritmo HAQ(λ) com 100 jogos cada uma. Pode-se verificar, pelo gráfico da Figura 7.13, uma queda de desempenho significativa quando o algoritmo HAQ(λ) é utilizado em ambiente real.

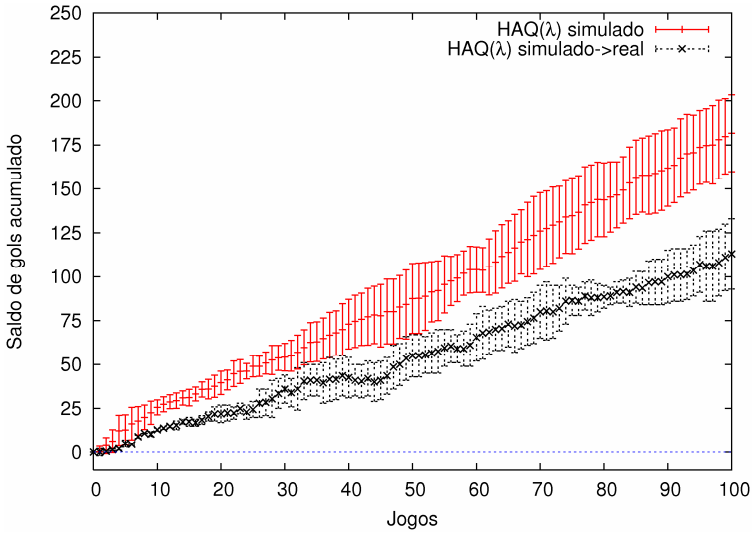

Figura 7.13 – Saldo de gols acumulado em 100 jogos do algoritmo HAQ(λ) em ambientes simulado e real

O gráfico da Figura 7.14, que representa o número acumulado de gols marcados pelo agente. Em contraste com o resultado do algoritmo Q(λ) apresentado no gráfico da Figura 7.11 onde o número de gols marcados no ambiente real aumentou, o agente do HAQ(λ) em ambiente real marcou, em média, um número menor quando comparado ao resultado do ambiente simulado. Já o gráfico da Figura 7.15 mostra que a média do número de gols sofridos pelo agente em ambiente real aumentou em relação ao ambiente simulado, resultado que era esperado, assim como havia acontecido com o algoritmo Q(λ) em ambiente real, cujo resultado foi apresentado no gráfico da Figura 7.12.

Então, é possível justificar a queda de desempenho do algoritmo HAQ(λ) em ambiente real pela influência da heurística. Essa influência foi mais positiva em ambiente simulado.

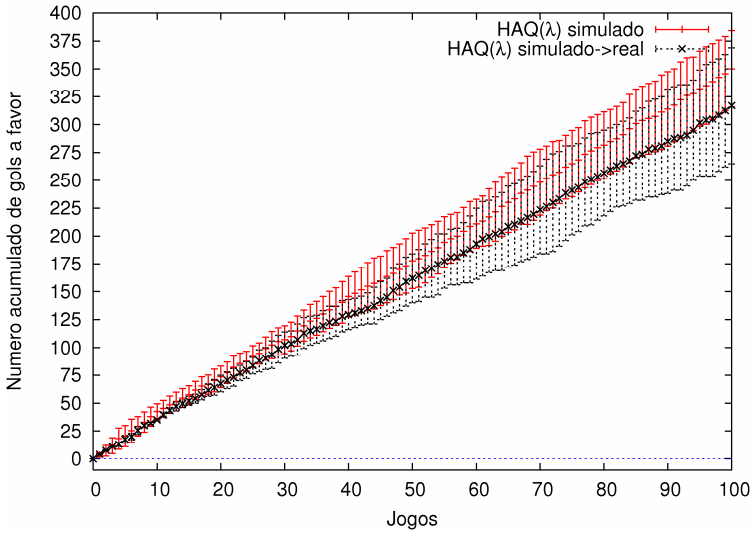

Figura 7.14 – Número acumulado de gols a favor em 100 jogos do HAQ(λ) em ambientes simulado e real

