Milton Roberto Heinen

Controle Inteligente do Caminhar de Robôs Móveis Simulados

São Leopoldo

Fevereiro de 2007

Milton Roberto Heinen

Controle Inteligente do Caminhar de Robôs Móveis Simulados

Monografia submetida a avaliação como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada.

Orientador:
Fernando Santos Osório

UNIVERSIDADE DO VALE DO RIO DOS SINOS
CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA INTERDISCIPLINAR DE COMPUTAÇÃO APLICADA

São Leopoldo

Fevereiro de 2007

Ficha catalográfica elaborada pela Biblioteca da
Universidade do Vale do Rio dos Sinos

H468c Heinen, Milton Roberto

Controle inteligente do caminhar de robôs móveis simulados / por Milton Roberto Heinen. – 2006.

121 f. : il. ; 30cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2006.

"Orientação: Prof. Dr. Fernando Santos Osório, Ciências Exatas e Tecnológicas".

1. Robótica móvel autônoma. 2. Aprendizado de máquina. I. Título.

CDU 004.896

Catalogação na Publicação:
Bibliotecária Vanessa Borges Nunes - CRB 10/1556

Dedico este trabalho à minha esposa,
Alessandra Huther Heinen, que sempre esteve
ao meu lado quando eu mais precisei.

Agradecimentos

Agradeço a todas as pessoas que me apoiaram durante esses anos, e tornaram este trabalho possível. Em especial, agradeço ao meu orientador, Dr. Fernando Santos Osório, por ter acreditado em meu trabalho e dado todo o apoio necessário para a realização do mesmo. Agradeço também a todos os professores do PIPCA, que me transmitiram os conhecimentos necessários para a realização desta tarefa.

Além disso, agradeço a toda a minha família, em especial os meus pais, à minha sobrinha Carolina, e principalmente à minha amada esposa Alessandra Huther Heinen, que sempre confiou em mim e me deu forças para prosseguir nesta longa jornada.

Resumo

O objetivo desta dissertação é propor, testar e avaliar o uso de técnicas de Aprendizado de Máquina (ML) na configuração automática do controle do caminhar de robôs com pernas. Para que este objetivo fosse atingido, um extensa pesquisa de técnicas do estado da arte foi realizada e descrita neste trabalho. Esta pesquisa permitiu a elaboração do modelo proposto, chamado de LegGen, que foi implementado em um protótipo. O protótipo modelo em questão permite a utilização de vários tipos de robôs, compostos de quatro, seis ou mais patas, e além disto permite a evolução da morfologia dos robôs.

Utilizando o protótipo, é possível a realização de experimentos com robôs autônomos dotados de pernas, em um ambiente virtual tridimensional realístico, através de simulações baseadas em física. Foi utilizada a biblioteca ODE (*Open Dynamics Engine*) para a simulação de corpos rígidos e articulações, permitindo assim simular forças agindo nas articulações (atuadores), gravidade e colisões, entre outras propriedades físicas dos objetos inseridos no ambiente 3D. O protótipo implementado também permite simular sensores integrados aos robôs, de modo a controlar seu estado, estabilidade e deslocamento.

Nesta dissertação, foram pesquisadas diversas técnicas de Aprendizado de Máquina para o controle autônomo de robôs articulados dotados de pernas. Para o controle das articulações durante o caminhar, quatro estratégias de controle foram propostas e implementadas: (i) um autômato baseado em uma tabela de ângulos; (ii) um autômato baseado em uma tabela de posições; (iii) funções cíclicas que descrevem a trajetória dos *endpoints* através de uma meia-elipse; e (iv) uma Rede Neural Artificial do tipo Elman. Estas estratégias de controle possuem diversos parâmetros, que são otimizados de forma automática através do uso de Algoritmos Genéticos, implementados com o uso da biblioteca GALib. Para acelerar a convergência e melhorar os resultados, quatro funções de *fitness* foram propostas e validadas utilizando o protótipo.

Através da realização de diversos experimentos, foi possível a validação das estratégias de controle propostas, e também foi possível realizar um estudo comparativo, que apresenta as vantagens e desvantagens de cada uma delas. Além disto, vários experimentos foram realizados comparando o desempenho de diferentes modelos de robôs, de forma que fosse selecionado o modelo mais eficiente para a tarefa em questão.

Todos os experimentos descritos foram validados através de métodos estatísticos e da análise dos dados através de gráficos, assim como foi realizada uma discussão e análise posterior sobre os resultados obtidos. Ao final deste trabalho, são descritas as principais contribuições do mesmo, bem como as perspectivas futuras.

Palavras-chave: Robótica Autônoma, Robôs Articulados com Pernas, Aprendizado de Máquina, Algoritmos Genéticos, Redes Neurais Artificiais, Controle Inteligente, Simulação Física, Ambiente Virtual 3D

Abstract

The main goal of this dissertation is to propose, to test and to evaluate the use of Machine Learning (ML) techniques in the automatic configuration of the gait control in legged robots. In order to achieve this goal, an extensive research about state-of-the-art techniques was accomplished and they are described in this work. This research allowed the development of the proposed model, called LegGen, which was implemented in a prototype. The proposed model allows the use of several different robot models with four, six or more paws. Besides that, the prototype allows also to study the robot's morphology evolution.

The implemented prototype allows to accomplish experiments with autonomous legged robots, in a realistic three-dimensional virtual environment, through physics based simulations. The ODE (Open Dynamics Engine) software library was used in the physical simulation of rigid bodies and articulations, allowing to simulate forces acting in the articulations (actuators), gravity and collisions, among other physical properties of the objects inserted in the 3D environment. The implemented prototype also simulates sensors integrated in the robots, in order to control its state, stability and displacement.

Several techniques of Machine Learning were studied to use in the control of the autonomous articulated legged robots. So, to control the articulations during the walk, four control strategies were proposed and implemented: (i) an automata based on angles tables; (ii) an automata based on positions tables; (iii) cyclic functions that describes the endpoints trajectory through a half-ellipse; and (iv) an Elman Artificial Neural Network. These control strategies have several parameters to configure, that are automatically obtained and optimized using Genetic Algorithms (GA). The GA were implemented into LegGen using the GALib software library. In order to improve the convergence and the gait control results, four fitness functions were proposed and validated using the prototype.

From the execution of several experiments, it was possible to validate the proposed strategies of control, and it was also possible to accomplish a comparative study, presenting the advantages and disadvantages of each strategy. Besides that, several other experiments were accomplished comparing the different robot models, so that the most efficient model, according to a specific task, can be selected to be implemented in hardware.

The experiments described in this work were validated through statistical methods and through graphical analysis of experimental data. A discussion about the experiments and posterior analysis of the obtained results, were also presented. Finally, the main contributions of this work are described, as well as the future work perspectives.

Keywords: Autonomous Robots, Articulated Legged Robots, Machine Learning, Genetic Algorithms, Artificial Neural Networks, Intelligent Control, Physical Simulation, 3D Virtual Environments

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviaturas

3D	–	Três dimensões, tridimensional
AI	–	*Artificial Inteligence* (Inteligência Artificial)
ANN	–	*Artificial Neural Networks* (Redes Neurais Artificiais)
API	–	*Aplication Programming Interface*
CBR	–	*Case Based Reasoning* (Sistemas baseados em casos)
CCD	–	*Charge-Coupled Device*
CGA	–	*Cyclic Genetic Algorithms* (Algoritmos Genéticos cíclicos)
CI	–	*Confidence Interval* (Intervalo de confiança)
CPG	–	*Central Pattern Generator* (Gerador central de padrões)
CTRNN	–	*Continuous Time Recurrent Neural Networks*
GA	–	*Genetic Algorithms* (Algoritmos Genéticos)
GAlib	–	*Genetic Algorithms Library* (Biblioteca de Algoritmos Genéticos)
GP	–	*Genetic Programming* (Programação Genética)
IDT	–	*Induction of Decision Trees* (Indução de árvores de decisão)
ILP	–	*Inductive Logic Programming* (Programação lógica indutiva)
MIT	–	*Massachusetts Institute of Technology*
ML	–	*Machine Learning* (Aprendizado de Máquina)
MLP	–	*Multi Layer Perceptron* (Perceptron de múltiplas camadas)
ODE	–	*Open Dynamics Engine* (ODE é uma biblioteca de simulação física)
PDW	–	*Passive Dynamic Walker* (Caminhador dinâmico passivo)
RBF	–	*Radial Basis Function* (Função de base radial)
RL	–	*Reinforcement Learning* (Aprendizado por Reforço)
SOM	–	*Self-Organizing Maps* (Mapas auto-organizáveis)
SVM	–	*Support Vector Machines* (Maquinas de vetor suporte)
TDNN	–	*Time Delay Neural Network* (Rede Neural com atraso temporal)
VRM	–	*Virtual Register Machine* (Máquina de registradores virtual)

1 Introdução

Os robôs móveis autônomos tem atraído a atenção de um grande número de pesquisadores, devido ao desafio que este novo domínio de pesquisas propõe: dotar os sistemas de uma capacidade de raciocínio inteligente e de interação com o meio em que estão inseridos (DUDEK; JENKIN, 2000). Os robôs móveis autônomos podem perceber o ambiente através da leitura de seus sensores (infravermelho, sonar, *lasers*, câmeras de vídeo, *bumpers*, giroscópios, acelerômetros, etc), e através desta percepção sensorial eles podem planejar melhor as suas ações (MEDEIROS, 1998; HEINEN, 1999).

