CENTRO UNIVERSITÁRIO DA FEI
JOSÉ ANGELO GURZONI JUNIOR

UMA ARQUITETURA DE ALOCAÇÃO DE TAREFAS PARA SISTEMAS MULTI-ROBÔS UTILIZANDO APRENDIZADO POR REFORÇO

São Bernardo do Campo, SP
2011

JOSÉ ANGELO GURZONI JUNIOR

UMA ARQUITETURA DE ALOCAÇÃO DE TAREFAS PARA SISTEMAS MULTI-ROBÔS UTILIZANDO APRENDIZADO POR REFORÇO

Dissertação de Mestrado apresentada ao Centro Universitário da FEI em cumprimento parcial dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.
Orientador: Prof. Dr. Reinaldo A. C. Bianchi

São Bernardo do Campo, SP
2011

Gurzoni Junior, José Angelo

Uma Arquitetura de Alocação de Tarefas para Sistemas Multi-Robôs Utilizando Aprendizado por Reforço / José Angelo Gurzoni Junior. – São Bernardo do Campo, 2011.

90 f. : il.

Dissertação de Mestrado - Centro Universitário da FEI.

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

1. Aprendizado por Reforço. 2. Futebol de Robôs. 3. Inteligência Artificial. 4. Robótica Multiagentes. I. Bianchi, Reinaldo A. C., orient. II. Título.

CDU 621.38:621.9

Centro Universitário da **FEI**

APRESENTAÇÃO DE DISSERTAÇÃO
ATA DA BANCA JULGADORA

PGE- 10

Programa de Mestrado de Engenharia Elétrica

Aluno: José Angelo Gurzoni Junior

Matrícula: 108111-6

Título do Trabalho: UMA ARQUITETURA DE ALOCAÇÃO DE TAREFAS PARA SISTEMAS MULTI-RÔBOS UTILIZANDO APRENDIZADO POR REFORÇO

Área de Concentração: Inteligência Artificial Aplicada à Automação

Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi

Data da realização da defesa: 03/ fevereiro/ 2011

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

APROVADO ☒ REPROVADO ☐

São Bernardo do Campo, 03 / 02 / 11 .

MEMBROS DA BANCA JULGADORA

Prof. Dr. Reinaldo Augusto da Costa Bianchi

Ass.: Reinaldo A Bianchi

Prof. Dr. Flávio Tonidandel

Ass.: Tonidandel

Prof. Dr. Fábio Gagliardi Cozman

Ass.: Cozman

VERSÃO FINAL DA DISSERTAÇÃO

ENDOSSO DO ORIENTADOR APÓS A INCLUSÃO DAS RECOMENDAÇÕES DA BANCA EXAMINADORA

Reinaldo A Bianchi

Aprovação do Coordenador do Programa de Pós-graduação

Prof. Dr. Carlos Eduardo Thomaz

À minha mãe, Maria Estela, para quem os inúmeros brinquedos espalhados pelo chão, desmontados e remontados tantas vezes, eram as primeiras obras de um engenheiro. Seu amor e atos me ensinaram o que livro nenhum poderia ensinar.

À Vanessa, minha melhor metade. Seu amor e apoio me fazem uma pessoa melhor a cada dia.

Agradecimentos

À minha esposa, Vanessa Lieberg, por revisar o texto, por abdicar de tantas horas livres e por sempre me fazer seguir em frente. Aos meus segundos pais, Nelcy e Roberto Lieberg, por todo o carinho.

Ao meu orientador, Reinaldo Bianchi, o *Grande Oráculo*, que realizou com excelência ímpar sua função de transferir conhecimento e emprestou sua experiência e competência.

Ao mentor e amigo Flavio Tonidandel, cujo apoio e confiança permitiram que eu explorasse meu potencial acadêmico.

Aos grandes amigos Murilo Martins e Valquíria Fenelon, por todas as contribuições para o rumo deste trabalho, por ceder suas coletâneas de artigos científicos, das quais boa parte da referência bibliográfica foi retirada, revisado o trabalho, além de impedir meus devaneios.

Aos amigos e companheiros de pedalada André Santos, Daniel Malheiro, Diego Oliveira, Felipe Zanatto, Gabriel Francischini, Milton Cortez Jr. e Ronaldo Satomi. Este trabalho só foi possível graças ao seu empenho no projeto RoboFEI.

Ao Centro Universitário da FEI e seus funcionários, pelo apoio e dedicação.

Resumo

Agentes operando em domínios multiagentes precisam cooperar e coordenar suas ações, e em alguns casos, competir com adversários ao mesmo tempo. Muitos destes domínios são também dinâmicos, como o futebol de robôs, a exploração submarina, planetária ou os ambientes com presença humana, criando a necessidade de que os agentes sejam capazes de tomar decisões complexas e se adaptar rapidamente a novas condições.

Na literatura existem resultados positivos à respeito da aplicação do Aprendizado por Reforço em problemas complexos, em parte porque os agentes que utilizam esta técnica aprendem por experiência, sem a necessidade de modelos do ambiente em que operam. Porém, os requisitos computacionais do Aprendizado por Reforço são ainda restritivos, especialmente em domínios que necessitam de resposta em tempo real. Por outro lado, muitos dos sistemas de alocação de tarefas em multi-robôs encontrados na literatura tem tempos de execução e custo computacional baixos, ideais para estas aplicações.

Este trabalho apresenta uma arquitetura de alocação de tarefas em sistemas multi-robôs em que os agentes participam de leilões pelas funções de alto nível disponíveis e utilizam Aprendizado por Reforço para aprender o valor de cada uma destas funções, dada a situação em que a equipe de robôs se encontra. A arquitetura foi aplicada a uma equipe de futebol de robôs da categoria RoboCup Small Size. Foram comparados os desempenhos do mecanismo de alocação de tarefas quando agentes utilizavam valores de seus lances ajustados manualmente, quando os valores eram aprendidos por aprendizado por reforço e também por aprendizado por reforço com heurísticas.

Os resultados dos experimentos mostram que,o sistema de alocação de tarefas proposto é capaz de aumentar significativamente o desempenho da equipe, quando comparado com algoritmos em que o comportamento da equipe é pré-programado.

Palavras-Chave: Aprendizado por Reforço. Futebol de Robôs. Robótica Multiagentes.

Abstract

Agents operating in multi-agent domains need to cooperate and coordinate their actions, while, in some instances, also competiting with adversaries. Many of these domains are also dynamic, such as robot soccer, submarine or planetary exploration, and the environments with human presence, thus creating the need to have agents able to take complex decisions and to quickly adapt to new conditions.

There are positive results in the literature regarding the employment of Reinforcement Learning in complex problems, partially because agents using this technique can learn by experience, without the need to know models of the environment they operate in. However, computational costs of the Reinforcement Learning algorithms are still restrictive, especially in applications requiring real time responses. On the other hand, many of the multi-robot task allocation systems found in literature have low execution times and computational cost, making them ideal for this kind of application.

This work presents a multi-robot task allocation architecture where agents participate in auctions for the available high level functions and use Reinforcement Learning techniques to learn the value of each of these functions, given the situation of the team at that point in the match. The architecture is applied to a RoboCup Small Size league robot soccer team. The performance of the task allocation mechanism is compared among the cases when the agents use manually adjusted bidding values, when these values are learned by Reinforcement Learning and also by Heuristically Accelerated Reinforcement Learning.

The results show that the proposed task allocation system is capable of significantly increasing the team performance, when compared to algorithms which pre-program the team behavior.

Keywords: Reinforcement Learning. Robot Soccer. Multi-Agent Robotics.

Lista de Figuras

Lista de Abreviaturas

RL	*Reinforcement Learning* - Aprendizado por Reforço
CMAC	*Cerebellar Model Articulation Controller*
CNP	*Contract Net Protocol* - Protocolo Rede de Contratos
DAI	*Distributed Artificial Intelligence* - Inteligência Artificial Distribuída
HAQL	*Heuristically Accelerated Q-Learning* - Q-Learning acelerado por heurísticas
HAQ(λ)	*Heuristically Accelerated Q(λ)-Learning* - Q(λ)-Learning acelerado por heurísticas
MDP	*Markov Decision Process* - Processo de Decisão Markoviano
MLP	*Multi-Layer Perceptron* - Perceptron Multi-camada
MRTA	*Multi-Robot Task Allocation* - Alocação de Tarefas em Multi-Robôs
SPAR	*Strategic Positioning with Attraction and Repulsion* - Posicionamento Estratégico com Atração e Repulsão
STP	*Skills, Tactics and Plays* - Habilidades, Táticas e Jogadas
TD	*Temporal Differences* - Método das Diferenças Temporais

Sumário

Capítulo 1

INTRODUÇÃO

As pesquisas em sistemas multiagentes e em sistemas multi-robóticos são tópicos que vem atraindo crescente atenção da comunidade científica. Conforme mostra *Weiss* (1999), estes sistemas de fato apresentam diversos benefícios, dentre os quais destacam-se:

- A capacidade de paralelismo, que permite que tanto problemas inerentemente distribuídos, como sistemas de controle de tráfego aéreo, quanto problemas de escala intratável para um agente único, sejam resolvidos;
- Redundância e tolerância à falhas, já que um agente que apresente problemas pode ser substituído por outro;
- Escalabilidade, já que muitas vezes não é econômica ou tecnicamente viável criar um agente que possa realizar uma dada tarefa mas é possível criar diversos agentes, mais simples e com capacidades complementares, capazes de realizá-la.

Em um primeiro estágio do desenvolvimento dos sistemas multiagentes, arquiteturas populares foram as *bio-inspiradas*, como as propostas por *Matarić* (1992), arquiteturas de enxame (*swarm*), coleta de itens (*foraging*) ou similares, tipicamente baseadas em grandes quantidades de agentes homogêneos que desempenham tarefas individuais. Nestes sistemas a cooperação não é explícita mas sim emergente, ou seja, as tarefas desempenhadas pelos agentes criam um comportamento de grupo que o agente não tem consciência da existência. Uma desvantagem das arquiteturas de enxame é que são bastante específicas para a tarefa a que foram projetadas e apresentam problemas em aplicações onde a distribuição ótima das tarefas não é conhecida, muda dinamicamente ou quando as tarefas são fortemente acopladas.

Da busca por soluções mais genéricas e eficazes, aplicáveis às situações em que tarefas mais complexas ou com maior coordenação são necessárias, surgem arquiteturas como descreve *Par-*

ker (1998b), em que os agentes têm consciência da existência de outros e sabem que cooperar é benéfico, ou até mesmo fundamental, para a realização das tarefas. Estas arquiteturas são chamadas arquiteturas cooperativas explícitas.

Se as arquiteturas cooperativas implícitas podem ser, em grande parte, classificadas como bio-inspiradas, as explícitas podem ser classificadas em alguns grupos, que serão descritos em maiores detalhes ao longo do capítulo: comportamentais (*behaviorial*), como a conhecida arquitetura Alliance (*Parker*, 1998a) e, mais recentemente, a arquitetura ASyMTRe (*Tang e Parker*, 2005a); organizacionais, como *Chaimowicz et al.* (2002) ou *Browning et al.* (2005), onde definem-se papéis a serem preenchidos pelos agentes e; baseados em métodos de otimização (*Gerkey e Matarić*, 2003; *Dias et al.*, 2004; *Lerman et al.*, 2006), onde reduz-se o sistema a um problema de otimização da programação linear, frequentemente o problema de atribuição ótimo (*Optimal Assignment*), o que permite o uso de formalismos e teorias consolidadas como as estudadas pela pesquisa operacional, teoria dos jogos e economia. Estas arquiteturas geralmente se focam em torno de um ou mais dos três pontos básicos a seguir:

- Projeto de Equipe - Consiste da definição do número correto de robôs e suas capacidades para a formação de equipes aptas a realizar determinada missão;
- Planejamento - Dada uma equipe de robôs, é a decomposição de uma missão de alto nível em um plano eficiente formado por tarefas mais simples que, executadas ordenadamente, completam a missão;
- Alocação de Tarefas - Trata a questão de encontrar o conjunto de mapeamentos robô-tarefa mais eficiente, de forma a otimizar uma função de custo ou utilidade da missão planejada.

Destes pontos básicos o mais amplamente estudado é o problema de alocação de tarefas, também chamado de MRTA (*Multi-Robot Task Allocation* - Alocação de Tarefas em Multi-Robôs). Resolver esse problema, segundo *Gerkey e Matarić* (2003), é responder a uma pergunta aparentemente simples, mas de resposta não trivial: "Qual robô deve executar qual tarefa ?"

Análises sobre arquiteturas e abordagens existentes para a alocação de tarefas em multi-robôs, como as apresentadas por *Dias e Stentz* (2003), *Gerkey e Matarić* (2004), *Dias et al.* (2006) e *Parker* (2008), evidenciam a complexidade dos problemas de distribuição de tarefas em equipes robóticas autônomas. Ao mesmo tempo, mostram que, nesta classe de problemas,

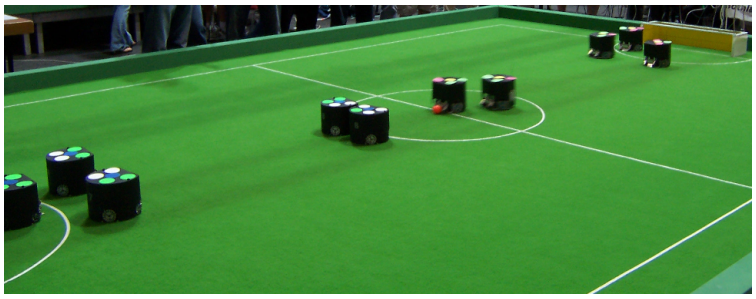


Figura 1.1: Cena de uma partida da categoria RoboCup Small Size (fonte: Robocup Federation).

não há uma abordagem superior em todos os critérios e domínios de aplicação, de forma que há espaço para novas técnicas e implementações.

1.1 Objetivo

O objetivo deste trabalho é definir e implementar uma arquitetura de alocação de tarefas em multi-robôs que utilize técnicas de Aprendizado por Reforço, em um domínio real, cooperativo e dinâmico, como o futebol de robôs.

1.2 Justificativa

Conforme será visto no Capítulo 2, existem diversas abordagens diferentes, com resultados interessantes, em sistemas de alocação de tarefas em multi-robôs. Porém, são poucas as implementações em ambientes mais complexos ou que utilizem técnicas de aprendizado. Por isso, neste trabalho, um domínio real e particularmente desafiador foi escolhido, o futebol de robôs categoria RoboCup Small Size (ilustrado na Figura 1.1), em que dois times, de 5 robôs cada, competem em um campo de 6 x 4 m.

O futebol de robôs é um problema desafiador, pois está contido em um ambiente multiagente em que existe, ao mesmo tempo, grande necessidade de cooperação e a presença de adversários. Além disso, é extremamente dinâmico, com robôs que chegam a quase 4 m/s de velocidade e podem chutar a bola com velocidades de até 10 m/s. Em um ambiente como este, a tomada de decisões, adaptação a novas condições, robustez e reação à imprevisibilidade do adversário, tem de ser quase imediatas, o que torna crucial que a alocação de tarefas dos agentes seja rápida e eficiente. Ao mesmo tempo, alocá-las de modo que produzam atitudes lógicas e compatíveis com as situações encontradas, sem utilizar técnicas de aprendizado, é extremamente difícil, tornando o futebol de robôs um cenário interessante para o uso de técnicas refinadas de MRTA

que aliem aprendizado.

Técnicas de aprendizado podem ser bastante atrativas neste tipo de problema, cujas instâncias são frequentemente problemas $\mathcal{NP}$-completos. E entre estas técnicas, o Aprendizado por Reforço (*Reinforcement Learning*) (*Sutton e Barto*, 1998) é especialmente atrativo para domínios de difícil modelagem, já que permite que os agentes aprendam por experiência, mesmo que não conheçam modelos do ambiente.

Porém, domínios competitivos impõem limitações à aplicação direta de Aprendizado por Reforço, uma vez que as partidas são relativamente rápidas e o aprendizado, mesmo utilizando técnicas heurísticas (*Bianchi et al.*, 2008), ainda tem tempos de convergência restritivos. Nesse sentido, dentre os sistemas de alocação de tarefas, uma classe se sobressai por sua simplicidade computacional e de implementação, além de boa fundamentação teórica: a classe dos métodos baseados em sistemas econômicos (apresentados na Seção 2.1). Desde que as funções utilidade dos agentes tenham valores corretamente atribuídos, as soluções são rápidas e eficientes.

Assim, o uso conjunto de técnicas de aprendizado, capazes de aprender funções utilidade mesmo em domínios de difícil modelagem, em conjunto com sistemas econômicos, que podem garantir que destas funções utilidade individuais emerja uma dinâmica de grupo eficiente e até mesmo ótima, parece um caminho promissor a ser seguido neste trabalho.

1.3 Organização do Trabalho

O Capítulo 2 apresenta uma revisão bibliográfica sobre as arquiteturas existentes para alocação de tarefas em multi-robôs e uma breve discussão de suas características. Em seguida, no Capítulo 3, técnicas de Aprendizado por Reforço existentes na literatura são apresentadas, bem como técnicas heurísticas para sua aceleração. No Capítulo 4 está contida a proposta de uma arquitetura de alocação de tarefas usando Aprendizado por Reforço, aplicada ao futebol de robôs, com detalhes de sua implementação e da metodologia de testes.

Capítulo 2

ALOCAÇÃO DE TAREFAS EM MULTI-ROBÔS

Apesar da existência de algum debate quanto à correta classificação dos sistemas existentes para alocação de tarefas na literatura, neste trabalho define-se uma classificação próxima à apresentada em *Parker* (2008). Neste estudo, Parker classifica as arquiteturas em três grupos distintos: bio-inspiradas, comportamentais e organizacionais. As bio-inspiradas (*Mataric*, 1995; *Kube e Zhang*, 1993) são bastante importantes, tendo predominado entre os primeiros sistemas de alocação de tarefas, e ainda sendo bastante utilizadas. Exemplos destes sistemas são as implementações de enxames e colônias de insetos. Contudo, embora importantes, este trabalho não faz uma revisão da classificação bio-inspirada, pois não se foca em arquiteturas de coordenação implícita.

Já os sistemas comportamentais são baseados em ontologias, semântica e bases de conhecimento, enquanto os organizacionais são compostos pelos sistemas inspirados nas relações sociais humanas ou nas teorias econômicas (mais comumente as de mercado). A distinção que este trabalho faz na classificação, em relação à *Parker* (2008), é considerar os sociais e os econômicos como dois grupos distintos, separando assim o que seria o agrupamento chamado organizacional. Este capítulo oferece uma análise mais aprofundada das soluções propostas para alocação de tarefas em cada um dos grupos acima descritos.

2.1 Arquiteturas Baseadas em Sistemas Econômicos

Arquiteturas baseadas em sistemas econômicos são inspiradas no estudo das economias de mercado, onde indivíduos ou grupos buscam maximizar lucro e podem agir livremente em

interesse próprio. A forma mais elementar de realizar MRTA neste tipo de sistema pode ser dada pela definição abaixo.

(Alocação de Tarefas em Multi-Robôs) *Dado um conjunto de tarefas T, um conjunto de robôs R e uma função utilidade u_{rt} para cada robô $r \in R$ que especifique a disposição ou interesse em realizar cada tarefa ou grupo de tarefas, encontrar a alocação $A^* \in R^T$ que maximize uma função objetivo global $U : R^T \rightarrow \mathbb{R}^+ \cup \{\infty\}$*

Esta definição utiliza a função *utilidade*, um conceito-chave da economia, bem como da teoria dos jogos e da pesquisa operacional. *Utilidade*, também chamada de aptidão, custo ou valor, baseia-se na noção de que o indivíduo consegue estimar o valor de uma dada ação. Seja, por exemplo, uma tarefa que ofereça pagamento p, e seja c o custo que um determinado indivíduo estima para realizá-la. Então uma provável função utilidade deste indivíduo é $u = p - c$, função esta que expressa seu interesse em realizar a tarefa em troca do pagamento. Caso $p > c$ o indivíduo tem interesse naquela tarefa. Cabe salientar, no entanto, que o lucro é apenas um exemplo, e a função utilidade não é necessariamente lucro. Economistas chegam até mesmo a chamá-la de função de felicidade de um indivíduo, e na analogia com sistemas robóticos um exemplo é um robô que prefere não obter tanto lucro mas retornar à base com segurança, satisfazendo assim tanto seu desejo por lucro quanto por continuar ativo. Obviamente, caso existam n tarefas, ao indivíduo interessará realizar a que trouxer o maior valor u.

2.1.1 Arquiteturas Clássicas

Dentre os primeiros trabalhos baseados em sistemas econômicos, duas arquiteturas se destacaram e formaram a base de boa parte dos sistemas modernos. São elas o Contract Net Protocol (*Smith*, 1980) e os sistemas de leilão desenvolvidos por *Bertsekas* (1979).

Na arquitetura Contract Net Protocol (CNP), tarefas são oferecidas através de anúncios de leilão e agentes realizam lances baseados em suas funções utilidade. O maior lance então ganha o contrato para a realização da tarefa, até que aconteça um próximo leilão. Este sistema de oferta e lance é chamado de leilão de primeiro preço, já que os agentes simplesmente aceitam que a maior oferta vença e não disputam como fariam em um leilão tradicional. O CNP estabelece um protocolo de mensagens para o anúncio das tarefas aos agentes e suas respectivas respostas. Neste protocolo, as tarefas são anunciadas juntamente com uma lista de requerimentos para que sejam realizadas. Os agentes recebem a mensagem e, caso atendam aos requisitos exigidos pela

tarefa, podem então enviar ao agente leiloeiro uma nova mensagem, especificando um lance pela tarefa, frequentemente uma medida escalar do seu grau de aptidão em realizá-la.

O CNP também implementa um método para a revenda de contratos, permitindo que um agente repasse uma tarefa à outro mais apto, através de um novo leilão daquela tarefa. Essa característica faz do CNP uma arquitetura decentralizada, onde quaisquer agentes podem se comunicar entre si, tanto oferecendo tarefas quanto dando lances por elas, criando assim tolerância à falhas. Entretanto, em essência, o Contract Net foi concebido para resolver o chamado *problema da conectividade*, que consiste em encontrar um agente apropriado para trabalhar em uma determinada tarefa. A maior contribuição do CNP está exatamente nesse protocolo de comunicação e controle do solucionador de problemas distribuído, capaz de encontrar o agente apropriado. O CNP não otimiza as alocações, apenas as encontra e estabelece.

Bertsekas (1979) também é considerado um dos primeiros a modelar conceitos da economia em sistemas multiagentes. Publicou diversos trabalhos sobre o tema, em que utiliza programação linear e introduz o uso de sistemas de leilão similares aos que ocorrem no mundo real (*Bertsekas*, 1992). O uso de programação linear oferece uma ferramenta importante para abordar otimização e, ao contrário do CNP, focado principalmente em encontrar um agente *qualquer* que possa executar um dada tarefa, Bertsekas cria uma arquitetura capaz de otimizar as alocações através da competição entre os agentes, que buscam aumentar lucros.