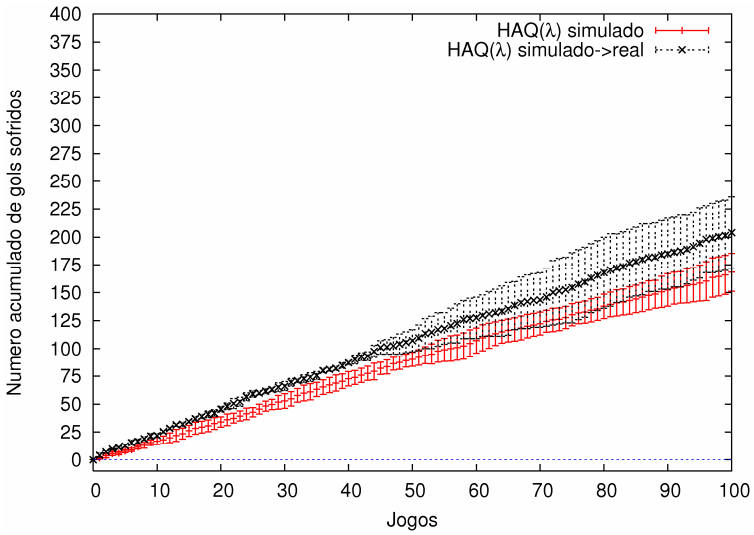

Figura 7.15 – Número acumulado de gols sofridos em 100 jogos do HAQ(λ) em ambientes simulado e real

Buscando o motivo pelo qual a heurística utilizada em ambiente real não resultou em um aumento do número de gols marcados em relação ao ambiente simulado, foi feita uma rodada de 100 jogos para o algoritmo HAQ(λ) fazendo uso da mesma heurística utilizada em ambiente simulado, definida conforme descrito na seção 6.2.4. O gráfico da Figura 7.16 apresenta a comparação entre o algoritmo HAQ(λ) com heurísticas definidas através do conhecimento adquirido em ambiente simulado e o algoritmo HAQ(λ) com a mesma heurística utilizada em ambiente simulado. Através dessa comparação, pode-se concluir que ambas as heurísticas influenciam de forma muito similar o comportamento do agente, não sendo possível afirmar que existe uma diferença considerável de desempenho entre os algoritmos.

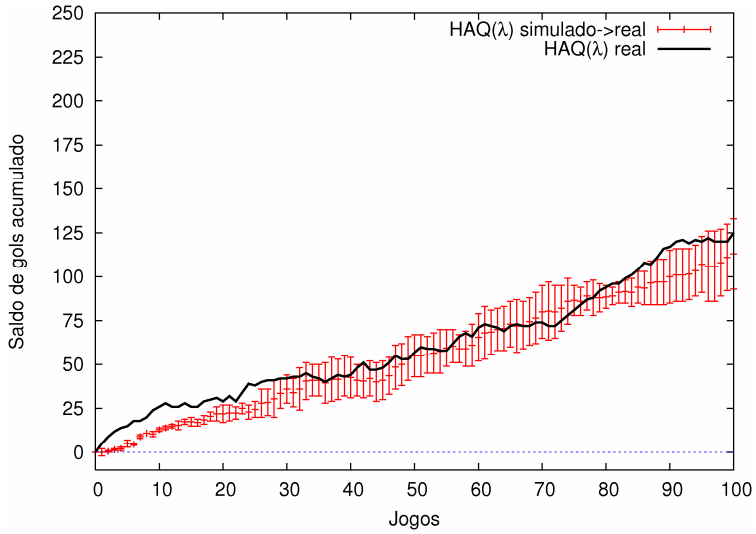

Figura 7.16 – Saldo de gols acumulado em 100 jogos do algoritmo HAQ(λ) com diferentes heurísticas