Atualmente os robôs móveis atuam em diferentes áreas, como o desarmamento de bombas, a exploração de ambientes hostis, e a condução de veículos robotizados. Alguns exemplos de robôs móveis autônomos são: o sistema desenvolvido pelo NavLab da CMU (POMERLEAU, 1990; BATAVIA; POMERLEAU; THORPE, 1996), que é capaz de conduzir uma caminhonete pelas estradas americanas; os robôs do tipo *rover* enviados para Marte pela NASA (STONE, 1996); o robô Dante, que explora o interior de vulcões (LEMONICK, 1994); o sistema de controle de um veículo Ligier elétrico, desenvolvido pelos pesquisadores do INRIA na França (PAROMTCHIK; LAUGIER, 1996; LAUGIER et al., 1998); e o SEVA3D (Simulador de Estacionamento de Veículos Autônomos em um ambiente tridimensional), que realiza a tarefa de estacionamento de um robô do tipo carro em uma vaga paralela de forma automática (HEINEN et al., 2006a, 2006b, 2006c, 2006d, 2007). Todos esses sistemas possuem em comum a capacidade de receber leituras de sensores que lhes dão informações sobre o ambiente em que estão inseridos, e de modo semi ou completamente autônomo geram os comandos que fazem com que eles se desloquem no ambiente de modo seguro, ou seja, sem se chocar contra obstáculos ou colocar em risco a sua integridade ou a dos demais elementos presentes no ambiente.

1.1 Motivação e justificativa

Os robôs com rodas conseguem se deslocar com bastante eficiência em superfícies planas e regulares, o que os torna bastante úteis em diversas situações. Mas em ambientes projetados para os seres humanos (o interior de prédios e casas, por exemplo), os robôs com rodas nem sempre conseguem se deslocar livremente, pois estes ambientes possuem diversas irregularidades como inclinações, desníveis, escadas e degraus (KNIGHT; NEHMZOW, 2002). Em tarefas mais complexas, como a exploração de outros planetas e do interior de cavernas, robôs com rodas também não são muito eficientes, pois os ambientes nos quais eles devem atuar possuem vários acidentes geográficos que dificultam o deslocamento (KNIGHT; NEHMZOW, 2002).

Sendo assim, para que um robô consiga se deslocar livremente em um ambiente irregular, ele precisaria de mecanismos de locomoção mais complexos, como pernas, por exemplo (BE-

KEY, 2005). Outra motivação para se construir robôs com pernas é que estes permitem um maior grau de identificação com os seres humanos, além de permitir uma melhor compreensão das formas de locomoção dos seres vivos (PFEIFER; SCHEIER, 1999).

Entretanto, o controle do caminhar de robôs com pernas é uma tarefa bastante árdua, que exige a configuração de diversos parâmetros relativos ao caminhar. A configuração manual destes parâmetros pode exigir várias horas de um especialista humano, e os resultados obtidos são sub-ótimos e dependentes da arquitetura do robô (CHERNOVA; VELOSO, 2004). Assim, seria útil realizar a configuração dos parâmetros do caminhar de forma automática, através do uso de técnicas de Aprendizado de Máquina (*Machine Learning* – ML) (MITCHELL, 1997).

Uma das técnicas de Aprendizado de Máquina mais adequadas para a solução deste tipo de problema são os Algoritmos Genéticos (*Genetic Algorithms* – GA) (HOLLAND, 1975; MITCHELL, 1996), pois segundo a teoria da evolução das espécies de Darwin (1859), os mecanismos de locomoção dos seres vivos são fruto da Evolução Natural, o que torna o uso dos GAs biologicamente justificável. Do ponto de vista computacional, os GAs também são bastante adequados para a configuração do caminhar de um robô, pois (GOLDBERG, 1989): (a) conseguem realizar uma busca multi-critério em um espaço multi-dimensional, ou seja, eles podem otimizar não apenas a velocidade do caminhar, mas também a estabilidade ou algum outro critério; (b) não necessitam de informações locais para a correção do erro nem do cálculo do gradiente; e (c) se corretamente utilizados são capazes de escapar de mínimos locais. De fato, os Algoritmos Genéticos vem sendo aplicados com sucesso inclusive na evolução de soluções aproximadas de problemas NP-Completos (HEINEN; OSÓRIO, 2006a, 2006h).

Apesar de serem bastante adequados para a configuração automática do caminhar de robôs com pernas, os GAs não são muito eficientes se forem utilizados diretamente em robôs reais. Isto ocorre porque os Algoritmos Genéticos são técnicas de ML que exigem centenas (ou até milhares) de experimentos até que se chegue a uma solução razoável (WOLFF; NORDIN, 2003a). De fato, se forem utilizadas 100 gerações em um GA com uma população de 50 indivíduos (valores típicos), no total serão necessários 5000 experimentos com o robô real. Se cada experimento durar um minuto, serão necessárias 83,33 horas para a realização da evolução. Este uso prolongado do robô, além de causar um desgaste excessivo dos componentes, necessita de supervisão humana para tarefas como o reposicionamento e a troca/recarga das baterias (WOLFF; NORDIN, 2003a).

Outras técnicas de Aprendizado de Máquina, como o Aprendizado por Reforço (SUTTON; BARTO, 1998) (*Reinforcement Learning* – RL) também podem exigir milhares de experimentos até que se chegue a uma política razoável (ASADA et al., 2003; OGINO et al., 2004). Já o uso de técnicas de aprendizado supervisionado, como as Redes Neurais Artificiais (*Artificial Neural Networks* – ANN) com o algoritmo *back-propagation* ou alguma de suas variações, não são muito indicadas para esta tarefa, pois não é possível de se obter de antemão informações locais para a correção do erro (saídas desejadas) através do cálculo do gradiente (REEVE, 1999).

Para evitar os problemas relacionados com a realização de milhares de experimentos em um robô real, uma alternativa viável é a realização dos experimentos em robôs simulados através da utilização de alguma uma biblioteca de simulação baseada em física, como a ODE (*Open Dynamics Engine*) (SMITH, 2006), por exemplo. Esta biblioteca permite a realização de simulações de robôs móveis com bastante realismo do ponto de vista físico. Com isto, o aprendizado pode ser realizado com um robô simulado em um ambiente virtual, e assim se economizam muitas horas de treinamento e se evita o desgaste dos equipamentos (WOLFF; NORDIN, 2003a).

É importante ressaltar que embora o objetivo final da construção de robôs com pernas seja o deslocamento em superfícies irregulares, a maioria das pesquisas ainda se concentra no deslocamento em superfícies planas e regulares, pois este é um problema que ainda não foi completamente solucionado (BEKEY, 2005).

1.2 Objetivos

Este trabalho visa realizar uma pesquisa de técnicas do estado da arte relacionadas ao controle de robôs móveis dotados de pernas, utilizando técnicas de Aprendizado de Máquina (ML).

1.2.1 Objetivos gerais

- Analisar diferentes formas de controle do caminhar, apontando as vantagens e desvantagens de cada uma delas;
- Propor um modelo de controle do caminhar, baseado em técnicas de ML;
- Implementar o modelo proposto em um protótipo completo e funcional;
- Utilizar um ambiente de simulação virtual que permita a simulação de robôs móveis com bastante realismo do ponto de vista físico.

1.2.2 Objetivos específicos

- Pesquisar o uso de técnicas do estado da arte no controle inteligente do caminhar de robôs com pernas, avaliando as capacidades e limitações de cada uma delas;
- Pesquisar diversas técnicas de Aprendizado de Máquina (ML) (MITCHELL, 1997), de forma a verificar a aplicabilidade de cada uma delas na tarefa em questão;
- Propor algumas formas de controle do caminhar, baseadas em técnicas de ML;
- Propor um modelo de controle inteligente, robusto e eficiente;
- Implementar e disponibilizar uma ferramenta para a simulação e estudo de robôs com pernas[1], configurável e com grande realismo do ponto de vista físico;
- Realizar diversos experimentos visando comparar diferentes estratégias de controle e diferentes configurações de robôs.

1.3 Escopo do trabalho

Em se tratando de um trabalho multidisciplinar, o projeto da construção de controladores inteligentes para robôs com pernas enfocará o uso de tecnologias pertinentes, sem cobrir exaustivamente todas disciplinas envolvidas.