Sua abordagem consiste em tratar o sistema como um problema de atribuição ótima e formulá-lo por programação linear inteira (*Gale*, 1960), como mostra a Eq. (2.1), onde o objetivo é encontrar $m.n$ números inteiros α que maximizem a utilidade global U do sistema. U é a somatória das utilidades u que cada agente m estima para cada tarefa n, com w representando a prioridade da tarefa. As restrições em (2.2) asseguram, já que α só pode ser 0 ou 1, que apenas um agente será alocado para cada tarefa.

$$\max(U) = \max_{m,n} \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_{ij} u_{ij} w_j \tag{2.1}$$

sujeito a

$$\sum_{i=1}^{m} \alpha_{ij} = 1, \quad 1 \leq j \leq n \tag{2.2}$$

$$\sum_{j=1}^{n} \alpha_{ij} = 1, \quad 1 \leq i \leq m$$

Para entender como a Eq. (2.1), e em geral a abordagem dos problemas de atribuição ótima, funciona é necessário entender o conceito de *dualidade*: Para todo problema de maximização em programação linear (neste caso o lucro total da economia), existe um problema de minimização (a soma dos preços das tarefas), em que deve-se encontrar m inteiros u e n inteiros v que minimizem Eq. (2.3). O problema de minimização é então chamado *dual* do problema *primal* de maximização.

$$P = \sum_{i=1}^{m} u_i + \sum_{j=n}^{n} v_j \tag{2.3}$$

sujeito a

$$u_i + v_j \geq U_{ij}, \quad \forall i, j \tag{2.4}$$

Segundo o teorema da dualidade, a utilidade total das soluções ótimas do primal e do dual são iguais, portanto eles são equivalentes (*Gale*, 1960). Este é um teorema de fundamental importância em sistemas econômicos como os leilões de *Bertsekas* (1992), pois através dele prova-se que a solução ótima ocorre quando os preços são tais que não existem agentes competindo pela mesma tarefa. A condição ótima do equilíbrio advém diretamente de uma relação chamada relação de folga complementar (do termo em inglês *complementary slackness*) existente entre o primal do lucro dos agentes e seu dual, os preços das tarefas.

Uma vez introduzidos os conceitos da programação linear, é possível então explicar problemas de atribuição simétricos e assimétricos. Nos simétricos, existem n agentes e n tarefas, portanto é possível atribuir a todos os agentes uma tarefa sem que haja tarefas sobressalentes. Sob a ótica de um problema de otimização, essa distribuição em que não restam agentes ou tarefas garante que a solução ótima seja alcançada. Já nos problemas de atribuição assimétricos, existem n agentes e m tarefas, com $m > n$. Nestes, deve-se atribuir a todos os agentes alguma tarefa, porém não é necessário atribuir todas as tarefas a algum agente, o que leva ao fim da garantia de que a solução ótima seja alcançada, uma vez que o preço das tarefas não atribuídas passa a não ser garantidamente mínimo. *Bertsekas* (1979) ajusta seus algoritmos para os problemas assimétricos, porém a garantia de encontrar a solução ótima passa a ser aproximada e não garantida, através de uma relação com o incremento de preço aplicado entre cada rodada do leilão.

Quando analisados em relação ao tipo de processamento, centralizado ou distribuído, nota-se que sistemas econômicos são bastante flexíveis. Nas aplicações com quantidades pequenas

ou médias de agentes e tarefas, abordagens centralizadas são mais comuns (*Werger e Mataric*, 2000; *Carpin e Parker*, 2002). Sistemas completamente centralizados produzem soluções em menor tempo e, em diversos casos, garantem que estas sejam ótimas. Por exemplo, caso seja possível coletar as utilidades de cada par robô-tarefa e processá-las centralizadamente, a solução ótima de U pode ser dada trivialmente, em tempo polinomial, através de um algoritmo de maximização como o método húngaro (*Kuhn*, 1955), que precisa de apenas $O(mn^2)$ tempo.

No entanto, estes sistemas podem ser pouco resistentes à falhas, por serem centralizados. Além disso, à medida que o número de agentes aumenta o problema pode se tornar intratável, seja por maiores demandas por comunicação, seja pelo tempo gasto nela ou ainda porque não é possível processar centralizadamente o volume de informações.

Em aplicações maiores ou que necessitam maior tolerância a falhas, as soluções distribuídas são preferidas, normalmente mais eficientes e com menor demanda por comunicação e recursos. Porém, topologias distribuídas podem levar à soluções sub-ótimas.

Prova da versatilidade dos métodos econômicos é que, entre os mais recentes, é crescente a adoção de métodos híbridos (*Dias et al.*, 2004; *Kalra et al.*, 2005), onde o processamento é distribuído entre os agentes mas estes, oportunisticamente, podem decidir por resolver tarefas centralizadamente em suas CPUs, decompondo-as, revendendo pedaços a outros agentes e monitorando seu progresso.

A seguir serão abordadas as arquiteturas econômicas mais recentes.

2.1.2 Arquiteturas Modernas

Muitos dos sistemas econômicos para MRTA desenvolvidos na década de 2000 trazem elementos do Contract Net Protocol (CNP), utilizam algum tipo de leilão em seus algoritmos, ou ambos. No entanto, antes de descrevê-los, é interessante abordar duas propostas aplicadas em sistemas multiagentes que evoluem destes algoritmos: *Sandholm* (1993) e *Sandholm e Suri* (2000).

Sandholm (1993) apresenta uma importante extensão do estudo em torno do CNP em que formaliza-se um algoritmo de decisão de ofertas e definição do vencedor, tópicos que o algoritmo CNP original deixa um tanto indefinidos. Neste estudo aparecem também o conceito de interação entre tarefas e uma solução para o problema de congestionamento de rede, causado por grandes quantidades de mensagens de anúncio. A solução do problema de comunicação permite que um número elevado de ofertas seja negociado simultaneamente.

Sandholm e Suri (2000) estendem o conceito de leilões, através de uma arquitetura em que os agentes participam de leilões simultâneos de tarefas complementares e podem dar lances por combinações de tarefas. Estes leilões são chamados leilões combinatórios. Neles, por poderem comprar grupos de tarefas, os agentes tem a possibilidade de otimizar seu trabalho, ao contrário do que acontece em leilões simples (sequenciais ou paralelos), em que lances devem ser feitos isoladamente para cada tarefa. Um ponto teórico importante deste trabalho é mostrar que a determinação do vencedor em um leilão combinatório é $\mathcal{NP}$-completo e inaproximável (por meio da redução ao problema de empacotamento de conjuntos) mas que algoritmos de busca podem produzir soluções com boas aproximações na prática.

Dentre as arquiteturas projetadas especificamente para sistemas robóticos, destacam-se as descritas a seguir.

Uma das primeiras arquiteturas a utilizar métodos econômicos em aplicações robóticas foi o M+ (*Botelho e Alami*, 1999, 2000), embora tenha sido implementado apenas em simulação. O M+ é um algoritmo distribuído baseado no CNP em que os robôs escolhem incrementalmente em qual tarefa trabalharão, dentre as executáveis - tarefas cujas predecessoras foram concluídas ou estão em execução. Cada robô então formula seu plano para a execução das tarefas, estima os custos de executar cada uma e dá uma oferta pela de menor custo. O robô com o menor custo dentre todos os outros é selecionado para executá-la.

Gerkey e Matarić (2002) implementa uma arquitetura similar ao M+, também baseada no CNP, chamada Murdoch. Destaca-se nela a utilização de um protocolo em que os robôs se inscrevem em grupos de comunicação de acordo com os recursos que possuem e as ofertas são comunicadas apenas aos que estão inscritos nos grupos necessários à realização da tarefa. No Murdoch os robôs têm também a capacidade de leiloar tarefas para outros robôs e os contratos tem duração de tempo pré-definida, o que gera um sistema com tolerância à falhas.

Porém, Murdoch não é um sistema com grandes inovações teóricas. Seu grande valor está na implementação de um sistema econômico em robôs reais. O sistema depende que os projetistas criem toda a estrutura de sub-divisão de tarefas e utiliza apenas um algoritmo guloso para decidir qual robô deve executar qual tarefa. A inserção de novas tarefas *on-line* é analisada e os autores dizem não representar grandes problemas, desde que suas quantidades sejam relativamente pequenas.

Dias et al. (2004) apresentam o TraderBots, um sistema completamente distribuído que utiliza conceitos mais avançados da economia, como o lucro de cada agente e o uso de uma

moeda de pagamentos. O diagrama estrutural do sistema pode ser visto na Figura 2.1, onde é possível ver os mecanismos internos de comercialização de tarefas, o *Trader*, de comunicação e execução, independentes entre si. Como nas economias de mercado, cada robô tenta acumular o máximo de riqueza, agindo em interesse próprio. Para isso, um algoritmo executando no módulo *Trader* de cada robô calcula os custos das tarefas sendo oferecidas e então negocia as que lhe interessam por um pagamento que cubra seus custos, como energia e desgaste, e permita algum lucro. Por meio desse mecanismo de negociação e da capacidade de subcontratação das tarefas para outros robôs é possível reduzir custos, o que é benéfico ao grupo como um todo.

A otimização do conjunto de tarefas como ação de grupo emerge do princípio de que sendo cada tarefa uma parte que contribui para o sucesso de uma missão, garante-se que mesmo agindo em interesse próprio os robôs contribuirão para a otimização da missão.

O TraderBots possui outros aspectos importantes:

- É um sistema projetado para o caso em que os robôs são responsáveis por cumprir várias tarefas. Por esse motivo os cálculos de custos são sempre de custos marginais (a diferença do custo total da lista de tarefas com e sem a tarefa sendo avaliada) e existem mais chances de otimização pela revenda ou subcontratação de tarefas.

- Existem mecanismos de tolerância à falhas por todo o sistema, como por exemplo no gerenciamento de pagamentos, onde somente após a conclusão da tarefa um pagamento é efetuado, e no sistema de leilões que prevê venda imediata, a qualquer preço, caso o robô detecte um alarme de defeito nele mesmo.

- Oportunisticamente um agente pode obter lucro simplesmente comprando e vendendo planos rentáveis. Isso permite que agentes/robôs dotados de maiores capacidades exerçam papéis de liderança, sem no entanto comprometer a tolerância à falhas.

Outra arquitetura bastante similar ao TraderBots é o Hoplites (*Kalra et al.*, 2005). A principal diferença está na capacidade do Hoplites em lidar com tarefas que requerem coordenação de múltiplos robôs, capacidade atingida através de uma mudança que permite aos robôs avaliar a lucratividade de planos completos ao invés de tarefas separadas, bem como vender e comprar participações nestes planos. Em *Kalra et al.* (2007) a arquitetura é estendida, tornando-se mais genérica e com maior variedade de algoritmos de planejamento, já extrapolando as definições de uma arquitetura de alocação de tarefas e abordando questões de planejamento de missões. Planejadores de missão são algoritmos que fazem parte de outro tópico de estudo dos sistemas

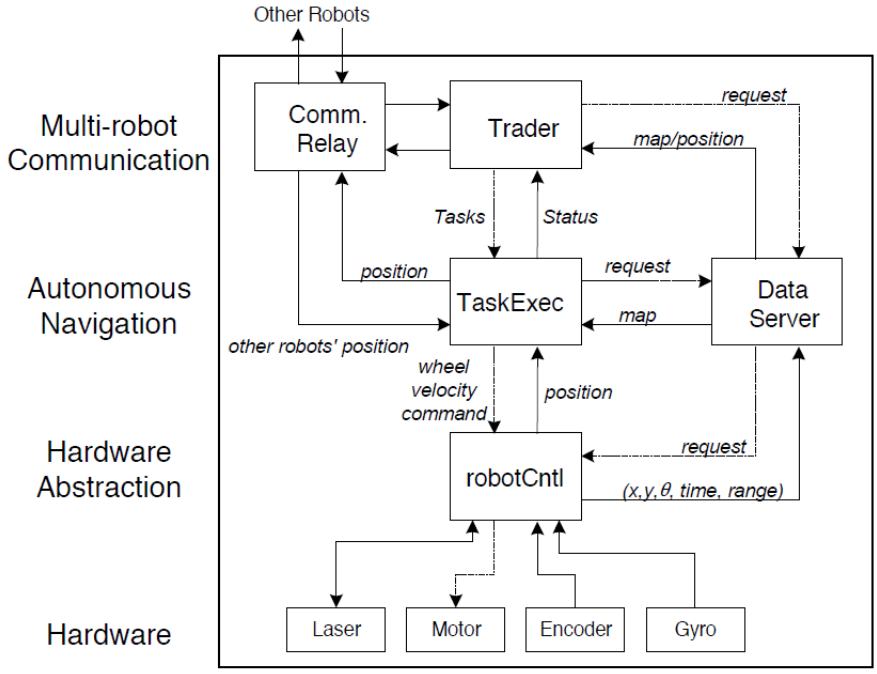


Figura 2.1: Diagrama estrutural do TraderBots, com seus módulos principais: comercialização (*Trader*), comunicação (*Comm. Relay*), execução de tarefas (*TaskExec*) e controle do robô (*robotCtnl*). (*Dias et al.*, 2004).

multiagentes, cujo objetivo é definir quais são as tarefas necessárias para atingir um determinado objetivo, ao passo que os sistemas de MRTA assumem que tal definição de quais são as tarefas a serem executadas é pré-existente.

2.2 Sistemas Comportamentais

Um dos mais conhecidos sistemas comportamentais é o Alliance (*Parker*, 1998a), uma arquitetura baseada em comportamentos, inspirada por algoritmos de inteligencia artificial distribuída (DAI). O objetivo principal do Alliance era criar um sistema cooperativo tolerante à falhas e incertezas, totalmente distribuído e aplicável em tempo real, usando para isto sistemas comportamentais e bases de conhecimento.

Na arquitetura Alliance, que pode ser vista na Figura 2.2, diversos comportamentos são embutidos nos robôs, em diferentes níveis de abstração. Nos níveis mais baixos, existem comportamentos como desvio de obstáculo e detecção de itens, enquanto nos níveis mais altos estão comportamentos como construção de mapas e exploração de ambientes. A diferença entre o Alliance e outros sistemas comportamentais está no agrupamento destes comportamentos em diversos conjuntos, responsáveis por cumprir determinadas partes da tarefa, e uma camada, acima de todas as outras, onde estão os chamados comportamentos motivacionais, ou simplesmente *motivações*. Através destas motivações, os robôs percebem seu ambiente, as motivações

e estados de outros robôs e são capazes de responder dinamicamente ao ambiente. Elas são responsáveis por ativar alguns conjuntos de comportamentos e inibir outros, de forma a cumprir a tarefa proposta.

Das motivações utilizadas pelo Alliance, duas são fundamentais ao sistema, *impaciência* e *consentimento*. A impaciência é a motivação pela qual um robô deseja realizar uma tarefa e à medida que o tempo passa, passa a ter cada vez mais motivação por cumpri-la, exceto se ele mesmo ou outro robô já o estiver fazendo. Já o consentimento governa o desejo que o robô tem de desistir de determinada tarefa que esteja executando, à medida que o tempo passa e o robô não detecta progresso na realização desta tarefa. É seu desejo de mudar para tarefas onde ele é mais produtivo. As bases de conhecimento dos robôs são as responsáveis por detectar motivações dos agentes à sua volta. É interessante notar que se for possível que os robôs detectem os estados de seus companheiros e de suas ações completamente quase nenhuma comunicação é necessária. Na prática, porém, o sistema tem algoritmos de comunicação explícita entre os agentes, já que na maioria dos casos detectar o estado de um companheiro não é trivial.

Uma extensão deste sistema é apresentada ainda no mesmo trabalho, chamada L-Alliance (*Learning*-Alliance), e consiste em algoritmos que monitoram a performance das motivações e atualizam os parâmetros que fazem com que sejam inibidos e ativados. Em trabalho posterior (*Parker et al.*, 2004) os autores se aprofundam na criação de motivações mais complexas para diferentes cenários, incluindo tarefas dependentes entre si, sempre seguindo as mesmas características dos trabalhos anteriores.

Uma crítica feita ao Alliance e à sua variação, o L-Alliance, é que estas arquiteturas não contém métodos de negociação pelos quais os robôs possam conjuntamente chegar à uma combinação ótima de tarefas, deixando-os apenas com suas próprias avaliações sobre qual sua aptidão em relação aos outros robôs para determinada tarefas. As motivações devem ser programadas para tratar esse tipo de evento, o que pode levar a problemas potencialmente intratáveis. Os autores mencionam que essa é uma opção feita para dar tolerância à falhas e robustez ao sistema, porém esse argumento só é forte o suficiente se os algoritmos de negociação mencionados forem algoritmos que não renegociam sua alocação uma vez que esta é determinada.

Outra crítica é que não há garantias de se atingir a condição ótima, ao contrário de outros métodos que trazem fundamentação da pesquisa operacional e outros campos do conhecimento. Além disso, o sistema é limitado à execução de tarefas independentes e sem hierarquia.

Outra arquitetura baseada em conceitos comportamentais é o ASyMTRe (*Tang e Parker*,

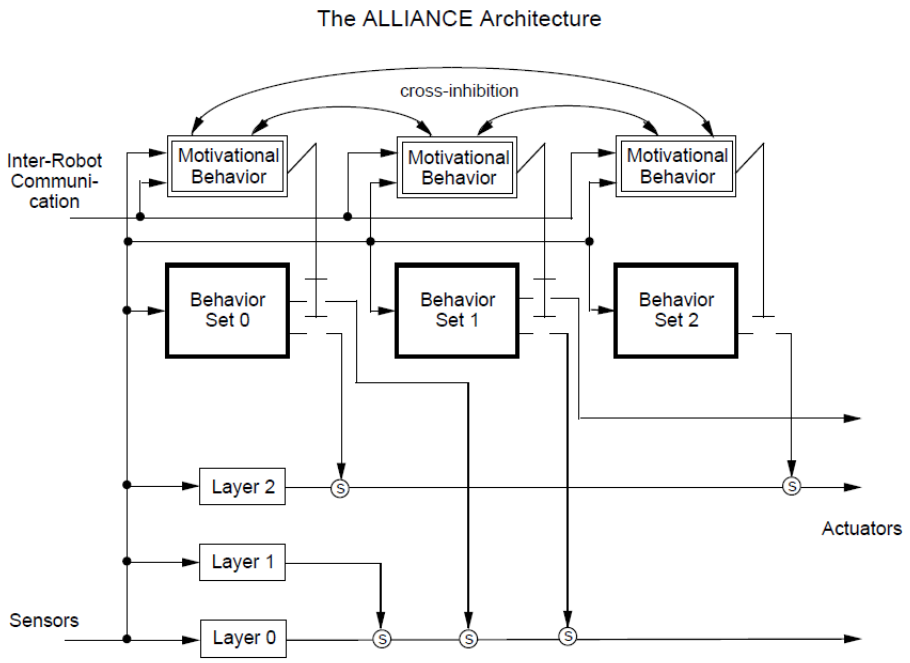


Figura 2.2: Diagrama de blocos da arquitetura Alliance. As linhas que ligam as motivações (*Motivational Behaviors*) aos comportamentos (*Behavior Sets*) indicam a capacidade das motivações em inibir ou permitir comportamentos. Por sua vez, os comportamentos permitem ou inibem atuadores específicos do robô. (*Parker*, 1998a).

2005a), uma arquitetura que apresenta uma metodologia não só para alocação de tarefas mas também para a definição de quais tarefas devem existir e suas relações de dependência, visando assim solucionar o problema de *como* a equipe de robôs deve decompor a missão, não somente o problema de *quem* vai cumprir cada uma das tarefas que a formam.

O ASyMTRe utiliza esquemas como os de *Arkin* (1987) e a teoria de invariáveis da informação (*Donald et al.*, 1997) para realizar o mapeamento das características intrínsecas das tarefas, capacidades de sensores e atuadores, para a forma de informações semânticas, abstrações que agentes e robôs diferentes podem entender como uma linguagem comum. Uma vez que existe esta linguagem para descrever as tarefas e capacidades dos robôs, é possível encontrar equivalências que, por sua vez, permitem agrupar diferentes combinações de sensores e atuadores (mesmo entre diferentes robôs), atendendo aos requisitos impostos pelas tarefas. A teoria de invariáveis da informação é a base para que o ASyMTRe consiga conectar automaticamente os diversos comportamentos existentes nos robôs, de acordo com as tarefas a serem cumpridas.

O algoritmo de busca por soluções utilizado pelo ASyMTRe é heurístico, começando de forma gulosa pelas soluções mais promissoras e então checando a qualidade de todas as potenciais soluções, armazenando sempre a melhor. Desta forma prova-se que o algoritmo é ótimo dado que haja tempo suficiente para analisar todas as possibilidades. No caso de haver limitação no tempo disponível para processamento, já que k tarefas em equipes de n robôs podem resultar

em n^k análises, então o algoritmo entregará a melhor solução encontrada durante seu tempo de operação. Quanto mais tempo estiver disponível para processamento, melhor será a solução.

Embora o ASyMTRe seja um sistema centralizado, existe uma variação dele criada para processamento distribuído, o ASyMTRe-D (*Tang e Parker*, 2005b), implementado utilizando o CNP, descrito na Seção 2.1.1. Em trabalho posterior (*Tang e Parker*, 2007) os autores o estendem ainda mais na direção dos sistemas econômicos, implementando leilões para a parte do sistema responsável pela alocação de tarefas. Por esse motivo, entende-se que o ASyMTRe e suas variações devem ser classificados como sistemas híbridos, não apenas comportamentais.

2.3 Sistemas Sociais

Sistemas sociais são inspirados na organização das sociedades humanas, em que indivíduos têm diferentes cargos e atribuições e desempenham suas funções de acordo com estes cargos. Neste tipo de sistema são utilizadas abstrações do campo da psicologia e sociologia para modelar as interações individuais entre os agentes e a dinâmica do grupo (ou grupos) que eles formam. Abstrações da economia também podem ser vistas como sistemas sociais, no entanto, conforme dito no inicio do capítulo, neste trabalho os sistemas inspirados na economia são classificados separadamente, na Seção 2.1, logo não estão inclusos nesta. Nos sistemas sociais voltados à robótica, dois tipos são comumente encontrados, os sistemas em que os agentes desempenham papéis e os sistemas chamados de modelos de equipes. Estas duas subdivisões dos sistemas sociais serão abordadas a seguir.

Sistemas que contém papéis a serem desempenhados pelos agentes são normalmente empregados para particionar a aplicação em porcões tratáveis por um determinado agente ou subgrupo deles. Estes sistemas são muito similares aos sistemas comportamentais descritos anteriormente, mas existe uma diferença fundamental, que fica clara através do seguinte exemplo: No Alliance, um robô tem um comportamento que expressa sua impaciência. Já em um sistema baseado em papéis, um possível papel é empacotador. Impaciência é um comportamento interno do indivíduo, não imposto por sua função no grupo, ao passo que empacotador é claramente sua função naquela sociedade ou organização.

Stone e Veloso (1999) apresentam uma arquitetura aplicada ao futebol de robôs simulado em que os agentes podem selecionar entre diversos papéis como o de atacante, defensor, meio-campista, lateral, líbero, entre outros. Estes papéis não determinam totalmente e de forma rígida o que eles devem fazer, mas sim quais suas diretrizes de comportamento, pré-disposições. Uma

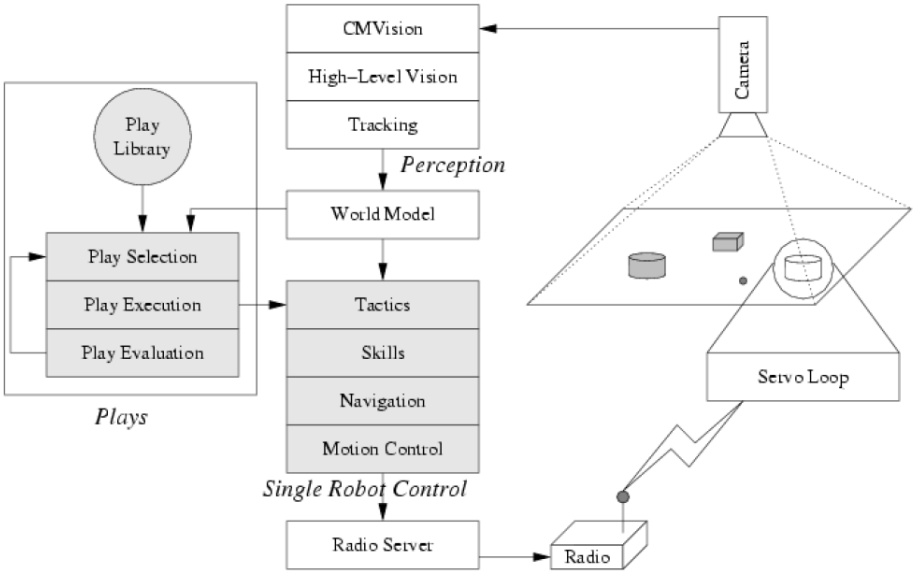


Figura 2.3: Diagrama de blocos do software da equipe CMDragons. A arquitetura STP pode ser vista nos blocos em cinza, os módulos de jogadas, táticas e habilidades (*plays*, *tactics* e *skills*). (*Browning et al.*, 2005)

camada de ações mais simples dentro dos papéis ainda permite flexibilidade em definir o que o agente deve fazer.