Em seguida, foram executados alguns testes do conjunto de ações em ambiente real para verificar o resultado da execução das ações em ambiente real. Em relação ao ambiente simulado, a única ação que apresentou resultados consideravelmente diferentes foi a ação "*Kick to Goal*". Enquanto no ambiente simulado a execução dessa ação apresentava probabilidade 0,8 de resultar em gol, no ambiente real observou-se que a probabilidade caiu para 0,3. Logo, o número de gols marcados pelo agente do HAQ(λ) em ambiente real não foi superior em relação ao agente do HAQ(λ) em ambiente simulado porque a ação sugerida pela heurística, que deveria resultar em gol com alta probabilidade, quando executada não resultava em gol. Dessa forma, observou-se que o agente aprendeu a superar a heurística e marcar gols utilizando outras ações para tal, fazendo uso da ação "*Kick to Goal*" apenas quando estava muito perto do gol.

No entanto, mesmo com o problema descrito sobre a heurística, o agente do HAQ(λ) superou em desempenho o algoritmo Q(λ) em ambiente real, o que confirma que, por menor que seja a probabilidade da ação "*Kick to Goal*", utilizar uma heurística que sugira essa ação ao agente influencia positivamente no desempenho do algoritmo, conforme pode ser visto na comparação dos algoritmos HAQ(λ) e Q(λ) em ambiente real, mostrada no gráfico da Figura 7.17. Pode-se observar, ainda, que a inclinação das retas aproximadas das curvas de aprendizado indica que a heurística acelera o aprendizado, mesmo quando a ação sugerida pela heurística não resulta em gol com alta probabilidade.

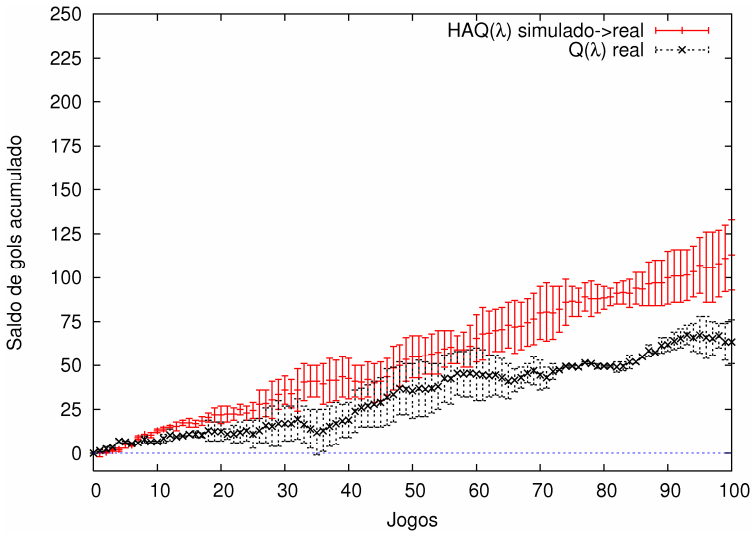

Figura 7.17 – Saldo de gols acumulado em 100 jogos dos algoritmos Q(λ) e HAQ(λ) em ambiente real

Esse capítulo apresentou os experimentos realizados em ambiente simulado e ambiente real de Futebol de Robôs, juntamente com análises críticas sobre os resultados, justificando o comportamento dos agentes nos diferentes ambientes e com diferentes algoritmos de AR. Esses testes mostraram que a utilização de heurísticas para a aceleração do AR contribui positivamente tanto em ambiente simulado como em ambiente real.

8 CONCLUSÃO E TRABALHOS FUTUROS

Esse trabalho apresentou a comparação de algoritmos de AR com e sem a utilização de heurísticas para aceleração do aprendizado em ambientes real e simulado de Futebol de Robôs. Foram comparados algoritmos que fazem uso de generalizações temporais e espaciais e foi discutida uma abordagem de abstração do conjunto de estados e ações para a definição da arquitetura do sistema de AR.