Com relação ao ambiente de atuação, este deve ser interno (*indoor*) e plano, tais como laboratórios, escritórios, fábricas, casas e apartamentos. Além disto, este ambiente deve ser estático e sem a presença de obstáculos fixos e móveis. Não se espera que o robô seja capaz de atuar em ambientes muito diferentes daqueles para os quais foi adaptado, mas deve possuir um certo

[1] A ferramenta de simulação desenvolvida, chamada de Simulador LegGen, está disponível para *download* no *site* http://www.inf.unisinos.br/~osorio/leggen.

grau de robustez frente à situações novas e inesperadas. Em relação ao tipo de deslocamento, o modelo proposto irá focar somente no deslocamento para frente e em linha reta.

Com relação aos robôs utilizados, estes serão compostos de quatro ou mais pernas, de forma que os robôs bípedes e humanóides estão fora do escopo deste trabalho no que diz respeito a simulação e controle implementados no protótipo. Além disto, os robôs utilizados serão simulados com a biblioteca ODE (SMITH, 2006), e não serão utilizados robôs reais, porque atualmente a instituição não possui o *hardware* necessário. Os sensores utilizados serão *encoders*, *bumpers*, giroscópios e acelerômetros simulados.

1.4 Contribuições

As principais contribuições deste trabalho são:

- A realização de uma extensa pesquisa bibliográfica de técnicas do estado da arte no controle inteligente do caminhar de robôs com pernas;
- O modelo de controle do caminhar proposto;
- A elaboração da função de *fitness*;
- A realização de diversos experimentos comparativos, visando determinar:
  - As técnicas de ML mais adequadas para o problema em questão;
  - A função de *fitness* mais eficiente;
  - Os modelos de robôs mais viáveis (número de pernas e articulações);
  - As formas de controle mais eficientes;
- A evolução da morfologia do robô.

1.5 Organização do trabalho

Esta dissertação está estruturada da seguinte forma:

- ***Capítulo 2 – Conceitos de robótica móvel:*** introduz diversos conceitos relativos à área de robótica móvel, como os sistemas de controle, sensores e o uso de simulação. Também são apresentados alguns pontos relativos aos robôs com pernas, como a estabilidade, o problema da cinemática inversa e os geradores centrais de padrões. O capítulo se encerra destacando os princípios básicos de um agente de *software*;
- ***Capítulo 3 – Aprendizado de Máquina:*** introduz a disciplina de Aprendizado de Máquina (ML), e discute algumas das técnicas utilizadas em aplicações de robótica móvel, principalmente os Algoritmos Genéticos (GA) e as Redes Neurais Artificiais (ANN);
- ***Capítulo 4 – Trabalhos relacionados:*** apresenta o estado da arte das técnicas utilizadas para o controle do caminhar em robôs com pernas. Além disto, são descritos alguns trabalhos prévios desenvolvidos pelo autor na área de robótica móvel autônoma;
- ***Capítulo 5 – Modelo proposto:*** descreve as diversas características do modelo proposto e do protótipo, as formas de controle do caminhar e os modelos de robôs utilizados;
- ***Capítulo 6 – Experimentos e resultados:*** apresenta o conjunto de experimentos realizados e discute os resultados obtidos;
- ***Capítulo 7 – Conclusões:*** conclui a dissertação com uma discussão final dos resultados e apresenta as perspectivas futuras.

2 Conceitos de robótica móvel

Neste capítulo serão apresentados diversos conceitos relativos a área de robótica móvel. A Seção 2.1 descreve as vantagens de se utilizar simulação em robótica móvel, o uso de simulação baseada em física e o simulador Webots. A Seção 2.2 descreve alguns conceitos relativos aos sistemas de controle de robôs móveis. A Seção 2.3 descreve diversos tipos de sensores possíveis de serem utilizados em robótica móvel. A Seção 2.4 descreve a estabilidade estática e dinâmica em robôs com pernas. A Seção 2.5 descreve o uso de Geradores Centrais de Padrões (CPGs) no controle das articulações. A Seção 2.6 descreve o problema da cinemática inversa, e algumas formas de contorná-lo. A Seção 2.7 descreve os princípios básicos que um agente de software (no nosso caso, um robô ou real ou simulado) deve seguir para ser considerado autônomo. A Seção 2.8 descreve a experiência anterior na área de robótica autônoma, e por último a Seção 2.9 traz as considerações finais do capítulo.

2.1 Simulação de robôs móveis

Quando se deseja realizar experimentos em robótica móvel, duas alternativas são possíveis: (i) realizar os experimentos diretamente em um robô real; ou (ii) realizar os experimentos utilizando um robô simulado em um ambiente virtual realista (PFEIFER; SCHEIER, 1999). A utilização de um robô real possui a vantagem de tornar realísticos os resultados obtidos, mas o uso de simulação possui as seguintes vantagens (LAW; KELTON, 2000):

- Na simulação não existe o risco de se danificar o robô;
- A troca ou recarga de baterias e a manutenção do robô não são necessárias;
- O reposicionamento do robô pode ser realizado sem a intervenção humana;
- O relógio da simulação pode ser acelerado, reduzindo assim o tempo de aprendizado;
- Pode-se testar várias arquiteturas e modelos diferentes de robôs antes da construção física, e assim descobrir com antecedência qual modelo de robô é mais eficiente.

Para o desenvolvimento de um simulador de robôs móveis, o uso de uma biblioteca de simulação baseada em física é bastante útil, como pode ser visto na próxima seção.

2.1.1 Simulação baseada em física

Para que uma simulação de robôs móveis seja realista, diversos elementos do mundo real precisam estar presentes no modelo de simulação, para que os corpos se comportem de forma similar à realidade. Em especial, é necessário que um robô sofra quedas se não for bem controlado ou se não estiver bem posicionado, e que colida contra os objetos de forma realista. Para

que isto ocorra, é necessário que as leis da física sejam modeladas no ambiente de simulação (gravidade, inércia, fricção e colisão) (OSÓRIO et al., 2006). Atualmente existem diversas bibliotecas de software disponíveis para se implementar este tipo de simulação. Uma das mais conhecidas é a ODE[1] (*Open Dynamics Engine*), descrita na próxima seção.

2.1.2 Biblioteca ODE

A ODE (*Open Dynamics Engine*), desenvolvida por Smith (2006), é uma biblioteca de software livre (*freeware* e *open source*) especialmente desenvolvida para a simulação da dinâmica de corpos rígidos articulados. Ela permite a criação de uma estrutura articulada, através da conexão de corpos rígidos de diversas formas utilizando articulações de vários tipos. A ODE foi projetada para ser utilizada de modo interativo e em simulações de tempo real, e é especialmente indicada para a simulação de objetos móveis em ambientes dinâmicos. Além disto, é possível mudar a estrutura do sistema, mesmo durante a simulação. A ODE utiliza um integrador de primeira ordem altamente estável, que evita que os erros de simulação cresçam de forma descontrolada. Isto permite que a ODE seja rápida, robusta e estável (OSÓRIO et al., 2006).

A simulação é baseada em um método no qual as equações de movimento são derivadas por meio de um modelo de velocidades baseado em multiplicadores de Lagrange (WOLFF; NORDIN, 2003a). A ODE possui juntas do tipo contato, que permitem a utilização de restrições de não-penetração sempre que dois corpos rígidos colidem. A ODE possui um sistema de detecção de colisões nativo, que suporta as seguintes primitivas de colisão: *sphere* (esfera), *box* (caixa), *capped cylinder* (cilindro com as extremidades arredondadas) e *plane* (plano, superfície). Outras características da ODE são a distribuição de massa arbitrária aos corpos rígidos, e um modelo de fricção/contato baseado no *Dantzig LCP solver* (BARAFF; WITKIN, 1997).

A ODE possui uma API (*Aplication Programming Interface*), escrita em linguagem de programação C (embora a ODE tenha sido desenvolvida principalmente em C++), e algumas otimizações específicas para diferentes plataformas. Uma simulação ODE típica ocorre da seguinte forma (WOLFF; NORDIN, 2003a):

1. Criação do mundo dinâmico;
2. Criação dos corpos rígidos no mundo dinâmico;
3. Ajuste do estado (posição e inclinação) dos corpos rígidos;
4. Criação das articulações no mundo dinâmico;
5. Conexão das articulações aos corpos rígidos;
6. Ajuste dos parâmetros de todas as articulações;
7. Criação do mundo colisivo e dos objetos geométricos neste mundo;
8. Criação de um grupo de articulações para armazenar as juntas do tipo contato;
9. Repetir:
   (a) Aplicação de forças aos corpos conforme a necessidade;
   (b) Ajuste dos parâmetros das articulações conforme a necessidade;
   (c) Execução da rotina de detecção de colisões;
   (d) Criação de juntas do tipo contato para todos os ponto de colisão;
   (e) Execução de um passo da simulação;
   (f) Remoção de todas as juntas do tipo contato;
10. Destruição do mundo dinâmico e do mundo colisivo.

[1] ODE – http://www.ode.org

Do ponto de vista físico, um robô é simplesmente um conjunto de corpos rígidos conectados através de diversas articulações. Cada um destes corpos pode interagir com os demais: uma força (ou torque) aplicada a um destes corpos também afeta os demais corpos conectados a este. Além disto, todos os corpos rígidos devem sofrer a ação da gravidade. As próximas seções descrevem de forma mais precisa os corpos rígidos e as articulações fornecidas pela biblioteca ODE.