Outra arquitetura nos mesmos moldes é a arquitetura STP (*Skills, Tactics and Plays*), aplicada nos robôs reais da equipe CMDragons desde 2004 até o presente (*Bowling et al.*, 2004; *Browning et al.*, 2005; *McMillen e Veloso*, 2006; *Bruce et al.*, 2008). É uma arquitetura que segue o mesmo conceito de camadas do trabalho de Stone (*Stone e Veloso*, 1999), separando ações de controle de baixo nível de ações mais complexas, mas com uma série de evoluções. A Figura 2.3 mostra o software da equipe CMDragons, incluindo os módulos da arquitetura STP.

Dentre as evoluções do STP estão a alocação dinâmica dos papéis em tempo de execução, sem a necessidade de aguardar uma paralisação do jogo, como ocorre na arquitetura de Stone (*Stone e Veloso*, 1999), e a adoção de uma camada de processamento, denominada camada de jogadas, responsável por implementar políticas conjuntas aplicadas a vários robôs, também dinamicamente selecionadas em tempo de execução. Esta camada de jogadas efetivamente comanda a camada de papéis e faz com que estes tenham características um pouco diferentes, agora não mais como atacantes, meio-campistas ou defensores. Ao invés disso, são peças dentro da jogada específica. Exemplos desses papéis são recebedor da bola ou marcador do oponente com a bola.

A decisão sobre qual jogada deve ser executada é feita em duas fases, através de predicados de estado altamente parametrizados. Na primeira fase, as jogadas são avaliadas contra sua aplicabilidade naquele determinado estado de mundo. Na segunda fase, as jogadas aplicáveis

são então analisadas por um algoritmo do tipo *weighted expert* (*Littlestone e Warmuth*, 1994), em que cada jogada é um *expert* dando um conselho ao processo de decisão, e este seleciona o melhor. O algoritmo tem capacidade de adaptação durante um jogo, ajustando os pesos de cada conselho à medida que os critérios de terminação da jogada (que medem seu sucesso ou falha) são conhecidos.

Cabe mencionar que, nesta arquitetura, os próprios autores salientam que a criação de jogadas e papéis, bem como a parametrização dos mesmos, é desafiador e trabalhoso. Quando uma nova implementação é necessária, especialistas no sistema levam semanas ou mais criando todas as relações entre os parâmetros, ajustando valores das funções que selecionam cada jogada, testando e assegurando-se que não existam conflitos com as jogadas existentes.

Nota-se também que, da mesma forma que as motivações dos sistemas comportamentais (como o Alliance), o uso de papéis nestes trabalhos está contido em arquiteturas com diversas camadas, que começam em níveis mais baixos, individuais, passam pelos papéis em si, que compõem a camada que define as características dos agentes, e terminam em níveis que tem abrangência sobre todo o grupo.

2.4 Discussão

Das arquiteturas abordadas neste capítulo, diversas poderiam ser aplicadas, em associação com algoritmos de aprendizado, neste trabalho. Entre os métodos comportamentais, o ASyMTRe, uma evolução do Alliance, é uma opção que aborda tanto alocação de tarefa quanto planejamento e seus métodos semânticos criam a possibilidade de projetar soluções por combinação de características, programadas como comportamentos mais simples. Um exemplo disso seria a criação de funções de passe, chute ao gol e drible e então utilizar o algoritmo de aprendizado para encontrar as conexões entre estas funções, de modo a cumprir as tarefas de se defender e atacar durante o jogo de futebol. Com isso, o algoritmo de aprendizado substituiria o algoritmo de busca do ASyMTRe original. Esse tipo de abordagem aparece em *Tang e Parker* (2008), trabalho que apresenta uma implementação do ASyMRTe aliado a um algoritmo genético, de modo similar ao descrito acima. No entanto, o uso do ASyMTRe aliado a um algoritmo de aprendizado, produziria um trabalho muito similar ao já realizado em *Tang e Parker* (2008), mesmo que o algoritmo de aprendizado fosse diferente.

Outra opção seria utilizar um sistema social como o implementado em *Browning et al.* (2005). Como a aplicação desta arquitetura é também o futebol de robôs e ela também uti-

liza, em sua versão mais recente (*McMillen e Veloso*, 2006), um método de aprendizado, seria também um trabalho inevitavelmente muito parecido com o STP.

Entre os métodos econômicos, vários são de implementação simples e apresentam vantagens como rapidez de processamento e possibilidade de uso distribuído, como o Murdoch ou alguma variação feita a partir de seus fundamentos. Essa rapidez seria bastante interessante em um ambiente como o proposto neste trabalho, e a possibilidade de estudar um mesmo sistema na forma distribuída e centralizada é interessante, já que o ambiente aceita ambas. Um trabalho que utiliza tanto métodos econômicos e aprendizado é *Kose et al.* (2004). Contudo, neste trabalho MRTA e RL não são usados em conjunto, mas sim alternadamente. Alguns agentes utilizam aprendizado enquanto outros utilizam leilões para decidir qual papel adotar.

A conclusão então foi a de utilizar um método econômico que oferecesse simplicidade e rapidez e não produzisse um trabalho demasiadamente similar aos existentes. O capítulo 4 apresentará a proposta de uma arquitetura de MRTA através de métodos econômicos aliada ao Aprendizado por Reforço.

O próximo capítulo apresenta o restante da revisão bibliográfica deste trabalho, sobre o tópico Aprendizado por Reforço.

Capítulo 3

APRENDIZADO POR REFORÇO

Uma definição de Aprendizado por Reforço (RL) que expressa bem o conceito dessa classe de problemas, dada por *Kaelbling et al.* (1996), é a de que "Aprendizado por Reforço é o problema enfrentado por um agente que deve aprender como se comportar em um ambiente dinâmico através de suas interações com ele, por tentativa e erro". Este capítulo apresenta uma revisão do RL, bem como algoritmos que serão de relevância para a implementação proposta neste trabalho.

3.1 O Problema do Aprendizado por Reforço

Em um problema de Aprendizado por Reforço (*Sutton e Barto*, 1998), o agente interage com o ambiente em um ciclo de passos discretos que alterna fases de percepção e ação: A cada nova interação, o agente observa o estado s do ambiente e escolhe uma ação a. Ao executar esta ação, o agente recebe um sinal escalar de reforço $r_{s,a}$, que indica a qualidade do estado resultante s'. O objetivo do agente é então determinar qual a melhor ação para cada estado, a política π, que maximiza a somatória dos reforços recebidos. Estes reforços são também chamados recompensas ou penalizações.

Pode ser que o ambiente em que este agente opera seja não-determinístico, o que significa que tomar duas vezes a mesma ação, quando no mesmo estado mas em instantes diferentes, pode resultar em diferentes valores de reforços. Assume-se também, normalmente, que o ambiente seja estacionário, ou seja, que as probabilidades de fazer transições de estado ou receber reforços específicos não mudem ao longo do tempo, ou que pelo menos as probabilidades variem lentamente (caso em que algoritmos específicos ainda conseguem tratar).

Devido a estas características, a maneira mais tradicional de formalização dos problemas de

Aprendizado por Reforço é dada através do conceito de Processos de Decisão Markovianos.

3.2 Processos de Decisão Markovianos

Processos de Decisão Markovianos (MDP) são processos que obedecem a condição de Markov. Resumidamente, esta condição diz, segundo *Russell e Norvig* (2009), que o estado futuro s_{t+1} de um sistema é uma função que depende exclusivamente do estado do sistema no instante atual s_t, de n estados $S_p = \{s_{t-1}, s_{t-2}, ..., s_{t-n}\}$ e da ação tomada no estado atual, isto é, o estado futuro depende do estado atual e de um número *finito* de estados passados. A condição de Markov é importantíssima exatamente por impor este limite finito às relações de dependência temporal de um processo, tornando o sistema tratável.

Formalmente (*Littman*, 1994; *Kaelbling et al.*, 1996), um MDP pode ser definido pela tupla $\langle S, A, R, T \rangle$, onde S é o conjunto de estados, A o de ações, $R(s, a)$ é o reforço imediato dado pela execução da ação a no estado s e a função de transição $T(s, a, s')$ a probabilidade de ir do estado s ao estado s' executando a ação a. Uma política π para M é um mapeamento de S para A.

Um MDP pode ser determinístico ou não. Se, dado o estado atual s e a função de probabilidade de transição $T(\cdot)$, a seleção de uma ação a resulta sempre no mesmo estado futuro s', ou seja, a transição $T(s, a, s')$ ocorre com probabilidade 1, então o MDP é determinístico. Se, por outro lado, a seleção da ação a no estado s poder resultar em estados diferentes, caso em que a transição $T(s, a, s')$ é representada por uma distribuição de probabilidades $P(s'|s, a)$, então o MDP é não-determinístico.

3.3 Componentes do Aprendizado por Reforço

Segundo *Kaelbling et al.* (1996), um problema de RL é composto por quatro componentes: política, função de recompensa, função valor e função de transição de estados.

Políticas são responsáveis por selecionar a ação do agente, seja qual for o estado do agente. Cada política π indica uma sequencia de ações ao agente, de modo que ele alcance o objetivo. Uma política pode determinar uma sequencia de ações que alcance o objetivo, mas cuja recompensa acumulada ao longo do tempo não seja a máxima possível. Estas politicas são as chamadas sub-ótimas. À política π^*, que maximiza o ganho de recompensas acumuladas, é dado o nome política ótima.

O objetivo do agente é representado através da função de recompensa $R(\cdot)$. Esta função deve retornar um valor escalar que represente uma gratificação ou punição para cada par estado-ação possível.

Enquanto a função recompensa representa a resposta imediata do ambiente à ação executada pelo agente, a função valor $V(\cdot)$, também chamada função *valor cumulativo esperado*, 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 $T(\cdot)$ é capaz de retornar o estado futuro, seja qual for o estado atual e a ação executada.

3.4 Modelos de Comportamento Ótimo

Em um problema de RL, é importante definir como o agente leva em conta o futuro em suas decisões atuais, já que esta definição impactará diretamente como o modelo de comportamento ótimo será. Dois modelos são utilizados na maioria da literatura existente sobre o tema. O mais simples é o modelo de *horizonte finito*, pelo qual, em qualquer dado instante o agente deve tentar maximizar a recompensa esperada para n passos no futuro.

O mais utilizado na prática, no entanto, é o modelo de *horizonte infinito descontado*, que leva em conta todo o futuro do agente, porém com as recompensas recebidas no futuro geometricamente descontadas de um fator γ (com $0 \leq \gamma < 1$), como mostra a Eq. (3.1)

$$\sum_{t=0}^{\infty} \gamma^t r_t \tag{3.1}$$

Uma vez definido como o futuro será tratado, pode-se então pensar em algoritmos para modelar o comportamento, ou política, ótima de um agente em um problema de RL modelado por um MDP. Esta política π^* deve ser tal que maximize a função valor $V^{\pi}(s)$ para qualquer estado s. A Eq. (3.2) mostra esta função, para o caso em que o modelo do horizonte infinito descontado é o adotado.

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{3.2}$$

onde:

- r_{t+i} é a sequencia de reforços recebidos a partir de s_t, usando a política π repetidamente

para selecionar ações,

- γ é o fator de desconto, com $0 \leq \gamma < 1$.

Fundações teóricas importantes para o cálculo da política ótima vem da programação dinâmica, através do princípio de Bellman (*Bertsekas*, 1987). Este princípio afirma que, dada uma política ótima $\pi^* = \{\mu_0^*, \mu_1^*, ..., \mu_n^*\}$ para um problema de controle, a política $\{\mu_i^*, \mu_{i+1}^*, ..., \mu_n^*\}$ também é ótima para o sub-problema de estado inicial s_i $(0 < i < n)$. A interpretação deste princípio é que tomar a ação que leve ao melhor estado imediatamente futuro de forma recursiva equivale a seguir a política ótima.

Dois métodos utilizados para solucionar o problema do Aprendizado por Reforço baseados em programação dinâmica são os algoritmos de *iteração de valor* e *iteração de política*.

O algoritmo de iteração de valor (*Bertsekas*, 1987) consiste no uso da Eq. (3.3), que realiza o cálculo iterativo da função valor ótima, e Eq. (3.4), responsável por encontrar a política ótima π^*.

$$V^*(s_t) = \max_{a_t} \left[r(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} T(s_t, a_t, s_{t+1}) V^*(s_{t+1}) \right] \tag{3.3}$$

$$\pi^*(s_t) \leftarrow arg\max_{a_t} \left[r(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} T(s_t, a_t, s_{t+1}) V^*(s_{t+1}) \right] \tag{3.4}$$

onde:

- s_t é o estado atual,
- a_t é a ação realizada em s_t,
- s_{t+1} é o estado resultante ao aplicar-se a ação a_t enquanto no estado s_t,
- $V_{s_t}^*$ é a função valor ótima e $V_{s_t+1}^*$ o valor da função valor ótima na iteração seguinte.

Por outro lado, o algoritmo de iteração de política (*Bertsekas*, 1987; *Kaelbling et al.*, 1996) manipula a política diretamente, sem a necessidade do uso da função valor ótima. Este algoritmo é composto pelas equações lineares Eq. (3.5) e Eq. (3.6), chamadas *passo de avaliação da política* e *passo de melhora da política*, respectivamente. Sua operação consiste em escolher

uma política inicial $\pi = \{a_0, a_1, ...\}$ e então calcular V^π para esta política (Eq. (3.5)). Verifica-se então a possibilidade de melhorar o valor V$^\pi$, mudando apenas a primeira ação tomada, a_t (Eq. (3.6)). Este passo, comprovadamente, só pode melhorar o desempenho da política.

$$V^\pi(s_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} T(s_t, a_t, s_{t+1}) V^\pi(s_{t+1}) \tag{3.5}$$

$$\pi'(s_t) \leftarrow arg\max_{a_t} \left[r(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} T(s_t, a_t, s_{t+1}) V^\pi(s_{t+1}) \right] \tag{3.6}$$

Porém, a política π' determinada pela Eq. (3.6) é uma política gulosa (*greedy*), pois escolhe sempre a ação imediata que maximiza o retorno esperado de recompensas. Um algoritmo guloso como este cria uma limitação: uma amostra inicial ruim pode levar o sistema a estabilizar em um máximo local. Uma solução é utilizar outros tipos de algoritmos, como a exploração aleatória. Na exploração aleatória, conhecida como ϵ-*greedy*, o agente escolhe, com probabilidade p, a melhor ação, ou uma ação aleatória com probabilidade $1 - p$, permitindo assim que quando não houver mais mudanças possíveis, $\pi' = \pi^*$, a política seja ótima.

Contudo, tanto o algoritmo de iteração de valor quanto o de política tem como pré-requisito o conhecimento do modelo de transição de estados T e do modelo de recompensas R. Como em diversos problemas estes modelos não são conhecidos, a alternativa é utilizar uma classe de algoritmos *livres de modelo*, conhecida como Aprendizado por Reforço. As próximas seções mostram alguns destes algoritmos de RL.

3.5 Método das Diferenças Temporais - TD-Learning

O Método das Diferenças Temporais TD(λ) (*Sutton*, 1988), um dos primeiros algoritmos de Aprendizado por Reforço com base teórica consistente, é uma versão adaptativa do algoritmo de iteração de política. Esse método também calcula uma estimativa $\hat{V^\pi}$ do valor acumulado V^π, selecionando as ações seguindo uma política π, de forma iterativa. Porém, para atualizar a estimativa $\hat{V^\pi}$, o método TD exige apenas que uma quantidade (dependente do valor de λ) de passos futuros s_{t+k} sejam observados, sem a necessidade de um modelo do ambiente.

As equações abaixo mostram o método de cálculo do TD(λ). A Eq. (3.7) é semelhante à equação do algoritmo de iteração de valor (Eq. (3.3)), porém, na equação do TD(λ), o valor de $T(s_t, a_t, s_{t+1})$ é amostrado do mundo real e não da simulação de um modelo.

$$V_{t+1}(u) \leftarrow V_t(u) + \alpha\delta^{\lambda}e(u) \tag{3.7}$$

onde:

- u é o estado sendo analisado,
- $r(s_t, a_t)$ é a recompensa por realizar a ação a_t no estado s_t,
- γ é o fator de desconto $(0 \leq \gamma < 1)$,
- α é a taxa de aprendizado $(0 < \alpha < 1)$,
- δ^{λ} é a diferença temporal, definida na Eq. (3.9),
- $e(u)$ é a elegibilidade do estado.

A diferença temporal, também chamada erro TD, é a diferença entre o valor estimado $\hat{V}_t(s_t)$ e o valor acumulado esperado $r(s_t, a_t) + \gamma V_t(s_{t+1})$. O cálculo desta diferença consiste em obter δ^0 (Eq. (3.8)) e então resolver a Eq. (3.9), que é a projeção de δ no futuro.

$$\delta_t^0 = r(s_t, a_t) + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{3.8}$$

$$\delta_t^{\lambda} = \delta_t^0 + \sum_{n=0}^{\infty} (\gamma\lambda)^n \delta_{t+n}^0 \tag{3.9}$$

No entanto, esta equação não é causal, já que utiliza os erros TD futuros na atualização do erro atual. Para resolver este problema utiliza-se o termo de elegibilidade.

O termo de elegibilidade permite que estados visitados anteriormente também recebam reforços, com os visitados mais recentemente recebendo valores mais influenciados pelo reforço recebido no estado atual. *Sutton* (1988) chama este termo de *rastro de elegibilidade*, implementado no algoritmo na forma de um vetor que armazena a elegibilidade de cada estado, conforme mostra a Eq. (3.10). Neste vetor, valores de elegibilidade maiores indicam que o estado foi visitado mais recentemente. Esta implementação é chamada rastro de elegibilidade por acumulação.

$$e(u) = \begin{cases} \gamma\lambda e(u) + 1 & \text{se } u = s_t \\ \gamma\lambda e(u) & \text{caso contrário} \end{cases} \tag{3.10}$$

onde:

- λ é o fator de desconto para as diferenças temporais,
- γ é o fator de desconto para as recompensas futuras,
- s_t é o estado atual.

Singh e Sutton (1996) argumentam, no entanto, que a implementação do rastro de elegibilidade por acumulação pode ser tendenciosa, e criam o termo de elegibilidade por substituição, conforme a Eq. (3.11).

$$e(u) = \begin{cases} \gamma\lambda 1 & \text{se } u = s_t \\ \gamma\lambda e(u) & \text{caso contrário} \end{cases} \tag{3.11}$$

Uma importante propriedade do TD é que sua convergência ao valor ótimo da política π, V^{π^*}, é garantida (*Sutton*, 1988) desde que a taxa de aprendizado decaia lentamente e a política seja mantida fixa.

No TD, λ pode variar de 0, o que significa utilizar apenas o estado imediatamente no futuro, até 1, que significa utilizar todos os estados futuros. Muitas implementações utilizam o TD(0), pela simplicidade. No entanto, embora mais custoso computacionalmente, valores de λ maiores que zero podem fazer com que o algoritmo tenha convergência mais rápida do que o TD(0).

Muitos outros métodos de RL são baseados no método das diferenças temporais, como o Q-Learning, onde a regra de aprendizado é um caso especial da regra do TD(0).

3.6 Q-Learning

O Q-Learning (*Watkins*, 1989; *Watkins e Dayan*, 1992) é um algoritmo que não necessita dos modelos de transição de estados ou de recompensas e, apesar de ser de simples implementação, tem um forte embasamento teórico. Estas características fazem dele um dos mais populares algoritmos aplicados ao Aprendizado por Reforço.

Neste método o agente deve, ao invés de maximizar V, aprender uma função de recompensa esperada com desconto conhecida como função valor-ação, ou simplesmente como função Q. Esta função é a soma da recompensa recebida pelo agente por, no instante t, estar no estado s_t e realizar a ação a_t, mais o valor de seguir a política ótima daí em diante. Por ser $V^*(s)$ o resultado da função valor ótima assumindo que a melhor ação foi tomada no estado inicial s, decorre que $V^*(s) = \max_a Q^*(s, a)$, e a equação Q^* pode ser escrita como mostra a Eq. (3.12).

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s_t+1 \in S} T(s_t, a, s_{t+1}) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \tag{3.12}$$

Outra inferência importante oriunda da igualdade $V^*(s) = \max_a Q^*(s,a)$ é que a política ótima pode ser obtida diretamente (Eq. (3.13)).

$$\pi^*(s_t) = arg \max_{a_t} Q^*(s_t, a_t) \tag{3.13}$$

Já que a função Q seleciona a ação explicitamente, os valores $\hat{Q}$ podem ser estimados durante a execução por um método igual ao TD(0), conforme a regra de aprendizado descrita pela Eq. (3.14), com probabilidade 1 de convergência para Q^*. A garantia da convergência faz parte da fundamentação teórica do algoritmo, porém existem condições para tal: cada par estado-ação deve ser visitado um número infinito de vezes e a recompensa deve ser limitada para que não possa tender a um valor infinito.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s,a)) \tag{3.14}$$

onde:

- α $(0 < \alpha < 1)$ é a taxa de aprendizado. Uma definição normalmente usada para esta taxa envolve decair α com o número de visitas passadas ao estado s em que a ação a foi escolhida, $\alpha = (1 + visitas(s,a))^{-1}$.

Um dilema do Q-Learning é o compromisso entre exploração e explotação (ou aproveitamento, em uma tradução menos literal do termo original). No início do algoritmo, quando os valores Q são inicializados, deve-se explorar o ambiente, tomando ações novas. Já quando os valores estão quase convergidos para seus valores ótimos, deve-se explotar o que foi aprendido pela função, escolhendo a ação com o maior valor de Q em cada situação. Uma estratégia para escolha de ações comumente empregada para resolver este dilema é a exploração aleatória, ϵ-*greedy*. Uma vantagem do Q-Learning é que, embora haja esse compromisso de balancear exploração e explotação, desde que se visite cada par estado-ação um número suficiente de vezes o algoritmo conseguirá convergir para os valores Q^*, não importando qual a estratégia de exploração utilizada.

Embora o Q-learning seja efetivo em aprender por recompensas acumuladas, o algoritmo não trata de nenhum dos problemas relacionados à existência de grandes espaços de estados ou

ações. Além disso, em muitos casos, converge de forma bastante lenta para uma boa política.

3.7 Q(λ)

O Q(λ) é uma extensão do Q-learning que adiciona a capacidade de propagação temporal das atualizações ao algoritmo, como no TD(λ). Foi criado com o objetivo de acelerar o Q-learning original através da propagação das experiências de um estado para outros, visitados recentemente. Duas abordagens conhecidas, que combinam o Q-learning e o TD(λ), foram propostas por *Watkins* (1989) e por *Peng e Williams* (1996).

Para que as diferenças temporais futuras sejam consideradas no algoritmo Q(λ), a regra de atualização do Q-learning é combinada à equação do TD(λ) (Eqs. (3.14) e (3.9)), originando a regra de atualização do Q(λ), Eq. (3.15).

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta^{\lambda} \tag{3.15}$$

Da mesma forma que no TD(λ), a equação acima não é causal. Para que seja utilizada, deve-se recorrer então ao uso do traço de elegibilidade, ajustando-o para considerar o par estado-ação. Porém, existem diferentes maneiras de implementar o rastro de elegibilidade. *Watkins* (1989) reinicializa o rastro toda vez que uma ação é selecionada aleatoriamente, já que a prova de convergência para a solução ótima do algoritmo depende que uma estratégia de seleção de ações gulosa seja utilizada. Por outro lado, a implementação de *Peng e Williams* (1996) não faz essa diferenciação, assumindo que a estratégia de seleção seja ϵ-*greedy*.