Os experimentos em ambiente simulado *Simurosot* tinham o objetivo de estender o AR para um ambiente complexo, dinâmico e não-determinístico, em contraste com o ambiente simplificado de Futebol de Robôs proposto por Littman (1994), conforme sugerido por Bianchi (2004) como uma das extensões de seu trabalho. Os resultados em ambiente simulado foram satisfatórios e apresentaram o comportamento esperado do agente de AR, confirmando a eficiência da utilização de heurísticas para aceleração do AR.

Já os experimentos em ambiente real tinham o objetivo de testar a eficiência da utilização de heurísticas em algoritmos de AR em robôs reais, além de verificar os resultados da transferência de conhecimento adquirido em ambiente simulado para o ambiente real através de heurísticas.

Quando testados em ambiente real, os algoritmos apresentaram um bom resultado, também demonstrando a influência positiva da utilização de heurísticas para aceleração do aprendizado, mesmo em um ambiente que apresentou tantas adversidades, as quais não estão presentes no ambiente simulado e foram propositalmente mantidas para que as condições do ambiente permanecessem distantes das condições ideais, tornando os resultados obtidos mais relevantes.

Tais adversidades começam no sistema de visão computacional que, apesar de ter apresentado resultados excelentes, possui um erro de precisão na determinação de posição e orientação dos robôs em relação ao campo, visto que a iluminação utilizada não era uniforme e os cantos do campo de jogo apresentavam menos luminosidade, além de ruídos intrínsecos a sensores, como a câmera de vídeo utilizada para capturar a imagem. Pela pequena imprecisão do sistema de visão computacional e os ruídos na imagem proveniente da câmera, o ambiente real apresentou, em algumas poucas situações, um problema quando a bola parava na fronteira entre duas regiões. O sistema de visão computacional determinava a posição da bola ora em uma região ora na região de fronteira. Essa oscilação de posição da bola causava uma

constante mudança de estados do ambiente, fazendo com que ambos agente e oponente selecionassem outras ações a executar a cada mudança. No entanto, após alguns segundos, um dos robôs conseguia chegar até a bola e tirá-la da região de fronteira.

O campo de jogo utilizado nos experimentos em ambiente real é irregular, com algumas saliências e pequenos buracos. Esse campo de jogo também apresenta ligeiras inclinações, o que alterava a trajetória da bola constantemente.

Outra adversidade encontrada no ambiente real foi o atrito com o campo, que variava de acordo com o acúmulo de poeira e com o desgaste da borracha das rodas dos robôs, robôs os quais são dotados de motores que não são idênticos, tampouco ideais e cujas baterias descarregam ao longo dos jogos.

No entanto, mesmo com as adversidades citadas, características de um ambiente real, os resultados obtidos em ambiente real poderiam ter sido melhores se o ruído da câmera, a falta de odometria nos robôs e, principalmente, o sistema de controle não tivessem comprometido o desempenho do sistema de AR. A queda da probabilidade da ação "*Kick to Goal*" resultar em gol de 0,8 para 0,3 é prova da influência negativa do sistema de controle utilizado.

8.1 Contribuições

Além do objetivo de comparar algoritmos de AR com e sem a aceleração por heurística em ambientes real e simulado, esse trabalho apresenta outras contribuições, descritas a seguir.

Os robôs móveis construídos para esse trabalho seguem rigorosamente as regras impostas pela regra da categoria *Mirosot*. Os robôs são uma contribuição para a equipe de Futebol de Robôs do Centro Universitário da FEI, que já os tem utilizado em competições, além de servir como plataforma de testes, podendo ter seu projeto melhorado sem a necessidade de se desenvolver um novo robô a partir do zero.

O sistema de visão computacional desenvolvido para esse trabalho é outra contribuição importante para a equipe de Futebol de Robôs do Centro Universitário da FEI, que também vem utilizando esse sistema em competições e como plataforma de testes e desenvolvimento de sistemas de estratégia, navegação e controle do time. Esse sistema de

visão computacional também gerou resultados relevantes o suficiente para a publicação dos artigos Martins, Tonidandel e Bianchi (2006a) e Martins, Tonidandel e Bianchi (2006b).