Corpos rígidos

Segundo Smith (2006), um corpo rígido possui as seguintes propriedades dinâmicas:

- Um vetor de posição, que corresponde ao centro de massa do corpo;
- Uma velocidade linear;
- Uma orientação;
- Um vetor de velocidades angulares, que descrevem como a orientação do corpo muda com o passar do tempo.

Além destas propriedades dinâmicas, um corpo possui as seguintes propriedades estáticas:

- A massa do corpo;
- A posição do centro de massa (relativa ao corpo);
- Uma matriz de inércia que descreve como a massa se distribui no corpo.

A biblioteca ODE vem sendo amplamente adotada em diversas aplicações de simulação de corpos rígidos, como por exemplo os *softwares*: Juice, Webots, Dance e OpenSim. Além disto, a ODE pode ser facilmente integrada junto as *engines* gráficas usadas no desenvolvimento de jogos, simuladores e visualizadores 3D, como: OSG, CrystalSpace, Ogre3D e DarkBasicPro.

Articulações

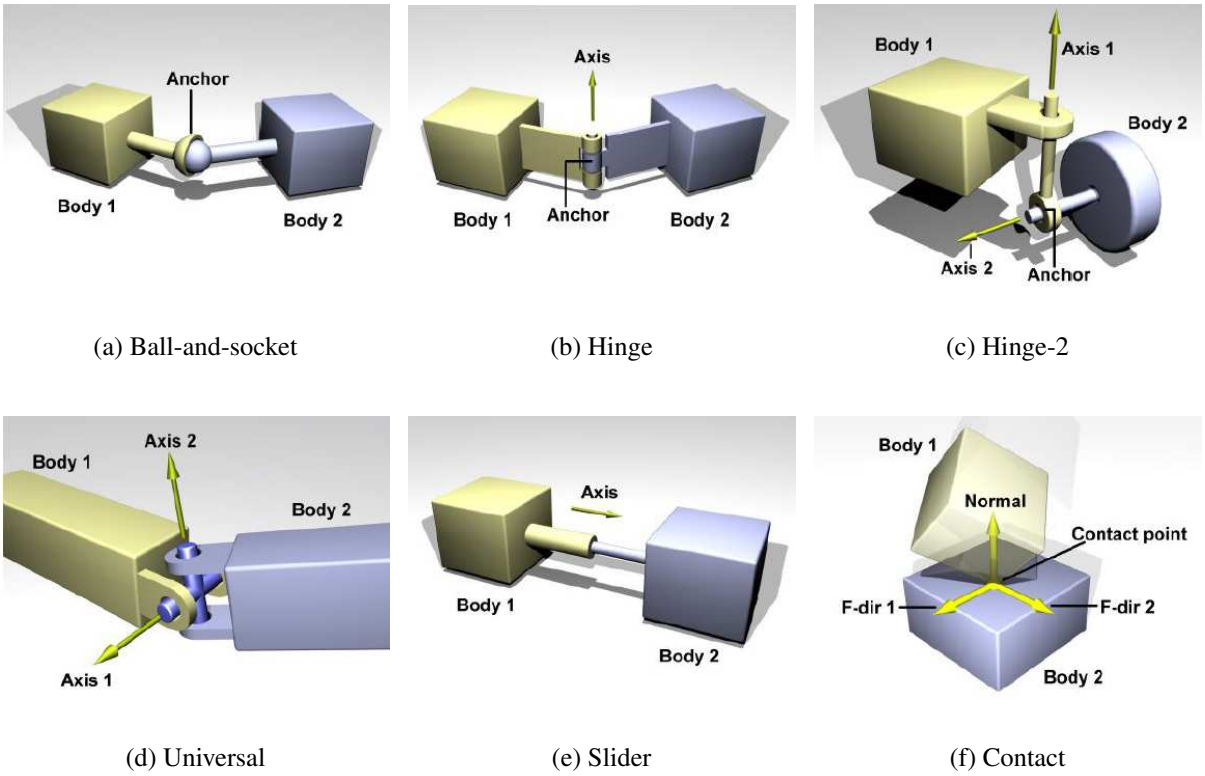
Os corpos rígidos, descritos na seção anterior, podem ser conectados através de uma grande variedade de articulações ou juntas. Os tipos de juntas implementados na ODE são *ball-and-socket* (similar ao nosso ombro), *hinge* (dobradiça ou joelho), *hinge-2*, *fixed* (fixa), *prismatic slider* (deslizante) e *angular motor* (motores angulares). As juntas do tipo *hinge-2* são iguais a duas dobradiças ligadas em série com diferentes eixos de rotação. A Figura 2.1 mostra alguns tipos de articulações disponíveis na ODE.

Desta forma, a ODE consegue tornar o ambiente virtual bastante realístico, pois os robôs simulados não são apenas figuras geométricas, mas interagem com o ambiente de forma coerente com as leis da física. Este realismo do ponto de vista físico é essencial nas pesquisas em robótica, onde o objetivo final é a construção de robôs reais. Outra maneira de se obter ambiente virtual realístico é utilizando um simulador como o Webots, descrito na próxima seção.

2.1.3 Simulador Webots

O simulador Webots[2] (Figura 2.2) desenvolvido e comercializado pela Cyberbotics Ltd, é um simulador de robôs móveis, baseado na biblioteca ODE, que possui um ambiente tridimen-

[2]Webots - http://www.cyberbotics.com/products/webots/



(a) Ball-and-socket (b) Hinge (c) Hinge-2

(d) Universal (e) Slider (f) Contact

Figura 2.1: Articulações disponíveis na ODE (SMITH, 2006)

sional que permite a modelagem, a programação e a simulação de robôs móveis autônomos (MICHEL, 2004). Para cada objeto presente no ambiente virtual, podem ser definidas diversas propriedades, como forma, cor, textura, massa, fricção, etc. Cada robô pode ser equipado com diversos tipos de sensores e atuadores, e vários robôs podem compartilhar o mesmo ambiente virtual e interagirem entre si. O simulador Webots também permite que os sistemas de controle possam ser transferidos para vários robôs reais comercialmente disponíveis, como os robôs Khepera, Alice, e Sony SDR-4X e Aibo (MICHEL, 2004).

Figura 2.2: Simulador Webots (MICHEL, 2004)

Apesar do simulador Webots ser um dos mais avançados existentes para a área de robôs móveis, seu custo elevado impossibilita sua utilização em muitas pesquisas, e não permite a personalização dos sensores (HEINEN, 2002).

O uso de softwares de simulação permite que seja criado um ambiente virtual bastante realista, no qual podem ser realizados experimentos em robótica móvel. Mas para os robôs possam se movimentar no ambiente virtual, eles precisam de algum tipo de sistema de controle, que pode ou não ser dotado de uma certa autonomia e inteligência.

2.2 Sistemas de controle de robôs móveis

A tarefa do sistema de controle é fazer com que todo o sistema alcance um determinado estado. Alcançar este estado pode envolver ou depender de mudanças que ocorrem no ambiente, no sistema controlado ou devido a interação entre os dois. Logo um sistema de controle é um processo que pode utilizar seus sensores para obter informações sobre o sistema controlado e sobre o ambiente (HEINEN, 2002). Ele pode utilizar este conhecimento para controlar seus atuadores, fazendo com que todo o sistema alcance um determinado estado.

2.2.1 Estratégias de controle

Um sistema de controle pode utilizar sensores inseridos no sistema controlado ou no ambiente, mas isso é opcional. É possível que o sistema de controle utilize sensores somente no sistema controlado, somente no ambiente, ou em lugar nenhum. Existem diferentes maneiras de se obter informações sobre o estado do ambiente e do sistema controlado. As técnicas mais utilizadas são (HEINEN, 2002; HEINEN; OSÓRIO, 2002):

- Sistemas de controle *open loop*: não utilizam nenhum sensor;
- Sistemas de controle *feedforward*: utilizam sensores somente para perceber o ambiente. Neste tipo de sistema de controle, medições do ambiente são utilizadas para atualizar variáveis no modelo do sistema;
- Sistemas de controle *feedback*: monitoram continuamente a situação dos sensores e ajustam seus atuadores de acordo.