Sutton e Barto (1998) dizem que, para uma política não gulosa, o algoritmo de *Peng e Williams* (1996) não converge nem para Q^{π} nem para Q^{*}, mas para algo entre as duas. No entanto, para uma política que se torne gulosa ao longo do tempo, este método converge para Q^{*} e apresenta desempenho significativamente melhor do que a versão de *Watkins* (1989).

3.8 Q-Learning Acelerado por Heurísticas - HAQL

Bianchi et al. (2004, 2008) propõe o Q-learning Acelerado por Heurísticas (*Heuristically Accelerated Q-learning*), uma extensão do Q-learning capaz de acelerar a convergência do algoritmo através da utilização de conhecimento existente sobre o domínio, expresso por meio de heurísticas.

Formalmente, um algoritmo de RL acelerado por heurísticas visa solucionar um problema modelado por um MDP com o uso explícito de uma função heurística. Esta função heurística, expressa por $H : S \times A \rightarrow \Re$, tem como objetivo influenciar a escolha das ações a que o agente deve tomar, em cada estado s.

Uma propriedade importante desta classe de algoritmos é que a função heurística pode ser modificada ou adaptada em tempo de execução. Pode-se, por exemplo, utilizar informação conhecida *a priori* ou do estágio inicial de aprendizado para acelerá-lo. Da mesma forma, informação inicial ruidosa ou incorreta pode ser removida da função heurística, adaptativamente.

Bianchi (2004) também apresenta um método para extração automática da função heurística em tempo de execução, chamado *retropropagação de heurísticas*. Este método propaga, a partir de um estado final, as políticas corretas que levam a este estado. Em seguida, ela é propagada para os estados antecessores dos que já possuem uma heurística definida, criando assim um algoritmo recursivo. Pelo teorema de Bellman, prova-se que o algoritmo leva à política ótima.

Características também importantes desta classe são a preservação das propriedades do algoritmo de RL original e a versatilidade de uso de operadores matemáticos. A primeira característica ocorre porque a heurística é utilizada somente para a escolha da ação, o que significa apenas uma mudança na forma em que a exploração do espaço acontece. *Bianchi* (2004) demonstra que as propriedades de convergência dos algoritmos de RL continuam garantidas após a inserção da função heurística, o que abre possibilidades de uso da aceleração por heurísticas em diversos algoritmos de RL. A segunda característica é que a inserção de H pode ser feita utilizando qualquer função matemática (desde que o conjunto resultante suporte a operação de maximização), como adição, subtração ou multiplicação.

Por influenciar a escolha de ações, indicando que uma ação deve ser tomada em detrimento das outras, pode-se dizer que a função heurística define uma política heurística, ou seja, uma política tentativa usada para acelerar o aprendizado. Por esta razão, no HAQL, a definição da política ótima passa a ser definida pela Eq. (3.16), que é a equação do Q-Learning original (Eq. (3.13)), com o termo H, referente à função heurística, inserido.

$$\pi^*(s_t) = arg\max_{a_t} \left[Q^*(s_t, a_t) + \xi H_t(s_t, a_t)^\beta \right] \tag{3.16}$$

onde:

- ξ e β são valores reais usados para controlar a influência da função heurística.

O erro na aproximação da função valor, causado pelo uso de heurísticas em algoritmos de RL, é definido conforme a Eq. (3.17)

$$L_H(s) = Q(s, \pi^*(s)) - Q(s, \pi^H(s)), \forall s \in S \tag{3.17}$$

onde:

- π^* é a política ótima e π^H é uma política heurística.

Para que a heurística não interfira na convergência do algoritmo para a política ótima, este erro L_H deve ser controlado. *Bianchi* (2004) demonstra que é possível alcançar este objetivo, garantindo limites máximos e mínimos para os valores da função heurística, conforme a Eq. (3.18).

$$L_H(s) \leq \xi \left[h^{\beta}_{max} - h^{\beta}_{min} \right], \forall s \in S \tag{3.18}$$

No caso do Q-Learning, os valores limite h_{min} e h_{max} apropriados devem ser anulados ou superados pelos valores da estimativa de recompensas futuras, que no caso do HAQL é igual a $\hat{Q}(s, a)$, ou o algoritmo não convergirá para a política ótima. A Eq. (3.19) mostra o valor máximo do erro L_H, para este algoritmo. O erro mínimo pode ser obtido de maneiro análoga.

$$\begin{aligned} L_H(s) &= \xi \left[h^{\beta}_{max} - h^{\beta}_{min} \right] \\ &= \xi \left[\frac{r_{max}}{1-\gamma} - \frac{r_{min}}{1-\gamma} + \eta - 0 \right] \\ &= \xi \left[\frac{r_{max} - r_{min}}{1-\gamma} + \eta \right] \end{aligned} \tag{3.19}$$

3.9 Generalização e Aproximação em RL

Algoritmos de RL comumente se baseiam no conceito de funções valor, responsáveis por indicar, para cada política, o valor do reforço proveniente de um determinado estado ou par estado-ação. Para pequenos problemas, esta função pode ser representada por uma tabela, mas em problemas mais complexos como os encontrados em aplicações reais, o uso do formato tabular torna o problema intratável.

Para resolver esta limitação, o agente deve ser capaz de generalizar as experiências adquiridas durante sua exploração do espaço. Esta intratabilidade não é consequência somente dos

requisitos computacionais mas também pela necessidade de coleta de um número de experiências elevado, que o agente não dispõe de tempo ou recursos para efetuar (caso frequente em aplicações com robôs reais). Por isso, o uso de métodos de generalização também está ligado à questão da aceleração do aprendizado, já que espaços menores permitem aprendizado mais rápido.

A generalização pode ocorrer através do espalhamento das experiências entre os estados, pelo uso de aproximadores de função ou pela combinação de ambos. Por exemplo, *Ribeiro et al.* (2002) utiliza espalhamento segundo similaridade espacial, o que significa estender a atualização do estado atual para estados que sejam semelhantes à ele, segundo algum critério. Já o Q(λ) (*Watkins*, 1989), descrito na Seção 3.7, utiliza espalhamento temporal, onde o reforço do estado atual é estendido para os estados visitados recentemente.

O uso de aproximadores de função, como redes neurais, funções de base radial e CMACs, também é bastante popular em aplicações reais. O TD-Gammon (*Tesauro*, 1995) é um exemplo extremamente bem sucedido do uso de redes neurais perceptron em conjunto com algoritmos TD.

Contudo, a literatura também traz falhas de aproximadores de função em domínios simples, o que evidencia que a correta especificação do aproximador é fundamental. *Whiteson e Stone* (2006), por exemplo, propõem o uso de algoritmos evolucionários para automaticamente selecionar a representação correta da rede neural utilizada como aproximador em um algoritmo TD.

3.9.1 CMAC

O *Cerebellar Model Articulation Controller* (CMAC), proposto por *Albus* (1975), é um aproximador de função, pertencente à classe das redes neurais, cujo funcionamento foi modelado a partir da estrutura e funcionamento do cerebelo dos mamíferos. Utilizado inicialmente para modelar o controle de motores em manipuladores robóticos, se tornou bastante usado em aprendizado de máquina, para classificação automatizada, e em Aprendizado por Reforço. Suas principais vantagens são o baixo custo de processamento e o aprendizado extremamente rápido, com tempos várias ordens de magnitude menores do que uma rede perceptron ou de base radial.

As implementações mais usuais de CMAC em aplicações de RL (*Watkins*, 1989; *Sutton*, 1996) envolvem o uso de CMACs com algoritmos de codificação por telhas (*tile coding*) e *hashing*. A codificação por telhas é uma técnica de discretização de variáveis contínuas que

particiona o espaço de entrada em várias camadas, chamadas *telhados* (*tilings*), formadas por variáveis binárias, chamadas telhas (*tiles*). A variação no tamanho e formato destas telhas controla o nível de quantização do espaço de entrada, pois define o número de estados vizinhos que terão telhas em comum, enquanto o número de camadas (telhados) sobrepostas controla o grau de generalização da resposta, de forma similar ao utilizado em redes perceptron. Da mesma forma que um maior número de neurônios na camada escondida de uma rede perceptron aumenta a capacidade de generalização da rede, um maior número de camadas também tem este efeito.

Mesmo com o uso da codificação em telhas, CMACs ainda são afetados pelo mal da dimensionalidade, e requerem grandes quantidades de memória. O uso de tabelas *hash* elimina este mal, já que na maioria dos problemas com grandes espaços de estados, apenas um número reduzido de estados é realmente visitado. Como o *hashing* é feito apenas quando os estados são visitados pela primeira vez, os requisitos de memória do algoritmo são significativamente reduzidos, e o impacto no desempenho é normalmente imperceptível.

Um CMAC é composto, em um primeiro estágio, por um algoritmo de *tile coding* de C camadas e por um vetor de pesos W. O algoritmo de *tile coding* é utilizado para cobrir o espaço de entrada U, como mostra a Figura 3.1. As telhas do primeiro estágio são conectadas ao vetor de pesos W, responsável por armazenar o valor com que cada telha, quando ativada, deve contribuir para o valor da saída. Esta conexão cria um mapeamento de telhas para pesos em que cada telha se conecta a uma, e somente uma, posição de W. O vetor W é responsável também pelo aprendizado dos CMACs, durante a fase de treinamento. Seu ajuste é feito de forma supervisionada, similar ao procedimento utilizado em redes perceptron. Também como nestas redes, métodos de gradientes descendentes como a regra Delta podem ser utilizados para o ajuste dos pesos.

A saída de um CMAC é calculada da seguinte forma: cada entrada u ativa uma telha a, em cada camada C. O conjunto de telhas ativadas $a(u)$ transporta os pesos w a elas mapeados para o estágio de saída, onde estes C pesos são somados, resultando no valor final de saída, conforme a Eq. (3.20).

$$y(u) = a(u)^T w \tag{3.20}$$

onde:

- $a(u)$ é o vetor de associação binário para a entrada u. Telhas ativadas em cada camada

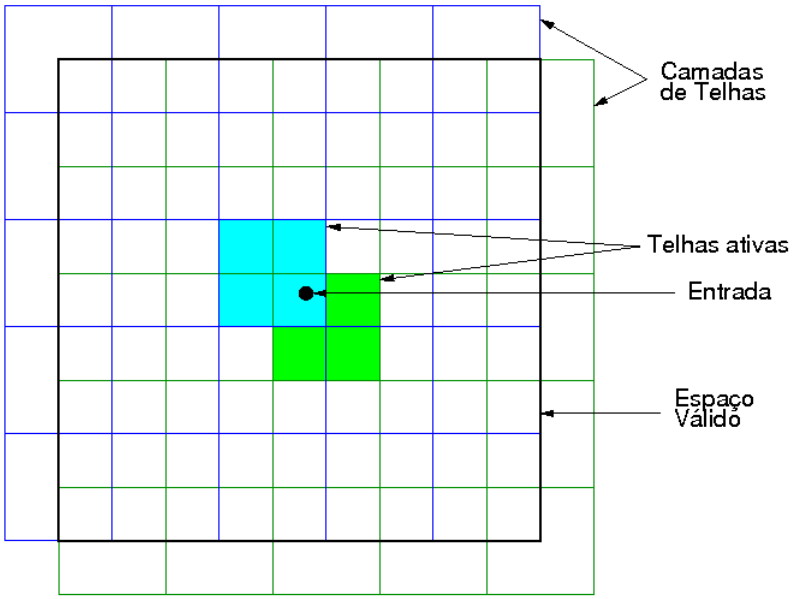


Figura 3.1: Exemplo das camadas de telhas em um CMAC (*Bianchi*, 2004).

tem valor 1, enquanto telhas inativas tem valor 0,

- w é o vetor de pesos,
- $y(u)$ é a saída da rede para a entrada u.

Conforme mencionado, CMACs tem baixo custo de processamento e convergem muito rapidamente, porém seu projeto requer que um especialista decida sobre qual a representação correta de cada uma das variáveis de estado, seu nível de quantização, enquanto redes neurais perceptron, por exemplo, conseguem aprender as relações não lineares entre as variáveis durante o treinamento.

3.10 Discussão

Neste capítulo foram descritos os fundamentos do Aprendizado por Reforço e suas técnicas clássicas, o método das diferenças temporais e o Q-Learning, bem como conceitos gerais sobre generalização e aproximadores de função. O capítulo apresentou também a proposta de *Bianchi et al.* (2004) para aceleração do aprendizado através de heurísticas, um método com grandes possibilidades de aplicação em um ambiente como o proposto, de difícil modelagem, dinâmico e que necessita de agentes capazes de toma decisões rapidamente.

A literatura sobre Aprendizado por Reforço é extensa, e neste capítulo não houve a pretensão de cobrir uma área tão vasta, mas apenas oferecer uma visão dos tópicos relevantes à este trabalho. Excelentes referências podem ser encontradas em publicações como *Kaelbling et al.* (1996), *Sutton e Barto* (1998) ou *Busoniu et al.* (2008) (que aborda Aprendizado por Reforço em sistemas multiagentes).

Trabalhos recentes apresentam métodos engenhosos e resultados interessantes da aplicação de RL em ambientes robóticos reais (*Martinson e Arkin*, 2003; *Dahl et al.*, 2006; *Whiteson e Stone*, 2006; *Latzke et al.*, 2007), mostrando que a solução de problemas complexos por RL é cada vez mais uma realidade. Mostra também que RL é adequado para uso em conjunto com diversas técnicas, como sistemas fuzzy, redes neurais, algoritmos genéticos e evolucionários, e que há uma tendência em favor do uso conjugado de RL com estas outras técnicas. Exemplos aplicados em ambientes similares ao deste trabalho, como *Stone et al.* (2005), *Kalyanakrishnan et al.* (2007), que utilizam o futebol de robôs simulado, também apresentam bons resultados.

Para a aplicação específica deste trabalho, o futebol de robôs reais, os resultados presentes na literatura confirmam a possibilidade de implementar soluções baseadas em RL. No entanto, como em qualquer ambiente real complexo, as escolhas feitas durante o projeto da solução são cruciais para seu sucesso, como quais os métodos de generalização e a escolha de uma representação que seja, ao mesmo tempo, significativa e o mais compacta possível, já que uma deficiência do Aprendizado por Reforço, a dificuldade em explorar grandes espaços de estados o suficiente para que possam aprender, se agrava em robôs reais, que têm limitações de bateria e capacidade de operação contínua e prolongada.

Capítulo 4

ARQUITETURA DE ALOCAÇÃO DE TAREFAS UTILIZANDO APRENDIZADO POR REFORÇO

Este trabalho descreve a implementação de uma arquitetura baseada em modelos econômicos (como os vistos na seção 2.1) para a alocação de tarefas entre os robôs de uma equipe de futebol de robôs. Nesta equipe, cada um dos robôs deve utilizar Aprendizado por Reforço para descobrir os valores ótimos de suas funções utilidade em relação às tarefas, ou seja, deve aprender sua aptidão em executá-las e em qual situação tais aptidões são úteis.

As tarefas a serem alocadas são os comportamentos existentes na camada mais alta do sistema de estratégia, chamados papéis.

Exitem diversas razões para a escolha de algoritmos baseados em modelos econômicos. A literatura sobre MRTA mostra extensivo estudo deste tipo de algoritmo, com resultados bastante positivos e comprovação de seu embasamento teórico e eficiência computacional. Este trabalho não tem como objetivo, no entanto, mostrar uma eventual superioridade dos sistemas econômicos sobre, por exemplo, os comportamentais, sociais ou ainda sobre outras técnicas de MRTA. De fato, nesta proposta, como em outros sistemas recentes (*Tang e Parker*, 2007; *Kose et al.*, 2004; *Parker e Tang*, 2006), existem características provenientes de mais de uma das técnicas de MRTA.

Já a utilização de RL se justifica pela possibilidade de utilizá-lo na forma de funções utilidade, uso que, embora ainda pouco explorado, parece promissor. Além disto, a capacidade do RL de aprender por experiência, mesmo na ausência de modelos do ambiente, se encaixam bem em problemas de MRTA.

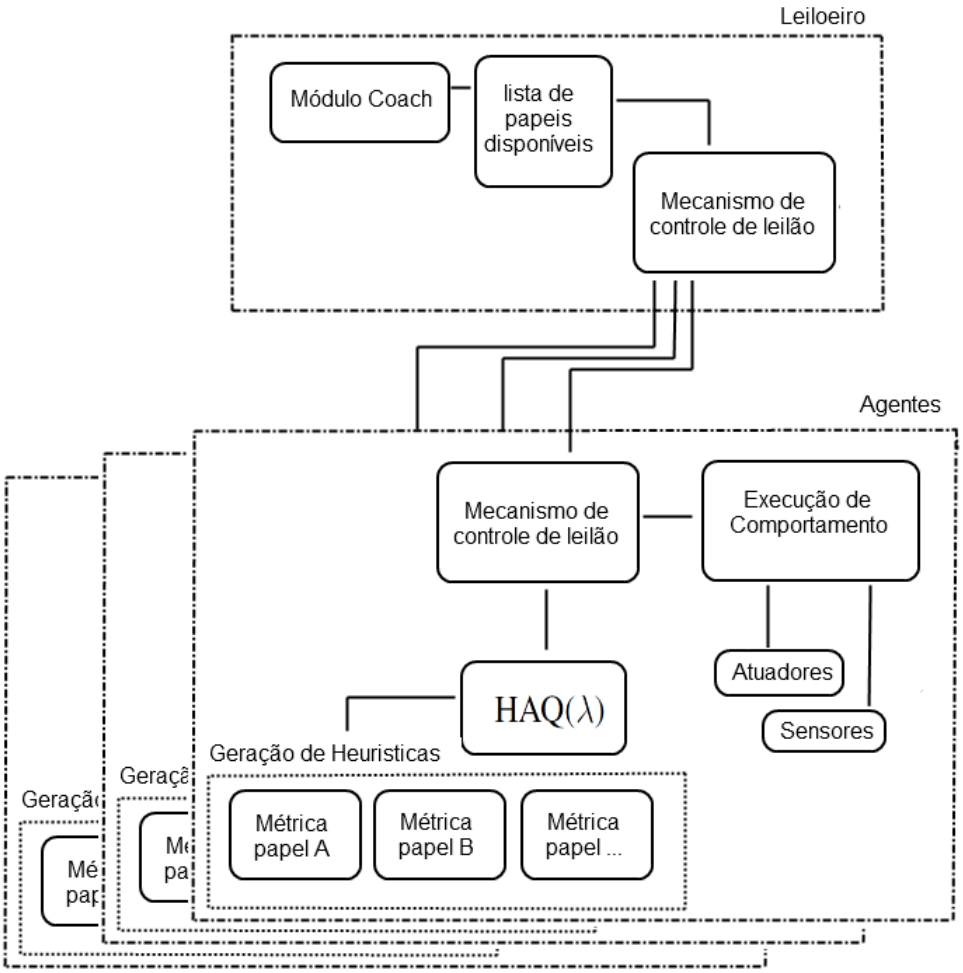


Figura 4.1: Diagrama da arquitetura implementada, ilustrando os componentes dos agentes participantes e do leiloeiro.

Este capítulo começa apresentando um resumo e um diagrama de módulos da arquitetura implementada. Em seguida, apresenta a estrutura do sistema de estratégia, que serve como conjunto de tarefas a serem alocadas. São detalhados então os algoritmos econômicos e de Aprendizado por Reforço que formam a arquitetura, e sua operação conjunta em detalhes.

4.1 Funcionamento Resumido da Arquitetura de MRTA

O funcionamento da arquitetura de alocação de tarefas, cujo diagrama de módulos aparece na Figura 4.1, é o seguinte: a cada ciclo de tempo do algoritmo, o módulo *Coach*, responsável por dizer quais papéis estão disponíveis e sua ordem de prioridade, envia a listagem destes papéis ao mecanismo de leilão. Estes papéis são, então, leiloados aos agentes, em sequência, começando pelo de maior prioridade. Os agentes robóticos, por sua vez, consultam a representação de suas funções Q para escolher o papel que têm interesse em dar uma oferta. O ganhador recebe seu papel e o processo se repete, sem este agente, até que todos tenham recebido al-

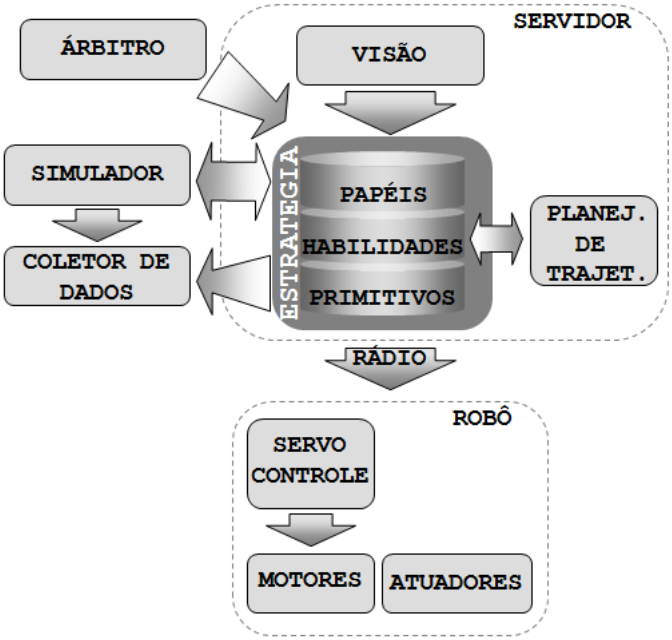


Figura 4.2: Diagrama dos módulos utilizados na equipe de futebol de robôs. A camada de papéis, dentro do módulo de estratégia, contém a lista de tarefas para a arquitetura de MRTA.

gum papel. Em paralelo, também a cada ciclo, o algoritmo de RL distribui reforços e computa atualizações de valores.

4.2 Sistema de Estratégia da Equipe

A estrutura do sistema de estratégia utilizado pela equipe RoboFEI, do qual o autor deste trabalho é também co-autor, pode ser visto no diagrama dos módulos da Figura 4.2. Este sistema estrutura diferentes níveis de abstração em diferentes camadas, abordagem bastante popular em sistemas multi-robôs que utilizam técnicas de aprendizado de máquina (*Matarić*, 2001; *Bowling et al.*, 2004; *Browning et al.*, 2005). Três camadas compõe o sistema. Na mais baixa, chamada camada de *primitivas*, estão as ações individuais mais simples , de curta duração e dependentes fortemente dos atuadores e sensores do robô. São funções como deslocar-se a um determinado ponto, rotacionar, ativar o dispositivo de drible ou de chute.

Sobre a camada de primitivas encontra-se a camada de *habilidades*. Habilidades são também ações de curta duração, mas envolvem o uso de uma ou mais primitivas e computação adicional, como cálculo de ângulos, estimativas de velocidade, predição da posição de obstáculos ou monitoramento do estado de uma ação primitiva. Exemplos de habilidades são ações do tipo chutar ao gol (mirando no ponto com maior probabilidade de sucesso), passar a bola para outro robô ou driblar um oponente.

No nível mais alto da estratégia está a camada de *papéis*, responsável por implementar os comportamentos que os robôs podem assumir durante o jogo. Cada papel é formado por uma combinação de diversas habilidades e pela lógica que coordena a execução, de forma a exibir o comportamento desejado. Exemplos de papéis são as funções de atacante, defensor, atacante ofensivo, meio-campista ou líbero. A seção 4.3 descreve em maiores detalhes as habilidades e papéis, existentes e a serem implementadas como parte deste trabalho.

O sistema de MRTA com aprendizado desenvolvido durante este trabalho é aplicado à camada de papéis do sistema de estratégia, portanto, ao longo do restante deste trabalho, deve-se entender o termo *papel* como a tarefa que um robô deverá desempenhar na equipe. Estes papéis são os itens oferecidos aos robôs pelos métodos econômicos. O algoritmo de RL é o responsável por ensinar aos robôs quais os valores de suas funções utilidade quanto a executar cada papel, ou seja, quão apto o robô é para desempenhar cada atribuição, assim permitindo que a seleção do que lhe for mais atrativo no momento. Utilizando uma notação de *Gerkey e Matarić* (2004), este é um sistema ST-SR-IA (*Single Task - Single Robot - Instantaneous Assignment*), ou seja, cada robô executa apenas uma tarefa por vez, cada tarefa deve ser executada por exatamente um robô e as alocações são de curta duração, válidas apenas até o próximo ciclo do algoritmo de alocação de tarefas.