Foram realizados experimentos em um ambiente simulado mais complexo, dinâmico e não-determinístico, conforme sugerido por Bianchi (2004) como extensão de seu trabalho. Também foram realizados experimentos em um ambiente real, com robôs reais, também sugeridos por Bianchi (2004) como extensão de seu trabalho, permitindo verificar a generalidade e a eficiência da aceleração do AR por heurísticas em um ambiente real, abordagem que, até então, não havia sido feita.

Foram propostos, implementados e testados os dois novos algoritmos HAQ(λ) e HAQS, ainda inéditos na literatura de AR.

Foi abordada a possibilidade de transferência de conhecimento em AR através de heurísticas para acelerar o aprendizado. Dessa forma, não é necessário que robôs reais sejam utilizados para o aprendizado. O aprendizado pode ocorrer em ambiente simulado e o conhecimento adquirido pode ser transferido para o ambiente real através de heurísticas, sem comprometer o aprendizado em ambiente real, caso os ambientes apresentem variações que resultem em aprendizados de diferentes comportamentos para cada ambiente, pois as heurísticas são desconsideradas caso não direcionem o agente ao objetivo.

8.2 Trabalhos futuros

Durante o desenvolvimento desse trabalho, surgiram diversos tópicos e questões que foram abordados de forma superficial e que merecem estudos mais aprofundados. Entre os mais relevantes, são sugeridos alguns trabalhos futuros a seguir.

A primeira sugestão é a utilização de algoritmos de AR com abordagem multi-agentes, como o Minimax-Q, proposto por Littman (1994), o HAMMQ, proposto por Bianchi, Ribeiro e Reali-Costa (2007) e o Minimax-QS, proposto por Ribeiro, Pegoraro e Reali-Costa (2002), para a solução do AR no Futebol de Robôs.

Um estudo sobre a definição da arquitetura do sistema de AR também pode ser feito para verificar a possibilidade de considerar outras informações como variáveis de estado, como a distância do agente até a bola, ou ainda, a direção de deslocamento da bola, por exemplo. Outras abordagens para decaimento da taxa de aprendizado α e um refinamento das

macro-ações que compõem o conjunto de ações, incluindo o desenvolvimento de um sistema de controle estável e mais preciso também são pontos a serem analisados.

Em relação aos algoritmos HAQ(λ) e HAQS propostos nesse trabalho, as generalizações espaciais e temporais podem ser estudadas em mais detalhes. O espalhamento espacial das experiências pode ser melhor definido de acordo com informações sobre o domínio, auxiliando o aprendizado. A propagação das experiências ao longo do tempo também pode ser mais estudada, verificando a influência de outros valores de λ quando utilizados com a aceleração por heurísticas.

As heurísticas provenientes do conhecimento adquirido em simulação podem ser definidas pela normalização dos valores da função ação-valor Q, ao invés de apresentar um único valor apenas para a ação que maximiza o ganho de recompensas acumuladas ao longo do tempo.

Finalmente, também pode ser analisado o comportamento de um agente quando o conhecimento adquirido em um ambiente, real ou simulado, com determinados conjuntos de estados e ações, é transferido, através de heurísticas, para um mesmo ambiente com conjuntos de estados e ações expandidos. Dessa forma, pode-se aproveitar o conhecimento prévio adquirido pelo agente e expandir o conjunto de estados, inserindo outras informações relevantes sobre o ambiente nas variáveis de estado. Pode-se, ainda, expandir o conjunto de ações e possibilitar que o agente aprenda novos comportamentos naquele ambiente, agregando conhecimento, ao invés de ter de adquirir todo esse conhecimento novamente.

REFERÊNCIAS BIBLIOGRÁFICAS

AZEVEDO, E.; CONCI, A. Computação Gráfica. Rio de Janeiro: Elsevier, 2003.