2.2.2 Arquiteturas de controle

Uma arquitetura de controle pode ser definida como uma abstração do sistema de controle, e o sistema de controle pode ser considerado uma realização da arquitetura. As principais arquiteturas de controle utilizadas em robôs móveis autônomos são (HEINEN, 2002):

Arquitetura horizontal: neste tipo de arquitetura as tarefas do sistema de controle são divididas em várias sub-tarefas baseadas em suas funcionalidades, de forma que as tarefas são realizadas em várias etapas. Primeiro, as entradas sensoriais são utilizadas para modificar a representação interna do ambiente. Segundo, baseado nesta representação um plano a longo prazo é elaborado. Isto resulta em uma série de ações que o robô deve executar para alcançar o seu objetivo. Terceiro, esta série de ações é utilizada para comandar os atuadores do robô;

Arquitetura vertical: nesta arquitetura, ao invés das tarefas serem divididas em função da funcionalidade, a divisão é realizada baseando-se em comportamentos que executam tarefas, organizados em camadas. Um exemplo de arquitetura vertical é a arquitetura *Subsumption*, introduzida por Brooks (1986), na qual o sistema de controle é constituído de diversos comportamentos

executados em paralelo. Cada comportamento gera as suas saídas diretamente para os atuadores utilizando as entradas sensoriais. As saídas sugeridas pelo comportamento com a mais alta prioridade são então utilizadas para controlar os atuadores do robô.

Desta forma, para que um robô móvel possa se deslocar, é preciso que o sistema de controle acione os atuadores de forma adequada, fazendo com que o robô se desloque no ambiente de forma segura e robusta. Para que isto ocorra, o robô precisa perceber o ambiente, o que pode ser obtido através do uso de diversos tipos de sensores, como os descritos na próxima seção.

2.3 Sensores

Os sensores são usados em robótica móvel para que seja possível perceber o ambiente, e assim poder comandar os atuadores de forma adequada. Para um melhor desempenho, um robô pode utilizar vários sensores ao mesmo tempo, integrando os dados destes sensores e fazendo com que seus atuadores se comportem de forma correta. Abaixo são descritos diversos tipos de sensores que podem ser utilizados em robótica móvel.

2.3.1 Câmeras de vídeo

Um robô móvel pode utilizar câmeras de vídeo para perceber o ambiente da seguinte forma: a imagem captada é processada pelo robô, que analisa e decide o que fazer. As imagens podem ser coloridas, preto e branco ou em tons de cinza, sendo que a imagem colorida demanda um maior tempo de processamento (HEINEN, 1999). Para o processamento das imagens, um método de reconhecimento de padrões precisa ser utilizado. O robô analisa cada imagem que ele obtém da câmera para identificar certos objetos, comparando esta imagem com os padrões armazenados na memória. Para que a visão seja tridimensional, pode-se utilizar duas câmeras ao mesmo tempo, o que permite calcular a distância dos objetos (DUDEK; JENKIN, 2000).

2.3.2 Infravermelho

Este tipo de sensor é muito utilizado, principalmente por ter baixo custo e um funcionamento relativamente simples, apesar de possuir um raio de ação bastante reduzido. O funcionamento deste tipo de sensor ocorre da seguinte forma: um diodo infravermelho emite um raio modulado; o raio atinge um objeto e uma porção da luz refletida é captada de volta através do receptor ótico, atingindo um vetor de foto-diodos; dependendo da posição do objeto, o tempo de resposta entre emissão e recepção e o ângulo de incidência da luz refletida são diferentes, e com isso pode-se calcular a distância deste objeto por triangulação (HEINEN, 1999).

2.3.3 Laser

O *laser* utiliza o mesmo princípio dos sensores infravermelhos, onde um feixe de luz é emitido, um foto-sensor capta a sua reflexão e é calculado o tempo que foi preciso para a luz retornar. Uma desvantagem é que os circuitos precisam ser muito rápidos e precisos, pois a velocidade do *laser* é muito alta (HEINEN, 1999). O *laser* também pode utilizar espelhos para

detectar um obstáculo. Um motor controla o ângulo do espelho até que o feixe do *laser* atinja o foto-sensor, e quando isto ocorre pode-se calcular a distância usando o ângulo do espelho por triangulação (DUDEK; JENKIN, 2000).

2.3.4 Sonar

O sonar é um dos sensores mais utilizados em robótica móvel devido ao seu baixo custo e a necessidade de poucos recursos computacionais (HEINEN, 2002). Seu funcionamento consiste em um transdutor que emite uma onda sonora de alta freqüência, e quando esta onda atinge um objeto, ela se reflete e é captada novamente pelo transdutor. À distância entre o transdutor e o objeto pode ser calculada pelo tempo entre a emissão e o recebimento da onda de som.

2.3.5 Giroscópios

Um giroscópio (BORENSTEIN et al., 1997) é um dispositivo utilizado para medir e/ou manter a orientação. Ele consiste de um rotor suspenso por um suporte formado por dois círculos articulados, com juntas tipo *cardan*. Seu funcionamento baseia-se no princípio da inércia. O eixo em rotação guarda a direção fixa em relação ao espaço, e com isto pode-se detectar as variações de direção que ocorrem em um corpo físico. Existem giróscopios completos, que atuam em todas as direções, e giroscópios simples, que atuam em apenas um sentido.

Nos vôos espaciais, o giroscópio é fundamental para manter a orientação das espaçonaves. Em robôs com pernas, o giroscópio pode ser usado para medir a instabilidade do caminhar, ajudando assim a manter a estabilidade do robô (BEKEY, 2005).

2.3.6 Acelerômetros

Um acelerômetro (BORENSTEIN et al., 1997) é um dispositivo projetado para calcular a aceleração a ao longo de um determinado eixo, pela medida da força F, exercida ao longo desse eixo, sobre uma dada massa m, usando a segunda lei do movimento de Newton: $F = m \times a$. Ele pode ser considerado, em sua forma mais simples, como uma massa suspensa por um fio (um pêndulo), ou que pode correr ao longo de um guia reto. Estando o suporte do pêndulo ou do guia em repouso, ou em movimento retilíneo uniforme, a massa estará em seu ponto neutro. Se o suporte inicia algum movimento ou altera a sua velocidade, a massa se desloca da posição neutra, e a quantidade de deslocamento é proporcional ao valor da aceleração. A medida do deslocamento é realizada por meios elétricos, pois assim conseguem-se detectar tanto pequenas quanto grandes acelerações. Em robôs móveis autônomos, o acelerômetro pode ser usado para detectar variações bruscas de velocidade, que são um indicativo de que o robô pode vir a sofrer alguma queda (BEKEY, 2005).

2.3.7 Inclinômetros

Inclinômetros são sensores que medem o grau de inclinação do robô em relação ao eixo de gravidade (DUDEK; JENKIN, 2000). Eles funcionam de forma similar aos instrumentos utilizados na construção cívil (nível e plumo) para medir o grau de inclinação das superfícies. Em

robótica móvel, um inclinômetro fornece informações que podem ajudar a manter o equilíbrio do robô.

2.3.8 Bumpers

Bumpers são sensores utilizados em robótica para simular o sentido do tato. Existem vários modelos de *bumpers*, os mais simples são interruptores que retornam um valor binário: aberto ou fechado (DUDEK; JENKIN, 2000). Os *bumpers* geralmente são utilizados em *end-effectors*, como manipuladores robóticos, por exemplo.

2.3.9 Encoders

Os *encoders* são sensores proprioceptivos de posição angular, que permitem ao sistema de controle conhecer com precisão os ângulos de cada uma das juntas do robô. Ao contrário dos tipos de sensores descritos acima, os *encoders* não fornecem informações relativas ao ambiente, mas sim informações relativas ao estado do próprio robô (DUDEK; JENKIN, 2000).

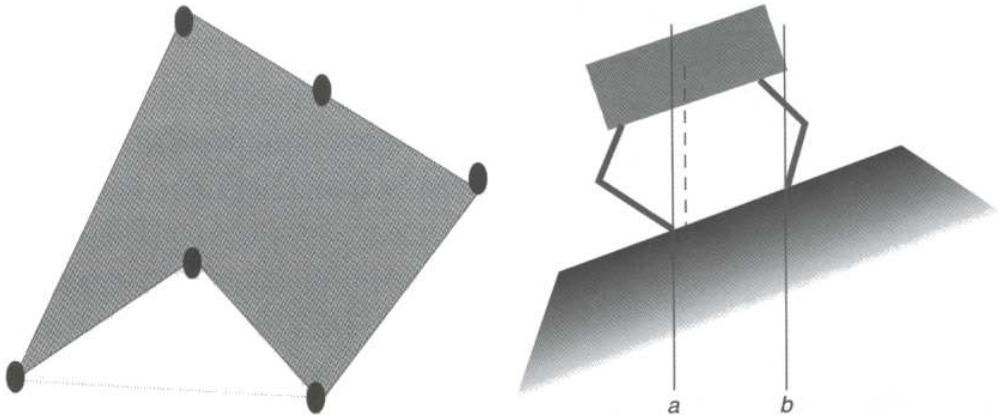
Através do uso de sensores, um robô móvel pode perceber o ambiente e assim agir sobre ele de forma adequada. Além disto, um sistema de controle precisa manter a estabilidade do robô durante o caminhar, através da movimentação das articulações de forma coordenada. A próxima seção descreve alguns conceitos relativos à estabilidade em robôs com pernas.