4.3 Habilidades e Papéis

Embora a arquitetura de MRTA implementada opere na camada de papéis, onde o comportamento dos agentes é definido, para que seja efetiva é fundamental que as ações da camada de habilidades, que compõem estes papéis, sejam também corretamente projetadas. Em uma analogia com um jogador humano, pode-se entender que é necessário aprender a chutar, passar e bloquear antes de aprender um posicionamento tático. Esta seção começa pela descrição do conjunto de ações da camada de habilidades, criadas em conjunto pelo autor deste trabalho e outros membros da equipe de software do projeto RoboFEI, e então passa a descrever a camada de papéis, onde a alocação de tarefas ocorre.

4.3.1 Habilidades

Habilidades expressam as ações de curta duração que um agente executa, porém não necessariamente são simples ou determinísticas. Um bom exemplo de ação, que ilustra estas duas

características, é passar a bola. Para passar a bola, o agente deve avaliar a situação de cada um de seus companheiros de equipe e então maximizar uma função com ao menos dois parâmetros interdependentes: qual companheiro está melhor posicionado e para qual deles a probabilidade de sucesso do passe é maior. É fácil observar a natureza não-determinística da ação de passar a bola, pois a interferência dos oponentes e imprecisões naturais do sistema fazem com que sua execução repetidas vezes produza resultados diferentes. Habilidades devem ser também modulares, para que sejam reutilizáveis por diversos papéis.

Para satisfazer estes requerimentos, bem como criar habilidades suficientes para representar as ações individuais mais comuns de um agente durante a partida, as seguintes habilidades foram criadas:

ShootToGoal( ) - Chuta ao gol. Seu funcionamento envolve selecionar o melhor ângulo para o chute, levando em consideração os adversários, o goleiro e a orientação do próprio robô, posicionar o robô neste ângulo e efetivamente chutar. A ação é uma composição lógica de uma função para calcular o ângulo de chute e das primitivas de movimentação, controle do dispositivo de drible (para posicionar a bola, se preciso) e ativação do dispositivo de chute. Caso não haja um angulo que permita o chute ao gol a função avisa seu processo coordenador, na camada de papéis. Este pode então confirmar a intenção de chute ou cancelar a ação.

PassTo( ) - Passa a bola. Esta função avalia qual dos companheiros de equipe está melhor posicionado para receber o passe, e o executa. A avaliação leva em conta múltiplos critérios, como a qualidade da posição do companheiro, ou seja, quão boa aquela posição é para chutar ao gol, e qual a probabilidade de sucesso do passe, considerando distância e chances de que oponentes interceptem a bola. O modelo de avaliação das chances de acerto deve ser empiricamente obtido ou modelado heuristicamente.

Para resolver problemas de múltiplos objetivos, potencialmente conflitantes, diversos métodos da literatura sobre otimização multi-critério (*Figueira et al.*, 2005) podem ser empregados, como, por exemplo, *Kyrylov* (2007), que utiliza a solução pelo Ótimo de Pareto. Nesta implementação, no entanto, opta-se por reescrever a função multi-objetivo como uma função de objetivo único, como mostra a Eq. (4.1). Os efeitos colaterais desta simplificação são que o programador tem a tarefa de ajustar a função, através dos pesos w de cada termo, uma tarefa empírica e que, ao reduzir a dimensão do problema, é possível que algumas das soluções mais eficientes sejam eliminadas. No entanto, a criação de algoritmos de otimização multi-critério estão fora do escopo deste trabalho.

$$Selected(r) = \arg\max_{t \in R} \left[w_0 G(t) + w_1 P(t, O) - w_2 Dist(r, t) \right], t \neq r \tag{4.1}$$

onde:

- r é o robô com a bola e t é um companheiro de equipe de r,
- $Dist(r, t)$ é a distância normalizada do robô r ao seu companheiro t,
- O é o vetor com as posições x, y de todos os robôs em campo,
- $P(t, O)$ é a probabilidade de sucesso do passe, obtida em laboratório, considerando quais dos obstáculos O estão ao longo do trajeto de r até t e a distância do passe,
- $G(t)$ define a aptidão normalizada de t chutar ao gol,
- w_0, w_1 e w_2 são escalares que controlam a influência de cada termo.

Por fim, caso o resultado da maximização desta função esteja abaixo de um nível pré-estabelecido, indicando que nenhum companheiro de equipe é uma boa opção de passe, o processo coordenador, na camada de papéis, pode cancelar a ação.

ReceivePass( ) - Posiciona o robô para receber o passe. Esta ação encontra uma boa posição para receber o passe e posiciona o robô. Para que seja compatível com a função *PassTo()*, a mesma função de estimativa da qualidade de uma posição em campo é utilizada para as duas habilidades. Seguindo o conceito do SPAR (*Veloso et al.*, 1999), a escolha da posição considera as posições dos outros robôs, tanto adversários como companheiros, de forma a contribuir para a um posicionamento coletivo que maximize as opções de passe do robô com a bola. Também como no SPAR, existem restrições impostas para evitar situações específicas, como nunca bloquear a linha de chute de um companheiro. Um exemplo do uso desta habilidade é mostrado na Figura 4.3. Nesta figura, as setas indicam para onde os robôs se movimentarão e qual deles receberá o passe.

DribleToPosition(x, y) - Dribla em direção a uma posição x, y no campo. A função utiliza as primitivas de navegação, para ir até o destino indicado, e de controle do dispositivo de drible, para manter a bola sob controle. Durante o trajeto, a função também mantém a bola fora do alcance de adversários, virando o robô para um angulo em que o adversário fique oposto à bola. Se não for possível encontrar um ângulo em que a bola fique segura, como no caso de múltiplos

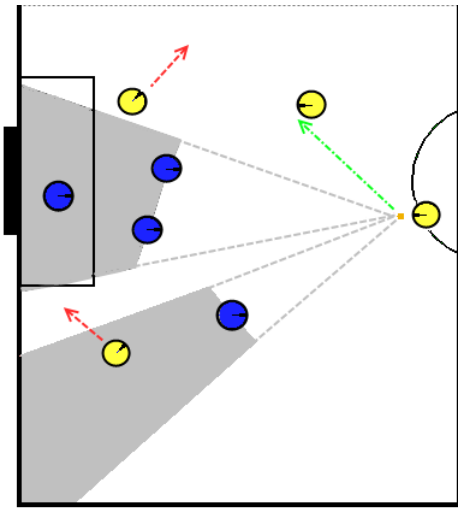


Figura 4.3: Execução da função ReceivePass pelo time atacante. Regiões cinzentas são ruins, de acordo com a função. Setas vermelhas (tracejadas) indicam as posições para onde os robôs desejam mover-se. A seta verde (traço-e-ponto) indica o trajeto planejado para o passe.

adversários, a função avisa seu processo controlador, na camada de papéis. Caso a bola escape do dispositivo de drible, a função também comunica seu controlador.

GoToBall( ) - Intercepta a bola. Esta habilidade utiliza as primitivas de predição da bola e navegação para calcular os pontos em que é possível interceptá-la, em função do tempo t. Se a bola estiver livre, a solução com o menor t é escolhida e executada. Se a bola estiver na posse de um adversário, a ação será a de retomar a bola do adversário.

Block( ) - Posiciona o agente entre o adversário com a bola e o gol, impedindo que este consiga chutar. Se o adversário não estiver com a bola, se posiciona à frente do adversário, impedindo que este receba a bola. O posicionamento exato leva em conta a região do campo e disposição dos adversários e companheiros.

DeltaDefense( ) - Esta habilidade tem o objetivo de formar uma linha defensiva na frente do goleiro, para ajudar a proteger o gol. É chamada de *DeltaDefense* porque forma um triangulo entre o goleiro e os defensores, quando dois ou mais deles executam a função ao mesmo tempo. O defensor se posiciona à frente da linha da área, levemente deslocado para a esquerda ou para a direita em relação à posição do goleiro. O robô executando esta ação também leva em conta, na hora de se posicionar, a posição dos outros defensores, assim evitando que tentem ocupar o mesmo espaço. De acordo com a posição do atacante e da bola em campo, as distâncias do robô à linha da área e aos outros defensores é ajustada, eliminando ângulos que o atacante possa tirar proveito.

4.3.2 Papéis

Ainda em analogia com um jogador humano, uma vez que este sabe como chutar, passar e se posicionar em campo, aprender a realizar as funções de defensor ou atacante é uma questão de assumir objetivos individuais dentro da equipe. Do mesmo modo, os papéis que os robôs poderão assumir podem ser vistos como composições de habilidades e condições lógicas que as ordenam. A seguir, são descritos alguns dos papéis que compõem a implementação proposta, em alto nível de abstração. Estas abstrações serviram de base para a implementação algorítmica de cada um deles.

Centro-Avante - Este papel representa o comportamento do atacante mais ofensivo. Um robô executando este papel tende a se posicionar nas regiões intermediárias do campo ofensivo quando não tem a bola, em posições que favoreçam a recepção de passes, além de não marcar o oponente. Quando em posse dela, sua função de decisão tem grande propensão ao chute a gol e somente passará a bola se, em sua percepção, a métrica de chute a gol do companheiro for muito superior à sua própria e não for possível driblar o oponente.

Atacante - Como o nome diz, representa um atacante, porém menos ofensivo do que o centro-avante. Embora tenha o mesmo *comportamento*, de chutar ao gol, sua função de decisão entre chute, passe ou drible favorece o passe como segunda alternativa e é menos propensa a tentar um chute, mesmo com chances razoáveis de sucesso. Quando a bola estiver com um companheiro, se posiciona para receber passes em qualquer parte do campo, não só no campo ofensivo, como faz o centro-avante e, nos momentos em que a posse de bola for do time adversário, tenta recuperar a bola, caso ela esteja no campo ofensivo ou ele seja o mais próximo.

Meia - Um papel que representa o jogador meio-campista, cuja função principal é entregar a bola aos atacantes. Quando tem a bola, tem a intenção de passar a bola a jogadores mais próximos do gol adversário e que não estejam oclusos por adversários. Se não for vantajoso passar, continua avançando em direção ao gol, até que as chances de sucesso de um chute ao gol sejam altas ou realizar um passe seja proveitoso. Quando um companheiro tem a bola, se posiciona para receber passes na região próxima ao meio do campo, tanto defensivo quanto ofensivo. Quando o adversário tem a bola, tenta retomá-la, se estiver próximo, ou posicionar-se de modo a bloquear passes para outros robôs adversários.

Zagueiro - Sua função é defender a região à frente da área. Executar continuamente a habilidade DeltaDefense e, se for possível recuperar a bola, o fazer e logo em seguida passar ou chutar ao gol. Se nenhuma destas ações for possível, chutar a bola o mais longe possível do

Tabela 4.1: Papéis e habilidades que os compõem

Papel	Habilidades
Atacante	DribleToPosition(x, y) ShootToGoal() PassTo_floor() ReceivePass()
Defensor	DeltaDefense() TackleBall() PassTo_floor()
Meio-Campista	DribleToPosition(x, y) ShootToGoal() TackleBall() PassTo_floor() ReceivePass()

campo de defesa, para a lateral.

4.4 Algoritmos Baseados em Modelos Econômicos

O núcleo da arquitetura de alocação de tarefas proposta é formado por um algoritmo baseado em métodos econômicos. Este algoritmo será o responsável pela distribuição das tarefas através de leilões entre os robôs da equipe, além da coordenação de quando e como novas distribuições, ou alocações, devem ser realizadas. Algoritmos que realizam leilões são conhecidos na literatura, como visto na seção 2.1, por sua eficiência computacional, versatilidade e simplicidade.

Estes algoritmos foram escolhidos por estarem dentre os mais simples modelos econômicos, dessa maneira minimizando a possibilidade de resultados com interferência quando da inserção do algoritmo de Aprendizado por Reforço. O funcionamento básico dos leilões de primeiro preço é o seguinte:

1. **Anúncio das tarefas em leilão.** Um agente, o leiloeiro, inicia um leilão onde os papéis existentes são colocados à venda. Uma mensagem é enviada à todos os agentes informando que há um leilão aberto e quais são os papéis disponíveis.

2. **Avaliação das métricas.** Cada agente avalia suas métricas e formula sua oferta. Estas métricas definem quão bom um agente pensa ser na execução deste papel.

3. **Submissão das ofertas.** Os agentes enviam suas ofertas ao leiloeiro e aguardam o resultado.

4. **Fechamento do leilão e entrega das tarefas.** O leiloeiro define os ganhadores de cada papel e envia uma mensagem aos agentes informando quem são os vencedores.

5. **Retorno ao passo 1.** O processo se repete assim que um intervalo de tempo entre leilões transcorre.

Um parâmetro importante é o intervalo de tempo entre leilões. Nos casos em que o leiloeiro pode monitorar o progresso do robô, esta questão não é tão importante, já que este pode decidir que a tarefa está progredindo e não deve ser posta novamente em leilão. Porém, as tarefas na arquitetura implementada não têm duração e estado terminal definidos, exceto quando o jogo é parado por um gol, bola chutada para fora ou infração. As opções seriam então (i) simplesmente alocar as tarefas apenas nos momentos em que o jogo é paralisado, o que poderia remover parte da capacidade da equipe em reagir a situações novas, ou (ii) expressar o problema como um problema de alocação instantânea iterada, uma solução utilizada comumente (*Stone e Veloso*, 1999; *Werger e Mataric*, 2000; *Weigel et al.*, 2001). A segunda opção foi a escolhida, com a definição do intervalo adequado através dos resultados empíricos.

4.4.1 Métricas

Algoritmos econômicos são tão bons quanto as funções utilidade responsáveis por expressar o interesse ou aptidão dos agentes, portanto é fundamental que estas funções, também chamadas neste trabalho métricas, funcionem corretamente.

Para cada um dos papéis disponíveis, uma métrica unidimensional foi criada, através de somas ponderadas de diversos critérios, com resultados sempre normalizados. Os ajustes de pesos dos critérios foram feitos manualmente.

Deve-se notar que esta redução dimensional para um escalar que agregue os critérios remove as garantias de que os valores atingidos sejam ótimos ou *quasi*-ótimos. Mesmo por vezes sub-ótimo, ajustes manuais e reduções a problemas unidimensionais através do uso de somatórias são técnicas utilizadas para criação de métricas em diversas equipes robóticas, mesmo em arquiteturas reconhecidamente eficientes (*Browning et al.*, 2005). Por ser o foco deste trabalho a inserção de RL em substituição às métricas e não a criação de métricas ótimas, considera-se aceitável esta redução.

Os critérios utilizados são descritos abaixo. Tanto os valores de cada critério quando o resultado da métrica são normalizados entre 0 e 1, exceto quando a descrição explicitar o contrário, e todos os critérios são diretamente proporcionais.

Centro-Avante

Seus critérios são (i) Proximidade da bola, em que quanto mais próximo o robô, maior o valor; (ii) posse da bola. O valor é igual a 1 quando o robô tem a posse de bola, 0 caso contrário; (iii) proximidade do gol adversário. Quanto mais próximo o robô, maior o valor, e (iv) posição da bola no campo ofensivo. Quanto mais próxima do gol adversário, maior o valor. Se a bola estiver no campo defensivo, o valor é 0.

Atacante

O atacante tem em sua maioria os mesmos critérios de seleção do centro-avante, porém seus pesos tem ajustes diferentes, que o tornam menos ofensivo. Seus critérios: proximidade da bola, proximidade do gol adversário e posição da bola no campo ofensivo.

Meia

As métricas de seleção deste papel são (i) a quantidade de companheiros no campo ofensivo sem a bola; (ii) a relação companheiros/oponentes no campo defensivo, cujos valores podem variar entre -1 e 1, sendo positivo quando a relação companheiros/oponentes é igual ou maior do que 1 e negativo caso contrário; e (iii) a existência de companheiro com a posse de bola no campo defensivo, cujo valor é igual a 1 se existir, 0 caso contrário.

Zagueiro

As métricas deste papel são (i) a posse de bola pelo time adversário, com valor igual a 1 se existir, 0 caso contrário; (ii) quantidade de companheiros mais distantes do gol a ser defendido do que o robô; (iii) a quantidade de oponentes no campo ofensivo; (iv) proximidade do gol a ser defendido.

4.5 Algoritmo de Aprendizado por Reforço

A literatura sobre Aprendizado por Reforço contempla diversos algoritmos adequados para uso em ambientes como o futebol de robôs: episódicos, discretizáveis, modeláveis por MDPs, entre outros. Neste trabalho, utilizou-se o algoritmo Q(λ) (seção 3.7), pelos motivos a seguir.

O Q(λ) é uma composição de dois métodos de RL com fundações teóricas reconhecidas e diversos exemplos de sucesso empírico, o Q-Learning e o método TD(λ), e portanto herda características desejáveis de ambos. Do Q-Learning, a facilidade de implementação. Do TD(λ), a capacidade de propagação temporal de reforços, que ajuda a amenizar os efeitos negativos gerados pelo atraso nas recompensas distribuindo partes do reforço a estados intermediários que contribuíram com a política alcançada.

Outra razão pela escolha é que em uma aplicação robótica, onde cada visita a um estado é custosa, generalização é quase um pré-requisito, e o Q-Learning e suas variações apresentam resultados práticos bastante positivos quando aliados a aproximadores de função como CMACs e redes neurais.

Na arquitetura implementada, CMACs foram utilizados para representar a função Q, implementados de forma similar ao proposto por *Watkins* (1989), com codificação em telhas e *hashing*.

A comparação dos resultados do algoritmo acima descrito com o algoritmo descrito na seção 3.8, o HAQ(λ) (*Martins e Bianchi*, 2007), que consiste do algoritmo Q(λ) aliado à aceleração por heurísticas proposta por *Bianchi* (2004), também foi realizada, e os resultados encontram-se no capítulo 5. Nesta comparação, buscou-se comprovar empiricamente que, em um ambiente complexo como o do futebol de robôs, com elevado número de estados e ações, o uso de heurísticas possibilitaria a redução significativa do tempo de convergência.

4.5.1 Vetor de Estados

Um tópico importante no projeto do algoritmo de RL é a definição do vetor que representa os estados, principalmente em um ambiente complexo e com número considerável de robôs atuando, como o utilizado. Uma representação comumente utilizada é criar um espaço de estados formado pelas posições em campo (x, y ou x, y, θ) de todos os robôs e da bola. Pode-se argumentar, no entanto, que esta é uma representação apropriada apenas para ações reativas imediatas. Para utilização em níveis de abstração cujas decisões tem maior duração temporal, como é o caso dos papéis, mostrou-se insuficiente. Observar a disposição dos robôs em um instante em que todos os adversários estão no campo defensivo não é suficiente para definir se o adversário tem comportamento defensivo ou ofensivo, se é ágil ou não. Contra um time defensivo, um número maior de atacantes pode ser mais apropriado, enquanto contra um time ofensivo e ágil esta disposição pode ser inefetiva.

Para que a representação do espaço de estados neste nível de abstração seja apropriada, então, deve-se utilizar representações que capturem aspectos de duração temporal mais próximos ao intervalo do algoritmo de RL ou representações que capturem a dinâmica do ambiente. Exemplos destes aspectos são condições dinâmicas, como velocidades dos robôs, ou ainda distribuições estatísticas a respeito de passes ou retomadas de bola. Este mesmo conceito, de representar atributos que tenham significância temporal mais ampla, aparece em outros estu-

dos que utilizam níveis de abstração como os propostos neste trabalho (*Vail e Veloso*, 2008; *Sukthankar e Sycara*, 2006).

O algoritmo de RL utilizado possui variáveis criadas a partir de um algoritmo que mantém 10 posições x, y anteriores dos robôs, amostradas uma vez por segundo, em um histograma de posições do campo. A cada ciclo de decisão do algoritmo de RL, extraem-se parâmetros que definem características do movimento destes robôs a partir destes dados, como maior distância percorrida sobre cada eixo do plano e as três posições x, y em que o histograma tem valores mais altos (indicando as posições onde o robô esteve posicionado por mais tempo). O uso destas características como variáveis do espaço de estados permite diferenciar robôs estáticos de robôs que percorrem porções do campo, bem como definir suas áreas de atuação, permitindo assim estabelecer seus padrões de movimentação. Outras variáveis do espaço de estado são a posição da bola em campo e a quantidade de chutes a gol de cada time durante o último ciclo de decisão.

O espaço de estados resultante, com 27 dimensões, passa então por um aproximador de função do tipo CMAC (seção 3.9.1), o que permite uma representação compacta o suficiente para ser armazenada em memória. Mesmo com o uso do CMAC e a alocação dinâmica de memória (um estado só é armazenado quando visitado), cada agente requer, em média, 2,8GB de memória para o armazenamento do vetor de estados. Consequentemente, são necessários 11,2GB de memória para a equipe com quatro jogadores e um goleiro.

4.5.2 Reforços e outros parâmetros

Na implementação descrita neste trabalho, segue-se o conceito de reforços atrasados, pois os reforços mais significativos são recebidos apenas ao fim de cada episódio, definido como a ocorrência de um gol ou o final de tempo de jogo. Nestes estados terminais, todos os agentes da equipe recebem reforços iguais. São reforços positivos quando o time marca um gol a favor, e negativos quando o time sofre um gol ou faz um gol contra. Existe também um reforço negativo, de significância bem menor e atribuído imediatamente, a cada passo que não atinja um estado terminal (gol). A existência deste reforço serve para evitar que os agentes aprendam a não fazer nada ou entrem em laços, bem como é um pré-requisito para que o método de heurísticas tenha convergência.

Os valores para as recompensas foram ajustados em testes iniciais no simulador, bem como os valores da função heurística. Os valores utilizados foram +100 para o reforço positivo por

alcançar o estado terminal favorável, -100 para o reforço negativo do estado terminal desfavorável e -1 para cada transição de estado que não atingiu nenhum estado terminal. Para a função heurística, o valor tem uma ordem de magnitude menor, de acordo com a Eq. (3.19) e o valor adotado como horizonte de desconto, γ.

Os outros parâmetros do algoritmo Q(λ) foram os seguintes: método de exploração ϵ-*greedy*, com $\epsilon = 0,1$ fixo e $\gamma = 0,9$ (estes são valores típicos), $\lambda = 0,3$, e rastro de elegibilidade por substituição. Os resultados experimentais do capítulo 5 apresentam novamente os valores destes parâmetros, para maior facilidade de entendimento.

O próximo capítulo detalha os experimentos e resultados da implementação desta arquitetura de alocação de tarefas, bem como os ambientes simulado e real, em que foi aplicada.

Capítulo 5

EXPERIMENTOS E RESULTADOS

Este capítulo descreve os experimentos executados para validar a arquitetura de alocação de tarefas implementada neste trabalho. O capítulo começa por uma breve descrição dos ambientes de implementação utilizados. Os experimentos em simulação são então descritos e tem seus resultados apresentados, seguidos pelos experimentos em ambiente real. O capítulo termina com uma discussão dos resultados.

5.1 Ambientes de implementação

Existem dois ambientes de implementação neste trabalho, um com robôs reais e um simulado. O ambiente simulado, de importância fundamental, permite testar rapidamente aspectos dos algoritmos, sem as dificuldades inerentes a um ambiente real, como limitações na autonomia das baterias, necessidade de supervisão durante testes, uso do campo do laboratório e, claro, a impossibilidade de acelerar o tempo.

O ambiente simulado implementado é baseado no simulador Teambots (*Balch*, 2000), mostrado na Figura 5.1. Ele foi desenvolvido, entre outros propósitos, para a liga Small Size da RoboCup, a mesma categoria dos robôs reais utilizados. O Teambots modela as características cinemáticas do ambiente, como velocidade e aceleração dos robôs e bola, bem como colisões simples.