BANERJEE, B.; STONE, P. General Game Learning using Knowledge Transfer. In *Proceedings of the 20th International Joint Conference of Artificial Intelligence (IJCAI)*, p. 672-677, 2007.

BIANCHI, R. A. C. **Uso de Heurísticas para a Aceleração do Aprendizado por Reforço.** 2004. 174 f. Tese (Doutorado em Engenharia) – Escola Politécnica, Universidade de São Paulo, São Paulo.

BIANCHI, R. A. C.; REALI-COSTA, A. H. O Sistema de Visão Computacional do Time FutePOLI de Futebol de Robôs. In: CBA 2000 - CONGRESSO BRASILEIRO DE AUTOMÁTICA, 13., Florianópolis, 2000. Anais. Florianópolis, Sociedade Brasileira de Automática, 2000. p.2156-2161, 2000.

BIANCHI, R. A. C.; PENHARBEL, E. A.; DESTRO, R. C.; TONIDANDEL, F. Filtro de Imagem Baseado em Matriz RGB de Cores-Padrão para Futebol de Robôs. In: I Encontro de Robótica Inteligente, 2004, Salvador-BA. Anais I Encontro de Robótica Inteligente (EnRI´04), 2004.

BIANCHI, R. A. C.; RIBEIRO, C. H. C.; REALI-COSTA, A. H. Heuristic Selection of Actions in Multiagent Reinforcement Learning. In *Proceedings of the 20th International Joint Conference of Artificial Intelligence (IJCAI)*, p. 690-695, 2007.

BRESENHAM, J. E. Algorithm for Computer Control of A Digital Plotter. **IBM Systems Journal,** v. 4, n. 1, p.25-30, 1965.

CANNY, J. A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, v.8, n. 6, p.679-698, nov. 1986.

DAYAN, P. The Convergence of TD(λ) for general λ. **Machine Learning**, v. 8, n. 3-4, p. 341-362, 1992.

EGLY, U.; NOVAK, G.; WEBER, D. Decision Making for Mirosot Soccer Playing Robots. In: CLAWAR/EURON WORKSHOP ON ROBOTS IN ENTERTAINMENT, LEISURE AND HOBBY, 2004, Viena, Áustria.

FARIA, G.; TEIZEN, L.; ROMERO, R. A. F. Controle de trajetórias utilizando campos potenciais aplicado ao futebol de robôs. In: I Encontro de Robótica Inteligente, 2004, Salvador-BA. Anais I Encontro de Robótica Inteligente (EnRI´04), 2004.

FIRA. Federation of International Robot-Soccer Association. Disponível em: <http://www.fira.net>. Acesso em: 9 abril 2007.

FORSYTH, D. A.; PONCE, J. Computer Vision: A Modern Approach. New Jersey: Prentice Hall, 2003.

FTB. FTB Miniature Drive Systems. Disponível em: <http://www.micromo.com>. Acesso em: 9 abril 2007.

GONNER, C.; ROUS, M.; KRAISS, K. Real-Time Adaptative Colour Segmentation for the Robocup Middle Size League. **Lecture notes in Artificial Inteligence**, v. 1856, p.243-256, 2000.

GRITTANI, G.; GALLINELLI, G.; RAMÍREZ, J. FutBot: A Vision System for Robotic Soccer. **Lecture notes in Artificial Intelligence**, v. 1952, p.350-358, nov.2000.

GUIL, N.; ZAPATA, E. L. Lower Order Circle and Ellipse Hough Transform. **Journal of Pattern Recognition**, v. 30, n. 10, p.1729-1744, out. 1997.

HAN, K.; LEE, K.; MOON, C.; LEE, H.; Kim, J. Robot Soccer System of SOTY 5 for Middle League MiroSot. FIRA Robot Congress. 2002.

HEARN, D.; BAKER, M. P. Computer Graphics: C version. 2. ed. New Jersey: Prentice Hall, 1997.

IBM. IBM's DeepBlue Supercomputer. Disponível em: <http://www.research.ibm.com/deepblue>. Acesso em: 9 abril 2007.