2.4 Estabilidade

Para que um robô consiga se deslocar livremente sem sofrer quedas, é necessário que o caminhar seja estável, e esta estabilidade pode ser obtida de forma estática ou dinâmica (DUDEK; JENKIN, 2000). Quando um robô se desloca de forma que o seu centro de gravidade nunca fique fora do polígono de suporte formado pelas pernas que estão em contato com o solo, é dito que o robô apresenta estabilidade estática. A Figura 2.3 ilustra esta situação. A Figura 2.3(a) mostra um exemplo de polígono de suporte formado pelas patas de um robô que estão em contato com o solo (seis patas, neste caso), e a Figura 2.3(b) mostra o centro de gravidade de um robô sobre o polígono de suporte. A principal vantagem da estabilidade estática é que o risco do robô sofrer quedas é menor, pois as patas que estão em contato com o solo conseguem garantir a estabilidade mesmo no caso de uma falha de energia ou se as baterias se descarregarem.

Se durante o caminhar o centro de gravidade do robô se deslocar periodicamente para fora do polígono de suporte, e mesmo assim o robô conseguir se movimentar de forma controlada, é dito que este robô apresenta estabilidade dinâmica. A estabilidade dinâmica é mais difícil de ser atingida, pois exige um sofisticado modelo da dinâmica do robô e do uso da inércia (DUDEK; JENKIN, 2000; BEKEY, 2005).

A estabilidade depende diretamente do número de pernas do robô. Para que um robô apresente estabilidade estática, o número mínimo de pernas necessário são quatro, e o robô precisa se deslocar deixando sempre três patas em contato com o chão. Já um robô de seis pernas consegue apresentar estabilidade estática mantendo apenas a metade de suas patas em contato com o chão, o que faz com que ele possa se deslocar mais rapidamente sem correr o risco de cair.



(a) (b)

Figura 2.3: Polígono de suporte de um robô (DUDEK; JENKIN, 2000)

De acordo com um estudo realizado por Muybridge (1957), abordando as formas de caminhar dos cavalos, estes animais possuem oito formas diferentes de caminhar, sendo que apenas uma (a mais lenta) é estável estaticamente. No trote, que é um passo de velocidade moderada, o cavalo afasta do chão duas pernas ao mesmo tempo, e no galope, que é o andar mais rápido, existem momentos em que o cavalo fica com as quatro patas no ar.

A coordenação das juntas durante o caminhar é uma tarefa bastante complexa, que exige a integração de diversos fatores como a estabilidade, as forças que atuam sobre o agente e as características do terreno. Nos seres vivos, esta coordenação é realizada através de grupos de neurônios especiais, chamados de geradores centrais de padrões.

2.5 Geração de padrões rítmicos

Os geradores centrais de padrões (*central pattern generator* - CPG) são grupos de neurônios que produzem padrões rítmicos de forma endógena, ou seja, sem a necessidade de estímulos oscilatórios ou de coordenação central. Eles são responsáveis pela produção dos padrões motores rítmicos da maioria dos seres vivos (MARDER; CALABRESE, 1996; STEIN et al., 1997). Uma das principais características dos CPGs é que a geração dos padrões não depende da atuação do sistema nervoso como um todo, mas sim de pequenos grupos de neurônios (os CPGs) que trabalham de forma autônoma (HOOPER, 2001).

Nos seres vivos, os CPGs utilizados na locomoção se encontram na espinha dorsal. Estes recebem sinais relativamente simples do sistema nervoso central, que controlam a velocidade e a direção do deslocamento (BUCHLI; IJSPEERT, 2004). Para a produção dos padrões básicos, geralmente não é necessário *feedback* sensorial, embora ele seja muito importante na adaptação dos padrões de acordo com a situação enfrentada pelo ser vivo (GRILLNER, 1981, 1985).

Atualmente, diversos modelos de robôs com pernas (reais e simulados) utilizam sistemas inspirados em CPGs para o controle das juntas durante o caminhar. Estes CPGs são geralmente implementados através de redes oscilatórias e/ou Redes Neurais Artificiais (ANN). O Capítulo 4 descreve alguns trabalhos que realizam o controle do caminhar através de CPGs.

2.6 Cinemática inversa

O Problema da cinemática inversa é fundamental no controle de robôs manipuladores e de robôs com pernas. Normalmente, a tarefa é especificada em um espaço cartesiano, enquanto que os controladores atuam sobre atuadores de junta, requerendo que as referências de controle sejam especificadas em espaço de juntas (HEINEN et al., 2001). Deste modo, torna-se necessário mapear referências especificadas em espaço cartesiano em referências equivalentes em espaço de juntas (LEMKE et al., 2002; SCIAVICCO; SICILIANO, 1996). Assim, o problema da cinemática inversa pode ser estabelecido da seguinte forma: especificada a localização (posição e orientação) desejada para o *endpoint*, determinar o valor das variáveis de junta (ângulos ou deslocamentos de junta) necessários para levar o *endpoint* a tal localização (CRAIG, 1989; WAGNER, 2003).

A localização do *endpoint* é uma função não linear composta de diversas equações linearmente independentes, que devem ser resolvidas para as variáveis de junta. Ao contrário da cinemática direta, o problema da cinemática inversa não é trivial e tem um complicador adicional: o mapeamento da localização do *endpoint* para os ângulos ou deslocamentos de junta não é um-para-um. Assim, dois problemas adicionais devem ser levados em conta: (i) a verificação da existência de solução; (ii) a possibilidade de existirem múltiplas ou infinitas soluções para uma dada localização do *endpoint* (LEMKE et al., 2002; WAGNER, 2003). Para solucionar o problema da cinemática inversa, existem as seguintes alternativas (SCIAVICCO; SICILIANO, 1996):

- Solução numérica: As equações não lineares simultâneas podem ser resolvidas por métodos iterativos. Por utilizar métodos iterativos, esquemas baseados neste tipo de abordagem podem ter problemas de convergência. Assim, estes esquemas não são muito indicados para implementações em tempo real, nas quais a cinemática inversa precisa ser calculada a taxas elevadas;
- Solução em fórmula fechada: As equações são resolvidas por métodos algébricos, freqüentemente fazendo uso de considerações geométricas, resultando em uma expressão analítica computável. Este tipo de abordagem resulta em soluções fáceis de se implementar e que envolvem pouco esforço computacional, encorajando aplicações em tempo real.

Neste trabalho, o cálculo da cinemática inversa foi realizado através do método de Powell (BRENT, 1973; ACTON, 1970), que apesar de ser um método iterativo, apresentou bons resultados nos experimentos realizados. Em robôs reais, uma solução em fórmula fechada seria mais indicada para o cálculo da cinemática inversa. É importante ressaltar que nem sempre é possível se calcular a cinemática inversa através de fórmula fechada, devido aos dois problemas descritos anteriormente.

2.7 Princípios básicos de um agente de software

No âmbito da Inteligência Artificial, um agente de software contemporâneo é visto como um sistema dinâmico, onde a percepção e a ação constituem processos simultâneos e inseparáveis (PFEIFER; SCHEIER, 1994). Existe, então, uma substituição da metáfora do processamento de informação, advinda do paradigma tradicional das ciências cognitivas, pela metáfora da coordenação sensório-motora (SCHEIER; PFEIFER, 1995; PFEIFER; SCHEIER,

1997). A seguir são descritos os princípios de projeto empregados na construção de agentes de software contemporâneos. Cabe salientar que ainda não existe nenhum agente que implemente todos estes princípios (FLORIAN, 2003). No entanto, estes princípios sumarizam e tornam explícito os conhecimentos adquiridos na área até o presente momento.

2.7.1 Princípio do agente absoluto

O projeto de um agente de software sempre envolve a definição de três componentes que são rigorosamente interconectados e mutuamente interdependentes (PFEIFER; SCHEIER, 1999):

1. Fixação de um nicho ecológico onde o agente irá atuar;
2. Estabelecimento de comportamentos desejados ou tarefas a serem cumpridas;
3. Determinação do agente propriamente dito.

A natureza de ambientes que um agente pode habitar varia significativamente. Nenhum agente pode adaptar-se, fisicamente e cognitivamente, para lidar com todas as variações possíveis. Por essa razão, um nicho ecológico deve ser fixado anteriormente ao projeto do agente. A partir de um dado nicho ecológico, são estabelecidos comportamentos ou tarefas a serem solucionadas pelo agente. Dessa forma, o agente pode ser projetado em conformidade com estas necessidades.

Por fim, a concretização de um agente deve priorizar quatro aspectos principais (PFEIFER; IIDA; BONGARD, 2005): autonomia (i.e., deve ser capaz de funcionar com pouquíssima intervenção, supervisão ou instrução humana), auto-suficiência (i.e., deve ser apto a sustentar-se por um período de tempo prolongado), localidade (i.e., deve adquirir informação sobre o ambiente somente através de seus próprios sensores), e corporificação (i.e., deve ser realizado como um sistema físico ou computacional). Apesar de certa objeção por alguns pesquisadores da área, existe consenso da maioria de que a corporificação não é necessariamente dada pela materialidade, como a apresentada em animais ou robôs físicos, mas por uma relação dinâmica com o ambiente. Diante disso, a pesquisa em Inteligência Artificial também pode ser realizada com ambientes de simulação genuinamente computacionais, desde que estes ambientes sejam realísticos do ponto de vista físico (FLORIAN, 2003; PFEIFER; SCHEIER, 1999).