Para uso neste trabalho, o Teambots foi modificado e funcionalidades foram adicionadas. Foi adequado às regras atuais, incluindo a inserção de programação do árbitro eletrônico. A cinemática omnidirecional e os dispositivos de chute e drible dos robôs reais foram modelados, e ganhou a capacidade de conexão via UDP/IP com o software que controla o time, para permitir que a arquitetura desenvolvida fosse utilizada tanto em simulação quanto no ambiente real,

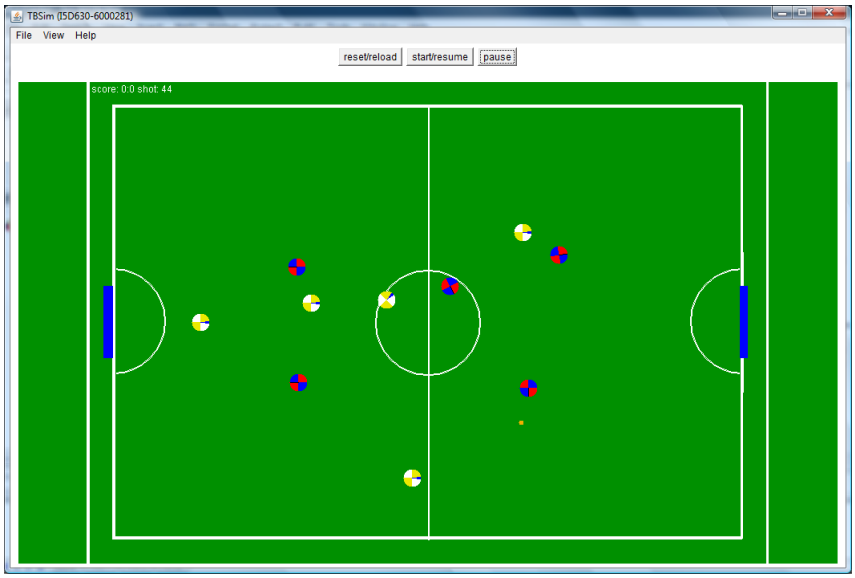


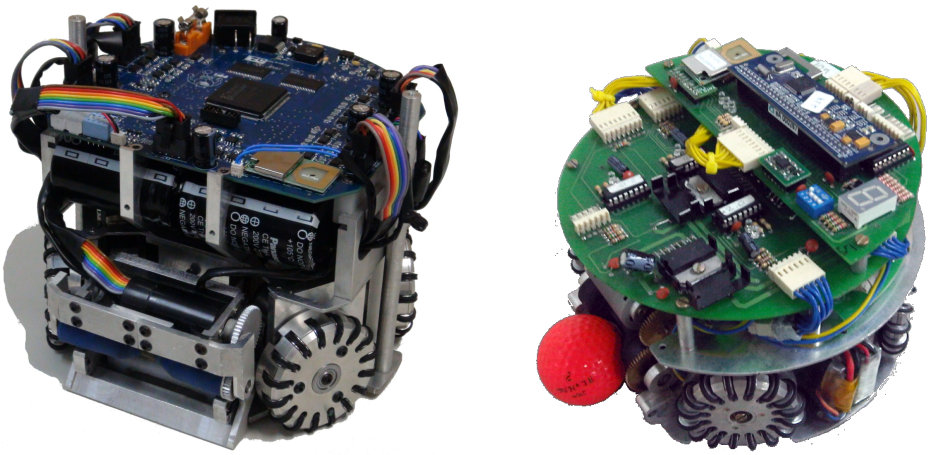
Figura 5.1: O Simulador Teambots

sem necessidade de alterações ou ajustes. Por fim, foram implementadas funções para que a simulação pudesse ser executada continuamente e em modo acelerado, o que possibilitou atingir velocidades de simulação mais de 200 vezes superiores ao modo de tempo real.

No ambiente real, os robôs utilizados foram os F180 gerações 2009 e 2010, desenvolvidos pelo projeto RoboFEI. Estes robôs foram projetados para jogar nas competições da liga Small Size, a mais dinâmica das categorias de futebol de robôs mantidas pela federação RoboCup, onde robôs de 180 mm de tamanho e 150 mm de altura, com velocidades que superam 5 m/s, jogam em um campo de aproximadamente 6 x 4 m de área, dispostos em equipes de 5 robôs.

Estes robôs, mostrados na Figura 5.2, tem formato cilíndrico e quatro rodas omnidirecionais, dispostas de forma que o robô pode se movimentar em qualquer direção no plano, sem a necessidade de girar em torno do próprio eixo. Os robôs possuem um dispositivo de chute de força variável, o que permite os passar a bola para companheiros posicionados em qualquer ponto do campo, bem como chutar a bola ao gol. Possuem também um dispositivo de drible, capaz de manter a posse da bola mesmo quando o robô necessita fazer curvas ou andar para trás, além de serem capazes de detectar, através de um sensor infravermelho, se a bola está efetivamente em posição para chutes. Os comandos que controlam os robôs são enviados por um ou mais microcomputadores, através de um rádio bidirecional de 2,4 GHz.

A localização dos robôs e da bola é feita através de um sistema de visão global, formado por câmeras montadas sobre o campo e que transmitem imagens a uma taxa de 60 quadros por segundo (aproximadamente 16 ms entre quadros). Uma imagem captada pelas câmeras sobre o campo pode ser vista na Figura 5.3.



(a) (b)

Figura 5.2: Robô RoboFEI Small Size, versão 2009. (a) Vista da montagem mecânica e (b) Foto do robô completo.

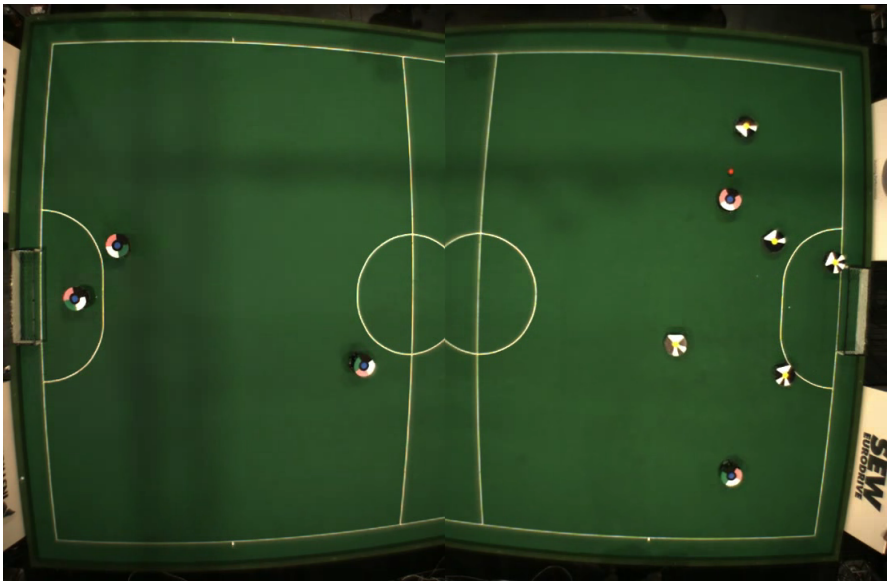


Figura 5.3: Imagem concatenada de duas câmeras, posicionadas acima do campo, em montagem tipicamente utilizada na categoria Small-Size. (fonte: equipe B-Smart)

5.2 Experimentos em Ambiente Simulado

Para a validação da arquitetura proposta, um teste de validação e quatro conjuntos de experimentos em ambiente simulado, com diferentes versões do mecanismo de alocação de tarefas, foram realizados, e serão descritos em detalhes nesta seção.

No primeiro experimento, a alocação de tarefas foi feita através de um método econômico, o algoritmo de leilões de primeiro preço, sem a utilização de Aprendizado por Reforço. No segundo experimento o inverso foi testado, com a alocação de tarefas realizada apenas por um algoritmo de RL, o Q(λ), sem nenhum método econômico. No terceiro experimento, os mesmos algoritmos, econômico e de RL, foram utilizados em conjunto, enquanto no quarto experimento foram utilizados em conjunto os algoritmos econômico e de RL acelerado por heurísticas. Estes

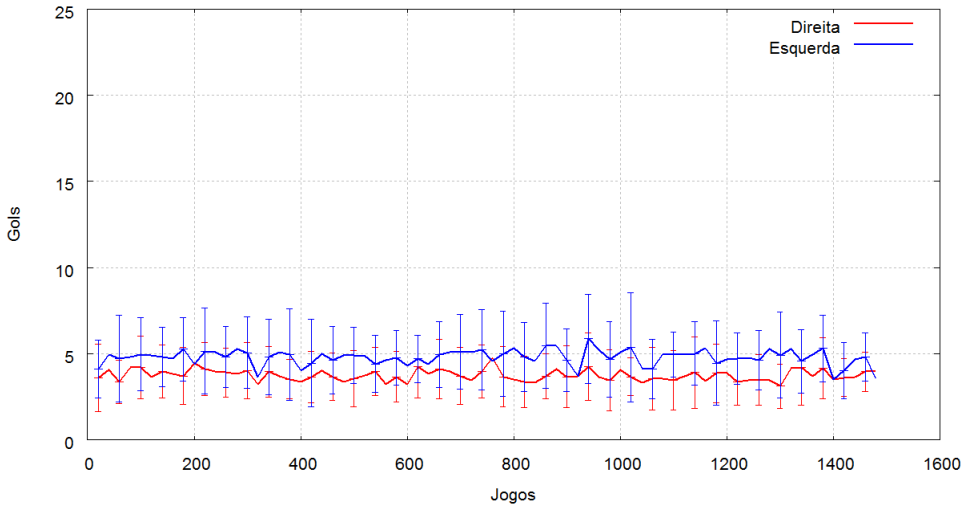


Figura 5.4: Influência do lado para o qual o time ataca, representado pelas médias de 20 jogos e desvio padrão do mesmo intervalo. A Linha vermelha representa o time que ataca para a direita, enquanto a linha azul o time que ataca para a esquerda.

algoritmos são os algoritmos detalhados nas Seções 4.4 e 4.5, respectivamente.

Os experimentos simulados foram executados em uma plataforma computacional com 2 processadores Intel Xeon Quad Core E5520 operando a 2,26 GHz, 32GB de memória RAM, executando a plataforma de virtualização VMWare ESXi 3.6. Nesta plataforma, duas máquinas virtuais com o sistema operacional Microsoft Windows Server 2008 64 bits operaram em paralelo, cada uma executando simultaneamente o simulador e os softwares do time base, usado como adversário, e da implementação apresentada neste trabalho. Durante as simulações, o simulador foi configurado para operar em modo de tempo acelerado, e a taxa de aceleração aproximada obtida foi de 30 vezes. Nesta taxa de aceleração, cada segundo em tempo real corresponde a 30 segundos de tempo de simulação e, portanto, cada treinamento de 20 mil jogos teve duração aproximada de 5 dias em tempo real.

A validade estatística dos resultados é verificada através de realizações do teste T de Student, encontrado em muitos livros de estatística, como *Spiegel* (1993). Todos os testes foram amostrados em 5 treinamentos independentes.

5.2.1 Teste de Validação

O teste de validação consistiu em verificações diversas do ambiente, como a confirmação do recebimento simultâneo dos pacotes de simulação pelos dois times, e um teste com o objetivo de verificar se existia influência do lado do campo em que cada equipe estava.

Por serem baseados no mesmo código de programação, uma vez desativada a arquitetura de alocação de tarefas, espera-se que não haja diferença entre o desempenho dos times em função

do lado para o qual atacam. O resultado do teste de influência do lado é apresentado no gráfico da Figura 5.4, que mostra a médio de gols marcados por cada time a cada 20 jogos. O teste foi feito com 5 séries de 1500 jogos.

Observa-se pelos resultados que existe uma situação levemente mais favorável ao time que ataca para a esquerda. Baseado neste resultado, decidiu-se por colocar o time aprendiz sempre atacando para a direita, na situação mais desfavorável.

5.2.2 Descrição e Resultados dos Experimentos em Ambiente Simulado

Os quatro experimentos conduzidos, e que serão agora descritos, são os seguintes:

- Experimento 1 - Leilões
- Experimento 2 - Q(λ)
- Experimento 3 - Leilões + Q(λ),
- Experimento 4 - Leilões + HAQ(λ)

Ao longo destes experimentos, as seguintes características foram mantidas constantes: (i) As tarefas, ou papéis, disponíveis para alocação, foram mantidas inalteradas. (ii) A estratégia empregada tanto pelo time que realiza MRTA quanto pelo oponente, em todos os experimentos, é o mesmo utilizado pela equipe RoboFEI durante o campeonato mundial RoboCup de 2010, chamado portanto RoboFEI'10. A única diferença entre os dois times então é a adição da arquitetura de alocação de tarefas.

Experimento 1 - Neste experimento, os leilões (Seção 4.4) foram utilizados sem Aprendizado por Reforço. Os objetivos do experimento foram ajustar os parâmetros do algoritmo, como os valores das funções utilidade e o intervalo entre leilões.

Em um primeiro teste, foram ajustados empiricamente os valores escalares retornados pelas métricas (descritas na Seção 4.4.1) que formam as funções utilidade de cada robô, para cada papel. Neste e em todos os leilões dos outros experimentos, com uma exceção em que há menção explícita, o leiloeiro foi programado para leiloar primeiro os papéis (tarefas) de ataque quando a equipe estava em situações ofensivas e leiloar primeiro os papéis de defesa, caso contrário. Os valores escalares das métricas que produziram o melhor desempenho global do time neste teste foram então fixados e utilizados em todos os experimentos conduzidos.

Tabela 5.1: Resultado do teste de intervalos entre leilões (em segundos), mostrando a média de gols por jogo e o respectivo desvio padrão para cada caso.

Intervalo (s)	Média de gols por jogo
5	$2,06 \pm 1,43$
15	$8,62 \pm 3,47$
30	$6,31 \pm 2,92$
45	$3,35 \pm 1,76$

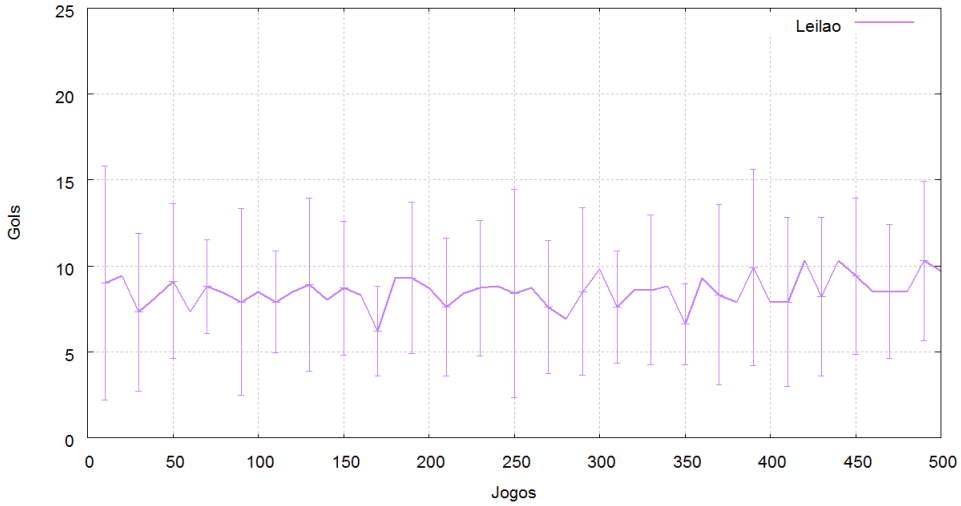


Figura 5.5: Experimento 1 - Leilões. Cada ponto no gráfico representa a média e o desvio padrão do número de gols marcados por partida, ao longo de 10 jogos, em 5 execuções independentes. Cada execução teve 500 jogos.

Uma vez definidos os valores das métricas ou funções utilidade, diferentes intervalos entre leilões foram testados, a fim de verificar sua influência no mecanismo de alocação de tarefas. O resultado do teste de diferentes intervalos aparece na Tabela 5.1, onde nota-se que o melhor intervalo foi o de 15 segundos, com média de 8,62 gols por partida. Este intervalo é apresentado também de forma mais detalhada, no gráfico da Figura 5.5.

Experimento 2 - Consistiu em utilizar o algoritmo Q(λ) no agente leiloeiro, permitindo que este selecionasse centralizadamente os papéis de todos os robôs. O algoritmo Q(λ) foi configurado com o vetor de estados descrito nas Subseção 4.5.1 e parâmetros $\epsilon = 0,1$, $\gamma = 0,9$ e $\lambda = 0.3$.

O algoritmo foi também ajustado para executar uma iteração a cada 15 segundos ou quando um estado terminal (gol, a favor ou contra) fosse alcançado. Este intervalo entre iterações foi escolhido por ser o mesmo intervalo com o melhor resultado no experimento do algoritmo de Leilões (experimento 1). Uma investigação sobre o intervalo ótimo para o algoritmo de RL não foi conduzida, pois o tempo de testes seria proibitivo para este trabalho, uma vez que envolveria executar tentativas de 20 mil jogos adicionais.

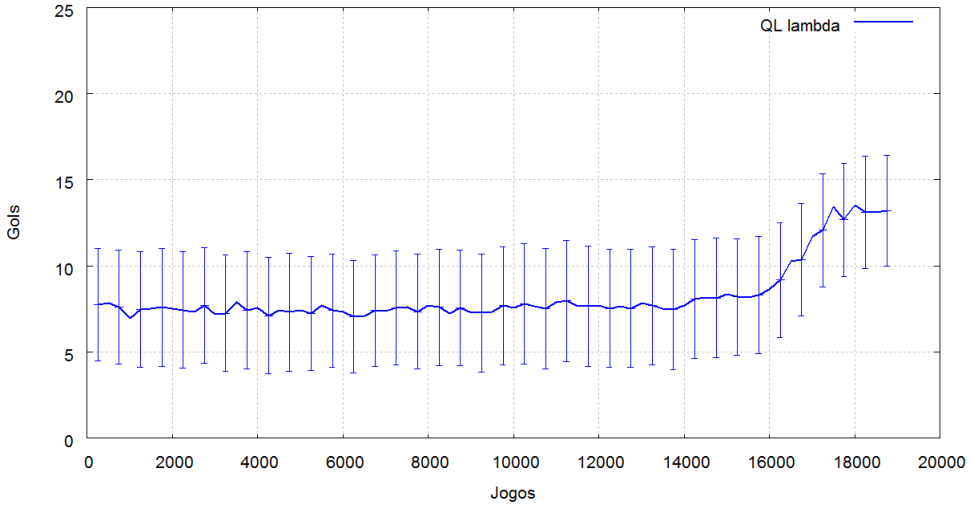


Figura 5.6: Experimento 2 - Q(λ). O gráfico representa as médias e desvios padrão do número de gols marcados por partida, em grupos de 10 jogos, em 5 execuções independentes. Cada execução teve cerca de 20 mil jogos.

Os resultados do experimento, apresentados no gráfico da Figura 5.6, mostram a média e desvio padrão do número de gols marcados por partida, em blocos de 10 jogos. O experimento foi repetido 5 vezes, em tentativas independentes, assim como os outros experimentos simulados. Estes resultados mostram que o algoritmo Q(λ) foi efetivo em aprender a distribuir os papéis dos robôs em campo, obtendo resultados melhores do que o uso de leilões apenas. Porém, como já era esperado, o número de jogos até que o aprendizado produzisse feitos foi bastante elevado.

Experimento 3 - Também utilizou um sistema de leilões, igual ao do experimento 1, porém as funções utilidade de cada robô foram aprendidas por algoritmos Q(λ) independentes, instanciados dentro de cada um deles. O espaço de estados e parâmetros utilizados foram os mesmos do experimento 2. Os resultados do experimento aparecem na Figura 5.7.

Experimento 4 - É uma repetição do anterior, porém utilizando o algoritmo HAQ(λ), uma variação do Q(λ) acelerado por heurísticas, visto na Seção 3.8. O sistema de heurísticas empregado utilizou as mesmas métricas criadas para o experimento 1, em que a alocação era feita através de leilões. Para definir o valor da função $H(p, s_t)$, utilizado na equação do HAQ(λ) (Eq. (3.16)), a função da Eq. (5.1), foi utilizada. Esta função tinha o funcionamento bastante simples: as métricas de todos os papéis p' eram executadas e, para a métrica de maior valor, o resultado de $H(p)$ retornado era 1. Para todas as outras, o valor era 0.

$$H(p, s_t) = \begin{cases} 1 & \text{caso } arg\max_{p'}(metrica(p', s_t)) \\ 0 & \text{caso contrário} \end{cases} \tag{5.1}$$

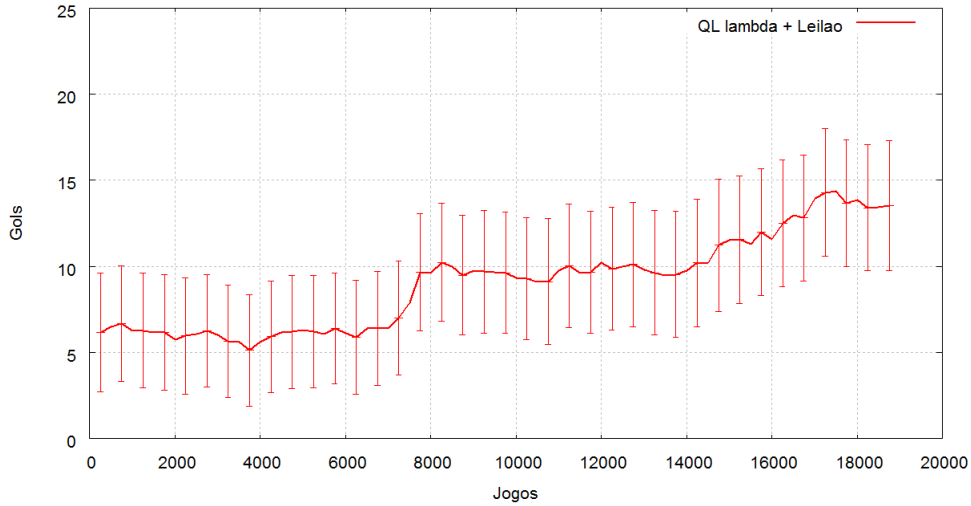


Figura 5.7: Experimento 3 - Leilões + Q(λ). O gráfico representa as médias e desvios padrão do número de gols marcados por partida, em grupos de 10 jogos, em 5 execuções independentes. Cada execução teve cerca de 20 mil jogos.

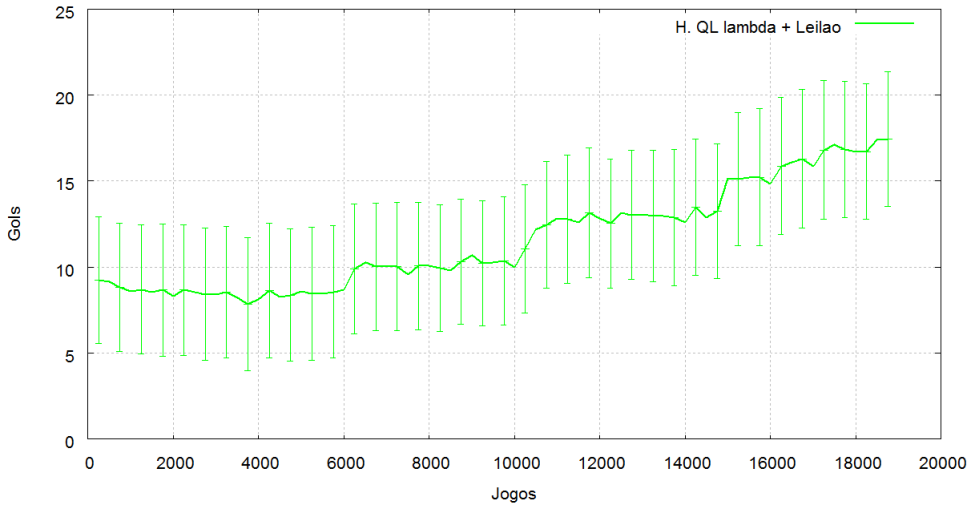


Figura 5.8: Experimento 4 - Leilões + HAQ(λ). O gráfico representa as médias e desvios padrão do número de gols marcados por partida, em grupos de 10 jogos, em 5 execuções independentes. Cada execução teve cerca de 20 mil jogos.