INTEL. OpenCV – Open Source Computer Vision Library. Disponível em: <http://www.intel.com/technology/computing/opencv/index.htm>. Acesso em: 9 abril 2007.

KAELBLING, L.P.; LITTMAN M. L.; MOORE, W.A. Reinforcement Learning: A Survey. **Journal of Artificial Intelligence Research**, n. 4, p. 237-285, maio 1996.

KIM, D.; PARK, J.; KIM, J. Limit-cycle Navigation Method for Soccer Robot. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE, ROBOTICS AND AUTONOMOUS SYSTEMS, 2001, Singapura. 2001.

LITTMAN, M. L. Markov games as a framework for multi-agent reinforcement learning. In: *Proceedings of the 11th International Conference on Machine Learning (ICML'94)*, p. 157-163, 1994.

MACKWORTH, A. K. On Seeing Robots. *Vision Interface*, 1992. Disponível em: <citeseer.ist.psu.edu/mackworth92seeing.html>. Acesso em 28 set. 2006.

MARTINS, M. F.; TONIDANDEL, F.; BIANCHI, R. A. C. Reconhecimento de Objetos em Tempo Real para Futebol de Robôs. In: JORNADA DE ROBÓTICA INTELIGENTE – ENCONTRO NACIONAL DE ROBÓTICA INTELIGENTE, 2006, Campo Grande. Anais do XXVI Congresso da Sociedade Brasileira de Computação. Campo Grande: Sociedade Brasileira de Computação. p. 173-182, 2006.

MARTINS, M. F.; TONIDANDEL, F.; BIANCHI, R. A. C. A Fast Model Based Vision System for a Robot Soccer Team. Lecture Notes in Computer Science, v. 4293, p. 704-714, 2006.

MARTINS, M. F.; TONIDANDEL, F.; BIANCHI, R. A. C. Um protocolo confiável e flexível de comunicação para Futebol de Robôs. In: IV Latin American IEEE Student Robotics Competition, 2005, São Luis. Anais do VII Simpósio Brasileiro de Automação Inteligente / II Latin American Robotics Symposium, 2005.

MARTINS, M. F. Desenvolvimento de Um Novo Protótipo para o Time de Futebol de Robôs da FEI - RoboFEI. Disponível em: <http://www.fei.edu.br/robo/publicacao.htm>. Acesso em 9 abril 2007.

MCLAUGHLIN, R. A.; ALDER, M. D. The Hough Transform versus the UpWrite. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 20, n. 4, p.396-400, 1998.

MICROCHIP. Microchip Technology Inc. Disponível em: <http://www.microchip.com>. Acesso em: 9 abril 2007.

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

NASA. NASA's Mars Exploration Program. Disponível em: <http://mpfwww.jpl.nasa.gov>. Acesso em 9 abril 2007.

NOVAK, G.; SEYR, M. Simple Path Planning Algorithm for Two-Wheeled Differentially Driven (2WDD) Soccer Robots. In: PROCEEDINGS OF THE SECOND WORKSHOP ON INTELLIGENT SOLUTIONS IN EMBEEDED SYSTEMS, WISES 2004, 2., 2004, Graz, Áustria. 2004.p. 91-102.

NOVAK, G. Roby-Go, a Prototype for Several MiroSot Soccer Playing Robots. In: PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON COMPUTATIONAL CYBERNETICS ICCC'04, Viena, Áustria. 2004.

NOVAK, G.; MAHLKNECHT, S. TINYPHOON – A Tiny Autonomous Mobile Robot. In: PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS, 2005, Dubrovnik, Croácia. 2005.

PENG, J.; WILLIAMS, R. J. Incremental Multi-Step Q-Learning. **Machine Learning**, v. 22, p. 283-290, 1996.

PICCARDI, M. Background Subtraction Techniques: A Review. In: PROCEEDINGS OF IEEE SMC 2004 INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, 2004, The Hague, Netherlands.