2.7.2 Princípio da redundância e do equilíbrio ecológico

De acordo com as abordagens contemporâneas da cognição, não basta um agente de software possuir variados mecanismos de obtenção de informação, é necessário também incorporar redundância nos dispositivos sensoriais. Em outras palavras, os sensores devem estar posicionados no agente de tal forma que exista sobreposição espacial nas informações adquiridas. A redundância promove correlações e associações entre as informações obtidas por diferentes modalidades sensitivas. Estas correlações ajudam o agente a reduzir drasticamente a incerteza do ambiente e a predizer eventos (PFEIFER; SCHEIER, 1999). Além da redundância, deve existir um equilíbrio da complexidade do agente (sistemas sensório, motor e de controle) com a complexidade de seu ambiente de tarefa. Um sistema de controle extremamente complexo é desnecessário se o agente ou o ambiente são extremamente simples. Por outro lado, um agente com um sistema de controle muito simples pode ter dificuldades em adaptar-se às circunstâncias ambientais (PFEIFER; SCHEIER, 1999).

2.7.3 Princípio dos processos paralelos e fracamente acoplados

Agentes de software devem possuir diversos comportamentos a fim de cumprir determinadas tarefas. Alguns comportamentos são compatíveis, mas outros são mutuamente exclusivos. Por causa disto, uma decisão deve ser tomada para selecionar, a cada momento, aquelas ações que são coerentes com o contexto atual do agente e do ambiente (BROOKS, 1986).

Um mecanismo que vem despontando como uma nova opção para esta problemática é assumir um número elevado de processos heterogêneos, paralelos e fracamente acoplados que são conectados ao aparato sensório-motor do agente (PFEIFER; SCHEIER, 1999). Estes processos não necessitam de um supervisor; ou seja, o controle é descentralizado e distribuído. A arquitetura de controle pode ser construída de forma gradual, com a adição de novos processos, assim como acontece na evolução biológica. Além disso, a união de todos os processos promove a emergência de novos comportamentos, os quais não foram previstos no projeto.

2.7.4 Princípio da coordenação sensório-motora

Nesta última década, o estudo da integração sensório-motora tem sido um tópico de pesquisa bastante ativo na área das ciências cognitivas (PFEIFER; SCHEIER, 1999). Existe consenso entre os pesquisadores da área que, em organismos biológicos, as informações oriundas de múltiplos sentidos (visão, audição, tato e etc.) são integradas e, diretamente, mapeadas sobre um conjunto apropriado de comandos motores (músculos e glândulas). Este processo sensório-motor, quando aplicado a agentes de software, conduz a inúmeros benefícios e simplificações de projeto (SCHEIER; PFEIFER, 1995; PFEIFER; SCHEIER, 1997; NOLFI, 2002).

Pfeifer e Scheier (1997) demonstraram que, através da coordenação sensório-motora, agentes de software podem estruturar suas percepções e, por meio disto, induzir regularidades que significativamente simplificam a aprendizagem. Dados sensoriais não são apenas adquiridos mas, sobretudo, gerados e correlacionados. A correlação reduz a alta dimensionalidade presente nos dados obtidos do ambiente que, por sua vez, capacita o agente a fazer associações entre diferentes modalidades sensitivas (TE BOEKHORST; LUNGARELLA; PFEIFER, 2003). A coordenação sensório-motora possibilita, ainda, resolver um dos maiores desafios dos roboticistas: a categorização de objetos físicos (SCHEIER; LAMBRINOS, 1996).

2.7.5 Princípio da aprendizagem

Os seres vivos têm a capacidade de aprender. O aprendizado, do ponto de vista da neurociência, ocorre através de mudanças estruturais nas conexões sinápticas entre os neurônios. Estas alterações podem ser realizadas de variadas formas, o que acaba por gerar inúmeros tipos de aprendizagem. Não existe ainda uma teoria unificada ou uma abordagem comumente aceita na comunidade científica para a aprendizagem robótica. Dessa forma, não há consenso de qual o melhor paradigma de aprendizagem (PFEIFER; IIDA; BONGARD, 2005). Apesar disto, pode-se enumerar uma série de características desejáveis e necessárias em um algoritmo de aprendizagem para agentes de software autônomos: robustez e tolerância a ruídos, convergência rápida, tratabilidade computacional, adaptatividade a eventuais mudanças ambientais, dependência de informações que possam apenas serem extraídas de sensores do agente, e não àquelas fornecidas por um projetista ou observador externo (PFEIFER; SCHEIER, 1999).

Com base nestes princípios básicos de um agente de software, várias pesquisas na área de robótica móvel autônoma vem sendo desenvolvidas nos últimos tempos pelo Grupo de Pesquisas em Veículos Autônomos (GPVA[3]) da Unisinos. Destaca-se particularmente o desenvolvimento do SEVA3D, descrito na próxima seção.

2.8 Trabalhos anteriores

Um dos trabalhos anteriores mais importantes, que forneceu as bases para o desenvolvimento desta dissertação, foi o SEVA3D (Simulador de Estacionamento de Veículos Autônomos em um ambiente tridimensional), que é um simulador que realiza o estacionamento de um veículo não-holonômico (tipo carro) simulado em uma vaga paralela de forma autônoma. Ele utiliza um modelo tridimensional do ambiente e sensores do tipo sonar, simulados através da técnica de *raycast*. O controle do veículo é realizado através de duas formas: (i) um autômato finito baseado em regras codificadas manualmente (SEVA3D-A) (HEINEN; OSÓRIO; HEINEN, 2005; HEINEN et al., 2006d, 2007); (ii) uma Rede Neural Artificial (ANN) inspirada no modelo de Jordan (1986) (SEVA3D-N) (HEINEN et al., 2006a, 2006b, 2006c).

A vantagem de utilizar a segunda abordagem (SEVA3D-N) é que as regras não precisam ser codificadas manualmente, pois a ANN pode aprender a partir do exemplo de uma manobra de estacionamento. Para o treinamento da rede, adaptou-se o SEVA3D-A de modo a gerar um "arquivo de log", contendo o registro do estado dos sensores, estado do autômato e comandos enviados aos atuadores (velocidade e rotação), gerados pelo SEVA3D-A. Este "arquivo de log" foi posteriormente utilizado no SNNS[4] (*Stuttgart Neural Network Simulator*) para o aprendizado supervisionado da ANN. A Figura 2.4 mostra o exemplo de uma manobra de estacionamento realizada pelo SEVA3D-N. O veículo modelado para realizar o estacionamento (Figura 2.5) é uma reprodução de um veículo real do tipo Mini-Baja Buggy, desenvolvido pelo Grupo de Pesquisas em Veículos Autônomos (GPVA) da Unisinos (KELBER et al., 2005).