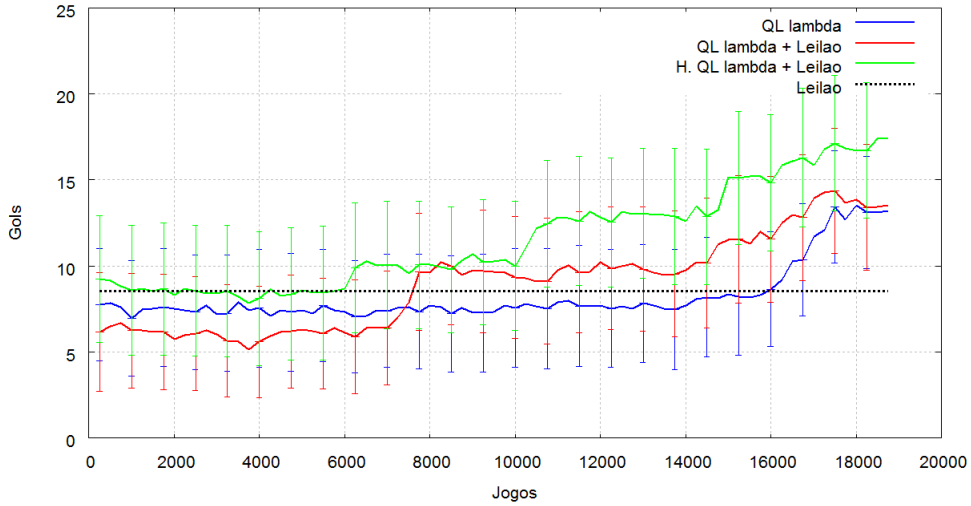


Figura 5.9: Comparativo das diferentes variações do algoritmo de alocação de tarefas. O gráfico representa as médias e desvios padrão do número de gols marcados por partida, em grupos de 10 jogos, em 5 execuções independentes. Cada execução teve cerca de 20 mil jogos, exceto o experimento apenas com Leilões, cujos resultados foram extrapolados a partir dos 500 jogos executados.

Apresentados os resultados de todos os experimentos, é conveniente agrupá-los em um gráfico que permita compará-los, e por este motivo foi criado o gráfico da Figura 5.9. Este gráfico é de relevância para a análise dos testes T de Student, mostrados na seção a seguir, bem como na discussão de resultados ao final do capítulo.

5.2.3 Testes T de Student

O teste T de Student (*Spiegel*, 1993)consiste em calcular o valor de T através da Eq. (5.2) e então comparar o módulo do resultado com o valor crítico de T, um valor tabelado de acordo com o número de graus de liberdade (amostras) e nível de confiança desejados. Se o valor crítico for maior do que o valor do módulo de T, então os experimentos diferem significativamente.

$$T = \frac{\mu_x - \mu_y}{\sqrt{(n_x - 1)\,\sigma_x^2 + (n_y - 1)\,\sigma_y^2}} \sqrt{\frac{n_x n_y\,(n_x + n_y - 2)}{n_x + n_y}} \tag{5.2}$$

Onde n_x e n_y são os números de amostras, μ_x e μ_y as médias e σ_x e σ_y os desvio padrão dos experimentos x e y.

Testes T foram realizados nos resultados experimentais deste trabalho com o objetivo de confirmar ou rejeitar a hipótese de que os algoritmos produzem médias de gols iguais após dados números de partidas. Todas as seis combinações de pares possíveis entre os quatro experimentos foram calculadas, e os resultados aparecem em seis gráficos, nas Figuras 5.10 a 5.15.

O número de graus de liberdade α, definido como $\alpha = n_x + n_y - 2$, foi o mesmo para todos os experimentos simulados. O cálculo de α foi feito da seguinte maneira: cada dado representa

a média de gols de 10 jogos ao longo de 5 tentativas independentes, logo $n = 10 \times 5 = 50 \Rightarrow n_x + n_y - 2 = 50 + 50 - 2 = 98$. Para este número de graus de liberdade, e nível de confiança 95%, o valor crítico $T\,(\alpha = 98, p = 95\%) = 1,9842$. Este valor crítico aparece em todos os gráficos, com a legenda *limite de* 5%.

Analisando cada um dos seis gráficos dos testes T (Figuras 5.10 a 5.15), em conjunto sempre com o gráfico comparativo da Figura 5.9, observa-se que os resultados são coerentes. Nos pontos em que os gráficos dos experimentos se cruzam ou são verdadeiramente próximos os resultados dos testes T indicam similaridade, e nos outros pontos indicam diferenças significativas. Alguns exemplos de estados de similaridade são os gráficos de *Q(λ)* e *Leilões + Q(λ)* em cerca de 7000 jogos, e os valores dos gráficos de *Leilões* e *Leilões + HAQ(λ)*, do início até 6000 jogos.

5.3 Experimentos em Ambiente Real

Experimentos em robôs reais foram conduzidos em duas situações: em laboratório, onde dados suficientes para análises estatísticas foram obtidos, e em alguns testes empíricos durante a Competição Latino-Americana de Robótica 2010 (LARC 2010). Os próximos parágrafos descrevem ambas situações e os experimentos realizados.

Em laboratório, experimentos foram conduzidos entre duas equipes compostas de dois robôs F180 versão 2010, dois robôs F180 versão 2009 e um goleiro inativo (robô desligado), cada. Esta configuração foi necessária pois existiam, no momento dos testes, apenas cinco robôs versão 2010 e, dentre os robôs versão 2009, apenas três apresentavam mínimas condições de operação. Não era possível reparar mais robôs versão 2009, pela ausência de partes sobressalentes, como engrenagens e motores.

O objetivo dos experimentos realizados foi transferir o conhecimento aprendido durante os jogos simulados para os robôs reais, em diferentes estágios do aprendizado, e verificar o desempenho dos robôs reais. A transferência foi realizada através da copia do conteúdo das tabelas Q dos agentes simulados para os agentes reais.

Foram testadas duas variações de algoritmo da arquitetura de MRTA: o Q(λ) e o HAQ(λ) + Leilões. Para cada uma destas variações foram realizados quatro testes de transferência, em diferentes estágios do aprendizado em simulação: 0 (nenhum), 5.000, 10.000 e 15.000 jogos experienciados. Cada teste teve 3 treinamentos de 50 jogos cada, e todos os parâmetros dos experimentos utilizados em ambiente simulado foram mantidos. O time oponente, como nos

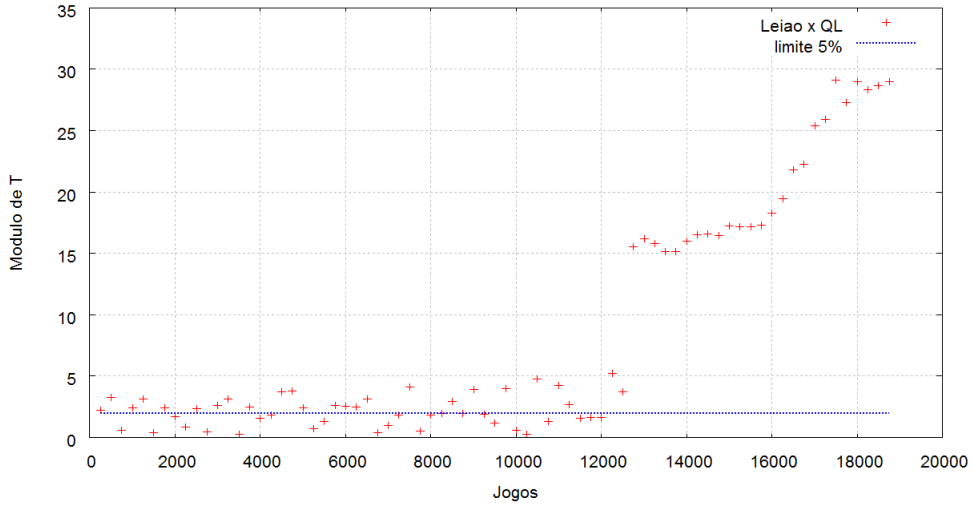


Figura 5.10: Teste T comparando a implementação utilizando Leilões e a implementação utilizando Q(λ).

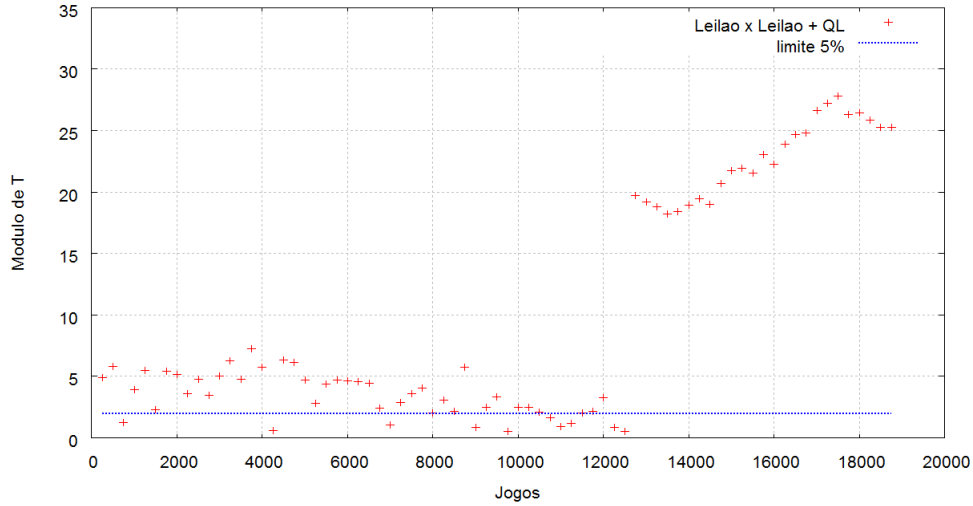


Figura 5.11: Teste T comparando a implementação utilizando Leilões e a implementação utilizando Leilões + Q(λ).

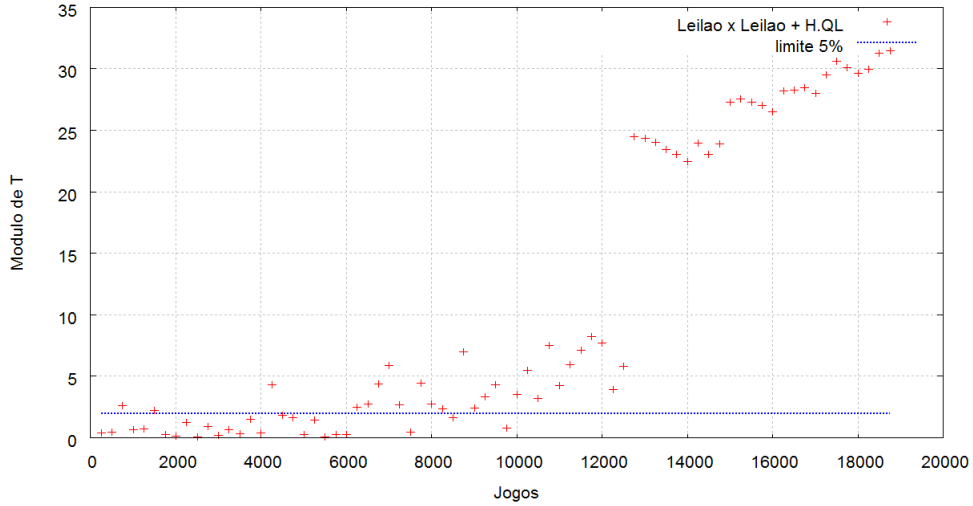


Figura 5.12: Teste T comparando a implementação utilizando Leilões e a implementação utilizando Leilões + HAQ(λ).

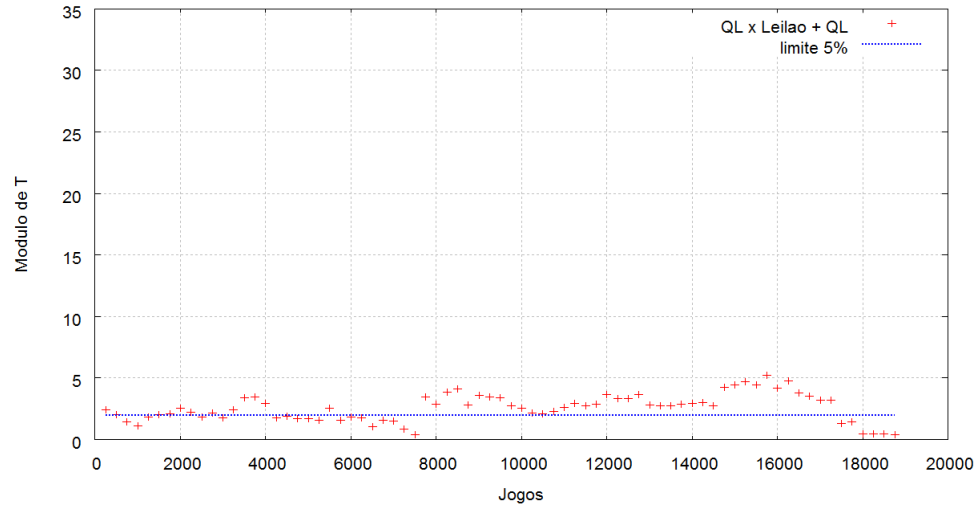


Figura 5.13: Teste T comparando a implementação utilizando Q(λ) e a implementação utilizando Leilões + Q(λ).

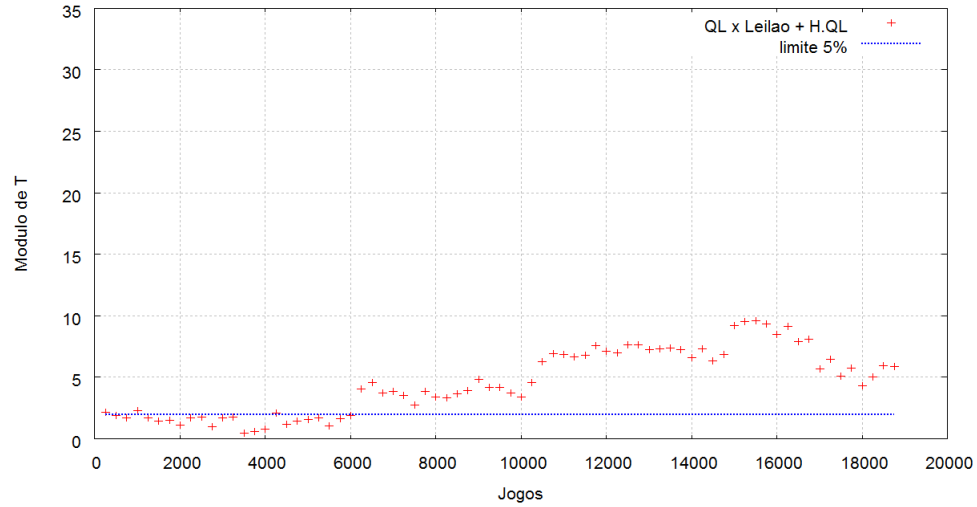


Figura 5.14: Teste T comparando a implementação utilizando Q(λ) e a implementação utilizando Leilões + HAQ(λ).

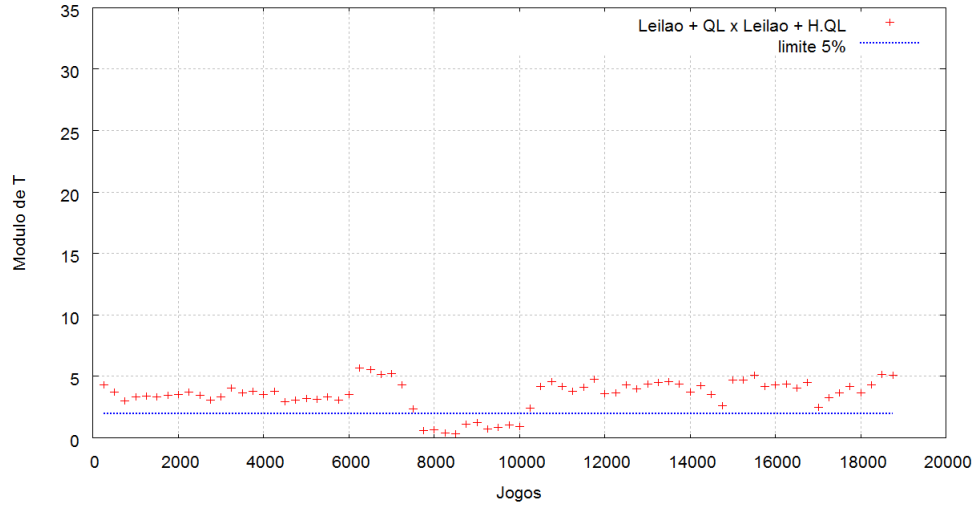


Figura 5.15: Teste T comparando a implementação utilizando Leilões + Q(λ) e a implementação utilizando Leilões + HAQ(λ).

Tabela 5.2: Resultado da transferência do aprendizado simulado para robôs reais, mostrando a média de gols por jogo e o respectivo desvio padrão. O resultado do time base, sem nenhum algoritmo de MRTA, é apresentado para fins comparativos.

Algoritmo	Jogos transferidos	Média de gols robôs reais
Time Base	-	$3,12 \pm 1,52$
Q(λ)	0	$2,12 \pm 2,54$
	5000	$5,18 \pm 2,97$
	10000	$5,43 \pm 2,83$
	15000	$6,55 \pm 3,10$
HAQ(λ) + Leilões	0	$3,01 \pm 2,34$
	5000	$5,20 \pm 3,11$
	10000	$7,69 \pm 3,65$
	15000	$7,44 \pm 3,04$

experimentos simulados, foi mantido com o software RoboFEI'10, já detalhado anteriormente neste capítulo. Os resultados são mostrados na Tabela 5.2, que apresenta também o desempenho do time sem a arquitetura de MRTA. Estes resultados tem tendências de aumento de performance similares aos resultados dos experimentos simulados, portanto conclui-se que a transferência entre os dois ambientes foi bem sucedida.

Os testes realizados durante a Competição Latino-Americana de Robótica 2010 consistiram em, de maneira similar aos testes em laboratório, transferir o aprendizado total, de 20 mil partidas em simulação, para a equipe real. O algoritmo utilizado foi o HAQ(λ) + Leilões. Estes testes tiveram duração de alguns minutos durante as partidas de oitavas-de-final e semi-final. Os testes foram executados durante porções de tempo destas partidas, e não em suas durações totais, porque as partidas eram eliminatórias, e uma eventual derrota desclassificaria a equipe da competição.

Durante o teste da partida de oitavas-de-final, a equipe adversária contava com apenas 3 robôs em campo, o que levou o algoritmo de RL a visitar estados que nunca haviam sido antes visitados em simulação, inutilizando quase na totalidade o aprendizado existente nas tabelas Q. O desempenho da equipe foi bastante afetado negativamente, e o teste interrompido no segundo minuto de execução. Embora com resultados negativos, foi um teste produtivo, pois apontou defeitos diante da situação em que o oponente tem menor número de jogadores.

Na partida semi-final, a equipe adversária contava com todos os cinco robôs em campo, portanto o teste pôde ser executado em um ambiente similar ao simulado. Novamente, o teste consistiu em transferir o aprendizado de 20 mil partidas simuladas e utilizar o algoritmo HAQ(λ) + Leilões. O teste foi realizado a partir do início do segundo tempo da partida, quando o placar

estava em 4 x 0 para a equipe RoboFEI. Durante os 4 primeiros minutos de jogo, mais três gols foram marcados, com a utilização do sistema de MRTA, e então a equipe adversária desistiu da partida. Embora não seja possível inferir conclusões com base em execuções tão breves, algum dos registros gravados pelo módulo de gravação de informações (*datalogger*) contém sequencias interessantes, como o que aparece na Figura 5.16. Na Figura 5.16 (a) aparecem áreas contornadas por linhas brancas ao redor dos jogadores e bola. Estas áreas representam as porções do campo que estes objetos ocuparam desde o último leilão (um período de 15 segundos, neste experimento), e são utilizadas no vetor de estados do algoritmo de RL e pelas métricas do leiloeiro. A sequência dos acontecimentos na Figura 5.16 é a seguinte: em 5.16(a) acontece um leilão. Neste momento, existem dois jogadores defensores e dois atacantes no time. Neste leilão, o módulo *Coach* coloca três atacantes e dois defensores em leilão. O resultado do leilão transforma a disposição do time, já que um dos defensores vence o leilão por um dos papéis de atacante. A mudança para três atacantes aconteceu principalmente porque os jogadores adversários no campo de ataque quase não se movimentaram deste o leilão anterior, o que indica uma menor necessidade de 2 defensores. O algoritmo de RL também continha experiência anterior em estados semelhantes e decidiu que era atrativo tentar um papel de atacante. A Figura 5.16 (b) mostra um segundo depois do leilão, quando o jogador que tem a posse da bola chuta ao gol. Em 5.16 (c), uma fração de segundo depois pode-se observar já o efeito da mudança de papéis. o jogador que estava na defesa agora ruma para o campo de ataque, para ajudar os companheiros. Enquanto isso, o goleiro adversário defende, e a bola é rebatida de volta. Em 5.16 (d) o jogador que vem da defesa já está no meio do campo, e se torna o mais apto a ir em direção à bola. Na Figura 5.16 (e) este jogador se aproxima da bola e, em 5.16 (f) a bola está em sua posse. O jogador chuta novamente ao gol. O resultado desta jogada foi um chute para fora do gol. Embora um gol não tenha sido marcado, esta sequência evidencia a efetividade do sistema de MRTA aplicado.

5.4 Discussão

Os resultados dos experimentos simulados indicam que o sistema de alocação de tarefas foi capaz de aumentar o desempenho da equipe em todas as suas variações de implementação, e a união dos métodos econômico e de RL resultou em desempenhos superiores aos de seus usos separadamente, conclusões visíveis no gráfico da Figura 5.9.

Ainda utilizando a Figura 5.9, outras observações podem ser extraídas. A primeira delas

é de que o uso apenas do algoritmo Q(λ) leva a um tempo de aprendizado significativamente maior do que as demais variações do algoritmo experimentadas, fato que era esperado.

Observa-se também que o uso do algoritmo Leilões + Q(λ) apresenta uma média inicial aproximada de 6,2, inferior tanto às médias do experimento apenas com leilões quanto do experimento apenas com o Q(λ), que são de 8,62 e 7,5, respectivamente. Esta dificuldade acontece porque, utilizando RL, inicialmente os robôs não tem conhecimento armazenado em sua tabela Q, e portanto não tem como avaliar para quais papéis estão mais aptos. No experimento em que RL é usado em conjunto com leilões, a falta de conhecimento inicial apresenta-se na forma do envio de lances com valores aleatórios para o algoritmo de leilões, resultando também em alocações deficientes. Porém, em ambos os casos, após algum tempo o desempenho do time se torna superior ao desempenho obtido sem o uso de RL. No caso do experimento Leilões + Q(λ), após 7000 jogos o resultado ultrapassou o do experimento apenas com leilões, e após 17000 jogos, a média de gols havia se tornado superior a 13 gols por partida, um aumento de mais de 50$\%$, em comparação com o algoritmo que empregou apenas leilões.

Observando-se os resultados do algoritmo com heurísticas, o Leilões + HAQ(λ), pode-se concluir que este não apresentou a deficiência inicial dos outros algoritmos com RL, indicando o benefício causado por possuir conhecimento inicial sobre a alocação de papéis. O algoritmo com heurísticas também obteve resultados superiores às outras variações, tanto em número médio de gols quanto no tempo necessário para atingir estes números, fatores que indicam que este ganho em desempenho é resultado do uso de heurísticas.