PISTORI, H.; PISTORI, J.; COSTA, E. R. Hough-Circles: Um Módulo de Detecção de Circunferências para o ImageJ. In: 6o. WORKSHOP SOFTWARE LIVRE 2005, Porto Alegre, 2005.

RIBEIRO, C. H. C. A Tutorial on Reinforcement Learning Techniques. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 1999. Washington, DC: INNS Press, 1999.

RIBEIRO, C. H. C.; SZEPESVÁRI, C. Q-Learning combined with spreading: Convergence and results. In *Proceedings of the ISRF-IEE International Conference on Intelligent and Cognitive Systems (Neural Networks Symposium)*, p. 32-36, 1996.

RIBEIRO, C. H. C. ; COSTA, Anna Helena Reali ; PEGORARO, R. . Experience Generalization for Concurrent Reinforcement Learners: the Minimax-QS Algorithm. In: First International Joint Conference on Autonomous Agents and Multiagent Systems - AAMAS2002, 2002, Bologna. Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems. New York : ACM Press, 2002. v. 3. p. 1239-1245, 2002.

ROBOCUP. Robot World Cup Initiative. Disponível em: <http://www.robocup.org>. Acesso em 9 abril 2007.

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

SEYR, M.; JAKUBEK, S. Mobile Robot Predictive Trajectory Tracking. In: SECOND INTERNATIONAL CONFERENCE ON INFORMATICS IN CONTROL, AUTOMATION AND ROBOTICS, 2., 2005, Barcelona, Espanha. 2005.

SEYR, M.; JAKUBEK, S.; NOVAK, G. Neural Network Predictive Trajectory Tracking of an Autonomous Two-Wheeled Mobile Robot. In: PROCEEDINGS OF THE 16TH IFAC WORLD CONGRESS, 16., 2005, Praga, República Checa. Elsevier, 2005.

STONE, P. **Layered Learning in Multi-Agent Systems.** 1998. 253 f. Tese (Doutorado em Ciência da Computação) – School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

STRICKER, M.; LEONARDIS, A. From Edgels to Parametric Curves. In: PROCEEDINGS OF THE 9TH SCANDINAVIAN CONFERENCE ON IMAGE ANALYSIS, 9., 1995., Uppsala. Suécia, 1995.

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

SUTTON, R. S. Learning to predict by methods of temporal differences. **Machine Learning**, v. 3, n. 1, p. 9-44, 1988.

TAYLOR, M.; STONE, P. Behavior Transfer for Value-Function-Based Reinforcement Learning. In: Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05), Utrecht, Holanda, p. 53-59, jul. 2005.

WATKINS, C. J. C. H. **Learning from Delayed Rewards.** 1989. Tese (Doutorado) – Universidade de Cambridge, Cambridge.

WATKINS, C. J. C. H.; DAYAN, P. Q-Learning. **Machine Learning**, v. 8, p. 279-292, 1992.

WEISS, N.; HILDEBRAND, L. An exemplary robots soccer vision system. In: CLAWAR/EURON WORKSHOP ON ROBOTS IN ENTERTAINMENT, LEISURE AND HOBBY, 2004, Viena, Áustria, 2004.

WENSHING. Wenshing Electronics Co. Disponível em: <http://www.wenshing.com.tw>. Acesso em 9 abril 2007.

WHITESON, S.; KOHL, N.; MIIKKULAINEN, R.; STONE, P. Evolving Soccer Keepaway Players through Task Decomposition. **Machine Learning**, v.59, p. 5-30, 2005.

WIERING, M.; SCHMIDHUBER, J. Fast Online Q(λ). *Machine Learning*, v. 33, n. 1, p. 105-115, 1998.

YUJIN ROBOTICS. YSR_A Soccer Robot. Disponível em: <http://www.edrobot.com/english/product/ysra.asp>. Acesso em: 9 abril 2007.

ZIOU, D.; TABBONE, S. Edge Detection Techniques: An Overview. **International Journal of Pattern Recognition and Image Analysis**, v. 8, p.537-559, 1998.