O capítulo seguinte apresenta as conclusões deste trabalho e sugestões para trabalhos futuros.

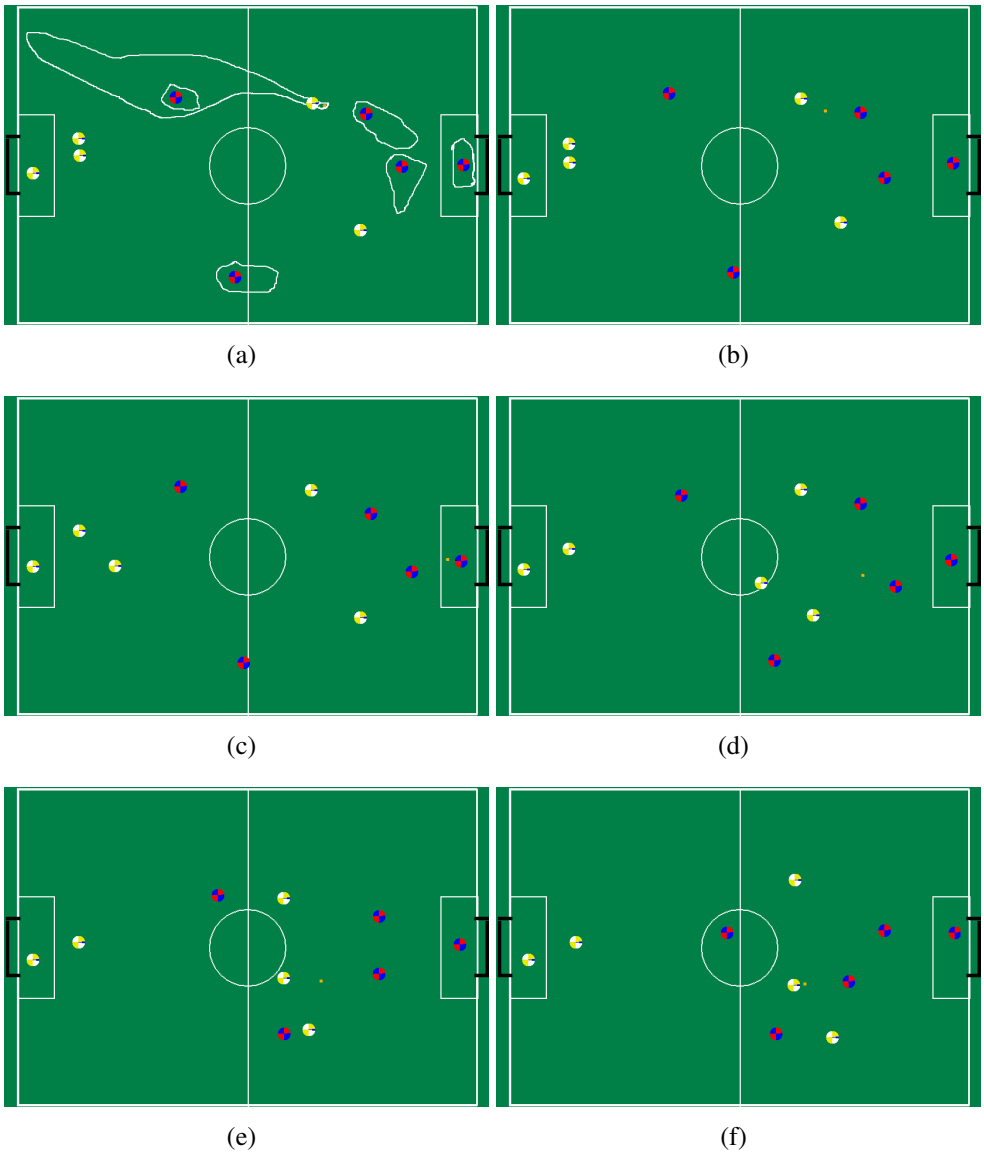


Figura 5.16: Registros (*logs*) de alguns segundos da partida semi-final do LARC2010, em ordem temporal. A equipe RoboFEI aparece em amarelo e branco. A Figura (a) mostra o momento em que um leilão ocorre, e um dos jogadores de defesa toma o papel de atacante. A sequência da jogada é: (a) Jogador amarelo com a posse de bola ruma ao gol. (b) Este jogador chuta a bola ao gol. (c) Goleiro do time azul defende, rebatendo a bola. Enquanto isso, jogador amarelo posicionado no campo defensivo ruma em direção à bola. (d) A bola rola em direção ao meio do campo (e) Jogador amarelo vindo do campo defensivo se aproxima da bola. (f) Este jogador vindo campo defensivo domina a bola e chuta novamente. O chute foi para fora do gol.

Capítulo 6

CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou uma arquitetura de alocação de tarefas em sistemas multi-robôs em que os agentes participam de leilões pelas funções de alto nível disponíveis e utilizam aprendizado por reforço para aprender o valor de cada uma destas funções, dada a situação em que a equipe de robôs se encontra. A arquitetura foi aplicada a uma equipe de futebol de robôs da categoria RoboCup Small Size. Foram comparados os desempenhos do mecanismo de alocação de tarefas quando agentes utilizavam valores de seus lances ajustados manualmente, quando os valores eram aprendidos por Aprendizado por Reforço e também por Aprendizado por Reforço acelerado por Heurísticas.

Os resultados dos experimentos mostram que, em um ambiente de modelagem complexa como uma partida de futebol de robôs, um sistema de alocação de tarefas é capaz de aumentar significativamente o desempenho da equipe, quando comparado com algoritmos em que o comportamento da equipe é pré-programado.

Os resultados sugerem também que o uso de Aprendizado por Reforço possibilita resultados superiores ao uso de funções utilidade manualmente criadas. Contudo, embora os resultados do uso de RL para a obtenção dos valores utilidade dos agentes tenham sido positivos, o tempo de convergência necessário é da ordem de muitos meses de tempo real, claramente tornando necessário o uso de ambientes simulados, onde o RL possa ser feito em tempos de execução superiores à execução em tempo real.

A principais contribuições deste trabalho foram a constatação da viabilidade de um sistema de alocação de tarefas para uma equipe robótica e a confirmação da eficácia de um algoritmo de RL acelerado por heurísticas, o HAQ(λ), quando utilizado neste sistema de MRTA. Os resul-

tados foram extraídos de um domínio dinâmico e com razoável número de agentes robóticos, fator que reforça a relevância da contribuição.

Outra contribuição deste trabalho foi a estruturação do sistema de estratégia da equipe RoboFEI Small Size de maneira modular e adequada à implementação de trabalhos futuros em inteligência artificial, o que contribuirá para o aumento da capacidade de pesquisa da equipe.

As necessidades computacionais desta pesquisa levaram também ao desenvolvimento de um simulador capaz de simular jogos em modo acelerado diversas ordens de magnitude acima do tempo real, fundamental para futuros trabalhos que utilizem aprendizado de máquina, além de diagnosticar falhas de software que poderiam somente aparecer em competições. Estas mesmas necessidades levaram também ao desenvolvimento de algoritmos de computação intensiva eficientes, utilizando pacotes de álgebra linear (BLAS) como o Intel Math Kernel Library e rotinas do OpenCV para computação de grandes matrizes. Este conhecimento foi incorporado ao projeto RoboFEI.

6.1 Trabalhos Futuros

Durante a pesquisa para este trabalho, diversas possibilidades para trabalhos futuros foram levantadas, e algumas são descritas a seguir.

A exploração de algoritmos de transferência do Aprendizado por Reforço, como descrevem *Taylor e Stone* (2009), podem possibilitar o uso do que foi aprendido em simulação com maior eficiência. Ainda no campo do Aprendizado por Reforço, análises aprofundadas da interferência causada pelo aprendizado simultâneo de diversos agentes podem gerar resultados superiores aos obtidos.

A exploração de algoritmos de leilão totalmente distribuídos e métodos que permitam a revenda de tarefas são também importantes possibilidades de trabalhos futuros, já que em alguns domínios, inclusive outros domínios em que a equipe RoboFEI pretende atuar, como a liga RoboCup humanoide, não existe comunicação com um computador que possa agir como leiloeiro.

Outra possibilidade de trabalho futuro está no uso de técnicas de decisão multiobjetivo (*Figueira et al.*, 2005) na criação tanto das métricas para os algoritmos econômicos quanto nos algoritmos de tomada de decisão existentes na camada de ações da estratégia da equipe. Existem trabalhos, tanto aplicados ao futebol de robôs (*Kyrylov*, 2007; *Kyrylov e Razykov*, 2008) quanto em outros domínios (*Marler e Arora*, 2004), com resultados promissores nesta área.

Por fim, o autor acredita que o uso de redes Bayesianas e outros métodos com modelos probabilísticos podem trazer avanços, tanto quando aplicados aos algoritmos econômicos quanto aos algoritmos de tomada de decisão utilizados.

Referências Bibliográficas

Albus, J. S., A new approach to manipulator control: the cerebellar model articulation controller, *Trans. of the ASME, Journal of Dynamic Systems, Measurement and Control*, *97*(3), 220–227, 1975.

Arkin, R. C., Motor schema based navigation for a mobile robot: An approach to programming by behavior, em *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 4, p. 264–271, 1987.

Balch, T., Teambots 2.0 documentation, http://www.cs.cmu.edu/trb/TeamBots, 2000.

Bertsekas, D., *Deterministic and Stochastic Models*, Prentice Hall, Upper Saddle River, NJ, 1987.

Bertsekas, D. P., A distributed algorithm for the assignment problem, *Lab. for Information and Decision Systems Report, MIT*, 1979.

Bertsekas, D. P., Auction algorithms for network flow problems: A tutorial introduction, *Computational Optimization and Applications*, *1*, 7–66, 1992.

Bianchi, R. A., C. H. Ribeiro, e A. H. Costa, Heuristically accelerated q-learning: A new approach to speed up reinforcement learning, em *Advances in Artificial Intelligence - SBIA 2004*, *Lecture Notes in Computer Science*, vol. 3171, editado por A. L. C. Bazzan e S. Labidi, p. 930–1008, Springer Berlin / Heidelberg, doi:10.1007/978-3-540-28645-5_25, 2004.

Bianchi, R. A. C., Uso de heurísticas para aceleração do aprendizado por reforço, Tese (Doutorado), Escola Politécnica da Universidade de São Paulo, 2004.

Bianchi, R. A. C., C. Ribeiro, e A. Costa, Accelerating autonomous learning by using heuristic selection of actions, *Journal of Heuristics*, *14*, 135–168, doi:10.1007/s10732-007-9031-5, 2008.

Botelho, S. C., e R. Alami, M+: a scheme for multirobot cooperation through negotiated task allocation and achievement, em *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, p. 1234–1239, 1999.

Botelho, S. C., e R. Alami, A multi-robot cooperative task achievement system, em *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, p. 2716–2721, 2000.

Bowling, M., B. Browning, e M. Veloso, Plays as effective multiagent plans enabling opponent-adaptive play selection, em *Proceedings of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, p. 376–383, 2004.

Browning, B., J. Bruce, M. Bowling, e M. Veloso, STP: Skills, tactics and plays for multi-robot control, *IEEE Journal of Control and Systems Engineering*, *219*, 33–52, 2005.

Bruce, J., S. Zickler, M. Licitra, e M. Veloso, Cmdragons: Dynamic passing and strategy on a champion robot soccer team, em *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Pasadena, CA, 2008.

Busoniu, L., R. Babuska, e B. De Schutter, A comprehensive survey of multiagent reinforcement learning, *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, *38*(2), 156–172, doi:10.1109/TSMCC.2007.913919, 2008.

Carpin, S., e L. E. Parker, Cooperative leader following in a distributed multi-robot system, *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE Int. Conf. on*, *3*, 2994–3001, doi:10.1109/ROBOT.2002.1013687, 2002.

Chaimowicz, L., M. F. M. Campos, e V. Kumar, Dynamic role assignment for cooperative robots, em *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, p. 293–298, 2002.

Dahl, T., M. Matarić, e G. Sukhatme, A machine learning method for improving task allocation in distributed multi-robot transportation, em *Complex Engineered Systems: Science Meets Technology*, editado por D. Braha, A. Minai, e Y. Bar-Yam, p. 307–337, Springer, 2006.

Dias, M., e A. Stentz, A comparative study between centralized, market-based, and behavioral multirobot coordination approaches, *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ Int. Conf. on*, *3*, 2279–2284 vol.3, doi:10.1109/IROS.2003.1249210, 2003.

Dias, M., R. Zlot, N. Kalra, e A. Stentz, Market-based multirobot coordination: A survey and analysis, *Proceedings of the IEEE*, *94*(7), 1257–1270, doi:10.1109/JPROC.2006.876939, 2006.

Dias, M. B., R. M. Zlot, M. B. Zinck, J. P. Gonzalez, e A. T. Stentz, A versatile implementation of the traderbots approach for multirobot coordination, em *Int. Conf. on Intelligent Autonomous Systems*, 2004.

Donald, B. R., J. Jennings, e D. Rus, Information invariants for distributed manipulation, *Int. Journal of Robotics Research*, *16*(5), 673–702, 1997.

Figueira, J., S. Greco, e M. Ehrgott (Eds.), *Multiple Criteria Decision Analysis: State of the Art Surveys*, Springer Verlag, 2005.

Gale, D., *The Theory of Linear Economic Models*, McGraw-Hill, 1960.

Gerkey, B., e M. Matarić, Sold!: auction methods for multirobot coordination, *Robotics and Automation, IEEE Transactions on*, *18*(5), 758–768, doi:10.1109/TRA.2002.803462, 2002.

Gerkey, B., e M. Matarić, Multi-robot task allocation: analyzing the complexity and optimality of key architectures, *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE Int. Conf. on*, *3*, 3862–3868 vol.3, doi:10.1109/ROBOT.2003.1242189, 2003.

Gerkey, B. P., e M. J. Matarić, A formal analysis and taxonomy of task allocation in multi-robot systems., *Int. Journal of Robotics Research*, *23*(9), 939–954, 2004.

Kaelbling, L. P., M. L. Littman, e A. W. Moore, Reinforcement learning: A survey, *Journal of Artificial Intelligence Research*, *4*, 237–285, 1996.

Kalra, N., D. Ferguson, e A. Stentz, Hoplites: A market-based framework for planned tight coordination in multirobot teams, *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE Int. Conf. on*, p. 1170–1177, 2005.

Kalra, N., D. Ferguson, e A. Stentz, A generalized framework for solving tightly-coupled multirobot planning problems, *Robotics and Automation, 2007 IEEE Int. Conf. on*, p. 3359–3364, doi:10.1109/ROBOT.2007.363991, 2007.

Kalyanakrishnan, S., Y. Liu, e P. Stone, Half field offense in RoboCup soccer: A multiagent reinforcement learning case study, em *RoboCup-2006: Robot Soccer World Cup X*, editado

por G. Lakemeyer, E. Sklar, D. Sorenti, e T. Takahashi, p. 72–85, Springer Verlag, Berlin, 2007.

Kose, H., U. Tatlidede, C. Mericli, K. Kaplan, e H. L. Akin, Q-learning based market-driven multi-agent collaboration in robot soccer, em *Proceedings of the Turkish Symposium on Artificial Intelligence and Neural Networks*, p. 219–2228, 2004.

Kube, C. R., e H. Zhang, Collective robotics: From social insects to robots, *Adaptive Behavior*, *2*(2), 189–219, 1993.

Kuhn, H. W., The hungarian method for the assignment problem, *Naval Research Logistics Quarterly*, *2*(1), 83–97, 1955.

Kyrylov, V., Balancing gains, risks, costs, and real-time constraints in the ball passing algorithm for the robotic soccer, em *RoboCup-2006: Robot Soccer World Cup X*, editado por G. Lakemeyer, E. Sklar, D. Sorenti, e T. Takahashi, p. 304–313, Springer Verlag, 2007.

Kyrylov, V., e S. Razykov, Pareto-optimal offensive player positioning in simulated soccer, em *RoboCup-2007: Robot Soccer World Cup XI*, p. 228–237, 2008.

Latzke, T., S. Behnke, e M. Bennewitz, Imitative reinforcement learning for soccer playing robots, em *RoboCup-2006: Robot Soccer World Cup X*, editado por G. Lakemeyer, E. Sklar, D. Sorenti, e T. Takahashi, p. 47–58, Springer Verlag, Berlin, 2007.

Lerman, K., C. Jones, A. Galstyan, e M. J. Mataríc, Analysis of dynamic task allocation in multi-robot systems, *Int. Journal of Robotics Research*, *25*(3), 225–241, doi:10.1177/0278364906063426, 2006.

Littlestone, N., e M. K. Warmuth, The weighted majority algorithm, em *Information and Computation*, vol. 108, p. 212–261, Elsevier, 1994.

Littman, M. L., Markov games as a framework for multi-agent reinforcement learning, em *Proceedings of the 11th Int. Conf. on Machine Learning (ML-94)*, p. 157–163, Morgan Kaufmann, New Brunswick, NJ, 1994.

Marler, R., e J. Arora, Survey of multi-objective optimization methods for engineering, *Structural and Multidisciplinary Optimization*, *26*, 369–395, 2004.

Martins, M. F., e R. A. C. Bianchi, Comparação de desempenho de algoritmos de aprendizado por reforço no domínio do futebol de robôs, em *SBAI'07, VIII Simpósio Brasileiro de Automação Inteligente*, 2007.

Martinson, E., e R. C. Arkin, Learning to role-switch in multi-robot systems, em *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, p. 2727–2734, 2003.

Matarić, M. J., Designing emergent behaviors: From local interactions to collective intelligence, em *From Animals to Animats 2, Second Int. Conf. on Simulation of Adaptive Behavior (SAB-92)*, editado por J. Meyer, H. Roitblat, e S. Wilson, p. 432–441, MIT Press, 1992.

Mataric, M. J., Issues and approaches in the design of collective autonomous agents, *Robotics and Autonomous Systems*, *16*, 321–331, 1995.

Matarić, M. J., Learning in behavior-based multi-robot systems: Policies, models, and other agents, *Cognitive Systems Research*, p. 81–93, 2001.

McMillen, C., e M. Veloso, Distributed, play-based role assignment for robot teams in dynamic environments, em *Proceedings of DARS*, Minneapolis, MN, 2006.

Parker, L. E., Alliance: an architecture for fault tolerant multirobot cooperation, *Robotics and Automation, IEEE Transactions on*, *14*(2), 220–240, doi:10.1109/70.681242, 1998a.

Parker, L. E., Toward the automated synthesis of cooperative mobile robot teams, em *In Proceedings of SPIE Mobile Robots XIII*, p. 82–93, 1998b.

Parker, L. E., Distributed intelligence: Overview of the field and its application in multi-robot systems, *Journal of Physical Agents*, *2*(1), 5–14, special issue on Multi-Robot Systems, 2008.

Parker, L. E., e F. Tang, Building multirobot coalitions through automated task solution synthesis, *Proceedings of the IEEE*, *94*(7), 1289–1305, doi:10.1109/JPROC.2006.876933, 2006.

Parker, L. E., B. Kannan, F. Tang, e M. Bailey, Tightly-coupled navigation assistance in heterogeneous multi-robot teams, em *Proceedings of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2004.

Peng, J., e R. J. Williams, Incremental multi-step Q-learning, em *Machine Learning*, p. 226–232, Morgan Kaufmann, 1996.

Ribeiro, C. H. C., R. Pegoraro, e A. H. R. Costa, Experience generalization for concurrent reinforcement learners: The minimax-qs algorithm, em *AAMAS 2002*, Bologna, Italy, 2002.

Russell, S. J., e P. Norvig, *Artificial Intelligence: A Modern Approach*, 3 ed., Prentice Hall, 2009.

Sandholm, T., An implementation of the contract net protocol based on marginal cost calculations, em *Eleventh National Conf. on Artificial Intelligence (AAAI-93)*, p. 256–262, 1993.

Sandholm, T., e S. Suri, Improved algorithms for optimal winner determination in combinatorial auctions and generalizations, em *Proceedings of the Seventeenth National Conf. on Artificial Intelligence*, p. 90–97, 2000.

Singh, S. P., e R. S. Sutton, Reinforcement learning with replacing eligibility traces, *Machine Learning*, *22*, 123–158, doi:10.1007/BF00114726, 1996.

Smith, R. G., The contract net protocol: High-level communication and control in a distributed problem solver, *IEEE Transactions on Computers*, *C-29*(12), 1104–1113, 1980.

Spiegel, M. R., *Estatística*, 3 ed., Makron Books, São Paulo, 1993.

Stone, P., e M. Veloso, Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork, *Artificial Intelligence*, *110*(2), 241–273, doi:http://dx.doi.org/10.1016/S0004-3702(99)00025-9, 1999.

Stone, P., R. S. Sutton, e G. Kuhlmann, Reinforcement learning for RoboCup-soccer keepaway, *Adaptive Behavior*, *13*(3), 165–188, 2005.

Sukthankar, G., e K. Sycara, Robust recognition of physical team behaviors using spatio-temporal models, em *AAMAS '06: Proceedings of the fifth Int. joint Conf. on Autonomous agents and multiagent systems*, p. 638–645, ACM, doi:http://doi.acm.org/10.1145/1160633.1160746, 2006.

Sutton, R. S., Learning to predict by the methods of temporal differences, em *Machine Learning*, vol. 3, p. 9–44, Morgan Kaufmann, 1988.

Sutton, R. S., Generalization in reinforcement learning: Successful examples using sparse coarse coding, em *Advances in Neural Information Processing Systems 8*, p. 1038–1044, MIT Press, 1996.

Sutton, R. S., e A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge,MA., 1998.

Tang, F., e L. E. Parker, Asymtre: Automated synthesis of multi-robot task solutions through software reconfiguration, *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE Int. Conf. on*, p. 1501–1508, 2005a.

Tang, F., e L. E. Parker, Distributed multi-robot coalitions through asymtre-d, *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ Int. Conf. on*, p. 2606–2613, 2005b.

Tang, F., e L. E. Parker, A complete methodology for generating multi-robot task solutions using asymtre-d and market-based task allocation, *Robotics and Automation, 2007 IEEE Int. Conf. on*, p. 3351–3358, doi:10.1109/ROBOT.2007.363990, 2007.

Tang, Y., e L. E. Parker, Towards schema-based, constructivist robot learning: Validating an evolutionary search algorithm for schema chunking, *Robotics and Automation, 2008. ICRA 2008. IEEE Int. Conf. on*, p. 2837–2844, 2008.

Taylor, M. E., e P. Stone, Transfer learning for reinforcement learning domains: A survey, *Journal of Machine Learning Research*, *10*(1), 1633–1685, 2009.

Tesauro, G., Temporal difference learning and TD-gammon, *Communications of the ACM*, *38*(3), 58–68, 1995.

Vail, D., e M. Veloso, Feature selection for activity recognition in multi-robot domains, em *AAAI'08, Twenty-third Conf. on Artificial Intelligence*, 2008.

Veloso, M., P. Stone, e M. Bowling, Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer, em *SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, vol. 3839, 1999.

Watkins, C. J. C. H., Learning from delayed rewards, Tese (Doutorado), University of Cambridge, 1989.

Watkins, C. J. C. H., e P. Dayan, Q-learning, em *Machine Learning*, vol. 8, p. 279–292, Morgan Kaufmann, 1992.

Weigel, T., W. Auerbach, M. Dietl, B. Dumler, J.-S. Gutmann, K. Marko, K. Muller, Bernhard-Nebel, e B. S. M. Thiee, CS Freiburg: Doing the right thing in a group, em *RoboCup 2000*, editado por P. Stone, G. Kraetzschmar, e T. Balch, Springer, 2001.

Weiss, G. (Ed.), *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999.

Werger, B., e M. J. Mataric, Broadcast of local eligibility for multi-target observation, em *5th Int. Symposium on Distributed Autonomous Robotic Systems (DARS)*, p. 347–356, 2000.

Whiteson, S., e P. Stone, Evolutionary function approximation for reinforcement learning, *Journal of Machine Learning Research*, *7*, 877–917, 2006.