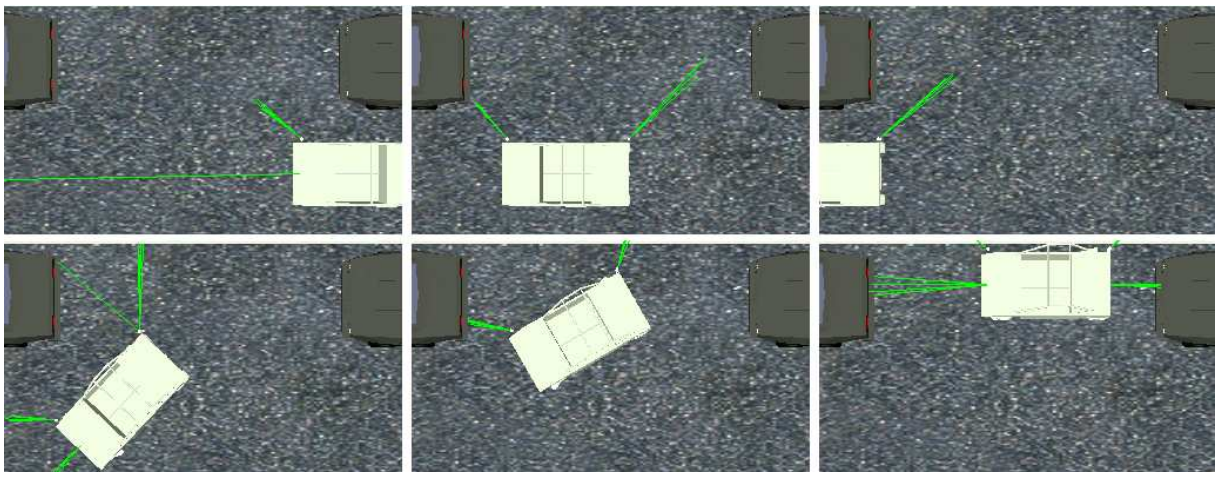


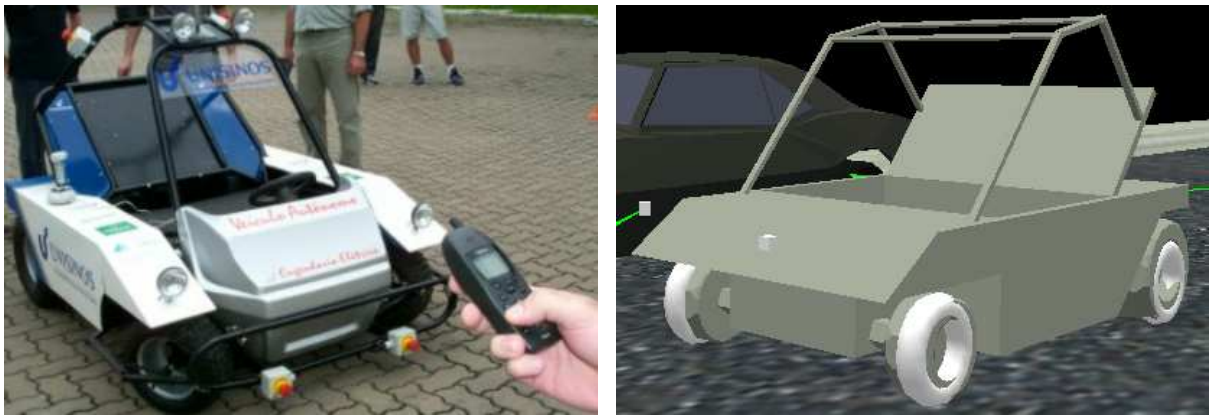
Figura 2.4: Exemplo de uma manobra de estacionamento

O SEVA3D segue vários dos princípios básicos de um agente de software, descritos na Seção 2.7, dentre os quais pode-se destacar:

[3]GPVA – http://www.exatec.unisinos.br/~autonom/

[4]SNNS – http://www-ra.informatik.uni-tuebingen.de/SNNS/

- Atua em determinado nicho ecológico (vias urbanas) e possui uma tarefa a ser realizada (estacionamento em vagas paralelas);
- É autônomo, auto-suficiente, localizado (adquire informações sobre o ambiente somente através dos sensores), e corporificado (simulado de forma realística);
- Aprende a tarefa em questão de forma supervisionada.



(a) Veículo real

(b) Veículo simulado

Figura 2.5: Veículo utilizado pelo SEVA3D

O SEVA3D permitiu que fossem criadas as bases para o desenvolvimento de aplicações na área de robótica móvel autônoma, e destacou a necessidade de se utilizar simulação baseada em física e sensores simulados realisticamente. Além disto, foi demonstrada a superioridade do modelo de controle baseado em uma ANN em relação ao autômato baseado em regras.

2.9 Considerações finais

Neste capítulo foram descritos diversos conceitos relativos às áreas de robótica móvel e agentes autônomos. Utilizando estes conceitos, é possível desenvolver um ambiente de simulação realístico para o estudo de sistemas de controle de robôs com pernas. Mas para a configuração automática do caminhar seja possível, é necessário o uso de técnicas de Aprendizado de Máquina (ML), como as descritas no próximo capítulo.

3 Aprendizado de Máquina

A Inteligência Artificial (*Artificial Intelligence* – AI) é uma área de estudos da computação que se interessa pelo estudo e criação de sistemas que possam exibir um comportamento inteligente e realizar tarefas complexas com um nível de competência que é equivalente ou superior ao de um especialista humano (NIKOLOPOULOS, 1997). As primeiras ferramentas de AI desenvolvidas adquiriam os conhecimentos que eram explicitados pelos especialistas de uma determinada área, o que era similar a programar um sistema computacional para resolver um problema. Posteriormente, mecanismos de aquisição automática de conhecimentos foram acrescentados a estas ferramentas, de onde surgiram a linguagem de programação Progol e os sistemas especialistas de 2ª geração. Estes mecanismos de aquisição automática de conhecimentos são conhecidos como técnicas de Aprendizado de Máquina (*Machine Learning* – ML) (MITCHELL, 1997). As técnicas de ML mais conhecidas são:

- Aprendizado por analogia ou por instâncias: sistemas baseados em casos – CBR (*Case Based Reasoning*) (MITCHELL, 1997; KOLODNER, 1993);
- Aprendizado por indução: árvores de decisão – ID3, C4.5, CN2 (IDT – *Induction of Decision Trees*) (QUINLAN, 1993), e ILP – *Inductive Logic Programming* (Progol) (NILSSON, 1998; REZENDE, 2003);
- Aprendizado por evolução/seleção: Algoritmos Genéticos (*Genetic Algorithms* – GA) e Programação Genética *Genetic Programming* – GP) (HOLLAND, 1975; GOLDBERG, 1989; MITCHELL, 1996);
- Aprendizado conexionista: Redes Neurais Artificiais (*Artificial Neural Networks* – ANN) (RUMELHART; HINTON; WILLIAMS, 1986; HAYKIN, 2001; BRAGA; LUDERMIR; CARVALHO, 2000), redes de função de base radial (*Radial-Basis Function Networks* – RBF) (HAYKIN, 2001; BRAGA; LUDERMIR; CARVALHO, 2000) e *Support Vector Machines* – SVM (HAYKIN, 2001; VAPNIK, 1995, 1998);
- Aprendizado por Reforço (*Reinforcement Learning* – RL) (SUTTON; BARTO, 1998; MITCHELL, 1997; HAYKIN, 2001);
- Outros tipos de aprendizado: mapas auto-organizáveis (*Self-Organizing Maps* – SOM) (KOHONEN, 1987), bayesiano e por explicações (*explanation based learning*) (MITCHELL, 1997; NILSSON, 1998);
- Técnicas de otimização: o método de Powell (*Powell's direction set*) (POWELL, 1964; BRENT, 1973; ACTON, 1970), o método do gradiente conjugado e métodos de programação linear (PRESS et al., 1992).

Neste capítulo são descritas as técnicas de Aprendizado de Máquina mais utilizadas em robótica autônoma. A Seção 3.1 descreve os Algoritmos Genéticos, a Seção 3.2 descreve as Redes Neurais Artificiais, a Seção 3.3 descreve a utilização das ANNs em conjunto com os GAs, e por último a Seção 3.4 descreve o método de Powell.

3.1 Algoritmos Genéticos

Os Algoritmos Genéticos (*Genetic Algorithms* – GA) são métodos de busca estocástica inspirados na Teoria da Evolução Natural das Espécies de Darwin (1859). A primeira tentativa de representação da teoria de Darwin por meio de um modelo matemático surgiu com o livro *The Genetic Theory of Natural Selection* (FISHER, 1958). Nos anos 60 e 70, John Holland dedicou-se ao estudo de processos naturais adaptáveis, que o levaram a inventar os Algoritmos Genéticos juntamente com os seus alunos e colegas da Universidade de Michigan. Os resultados destes estudos foram publicados no livro *Adaptation in Natural and Artificial Systems* (HOLLAND, 1975), hoje considerado um dos mais importantes livros de Algoritmos Genéticos.

Os Algoritmos Genéticos utilizam procedimentos iterativos que simulam o processo de evolução de uma população de possíveis soluções de um determinado problema. O processo de evolução é aleatório, porém guiado por um mecanismo de seleção baseado na adaptação de estruturas individuais. A cada iteração do algoritmo (uma geração), um novo conjunto de estruturas é criado através da troca de informações (*bits* ou blocos) entre estruturas bem adaptadas selecionadas da geração anterior (GOLDBERG, 1989). Novas estruturas também são geradas aleatoriamente com uma dada probabilidade e incluídas na população. O resultado tende a ser um aumento da adaptação de indivíduos ao meio, podendo acarretar também em um aumento global da aptidão da população a cada nova geração. Neste caso, a população evolui a cada geração se aproximando de uma solução ótima (GOLDBERG, 1989).

3.1.1 Algoritmo Genético básico

Um Algoritmo Genético é estruturado de forma que as informações referentes a um determinado sistema possam ser codificadas de maneira análoga aos cromossomos biológicos. Desta forma o algoritmo proposto assimila-se muito ao processo evolutivo natural. Um GA básico envolve seis passos: codificação das variáveis, criação da população inicial, avaliação da resposta (*fitness*), cruzamento (*crossover*), mutação e seleção dos mais aptos.

O Algoritmo 1 mostra o pseudo-código de um Algoritmo Genético básico. Neste algoritmo pode ser visto que os Algoritmos Genéticos começam com uma população de n estruturas aleatórias (indivíduos), onde cada estrutura codifica uma solução do problema. O desempenho de cada indivíduo é avaliado com base em uma função de avaliação de aptidão. Os melhores ten-

Algoritmo 1 *Algoritmo Genético básico*

```
geração ← 0
população ← InicializarPopulação(n)
enquanto geração < MAX_GERAÇÃO faça
    AvaliarFitness(população)
    MostrarEstatísticas(população, geração)
    Seleção(população)
    Cruzamento(população)
    Mutação(população)
    geração ← geração + 1
fim enquanto
SalvarMelhorIndivíduo(população)
```

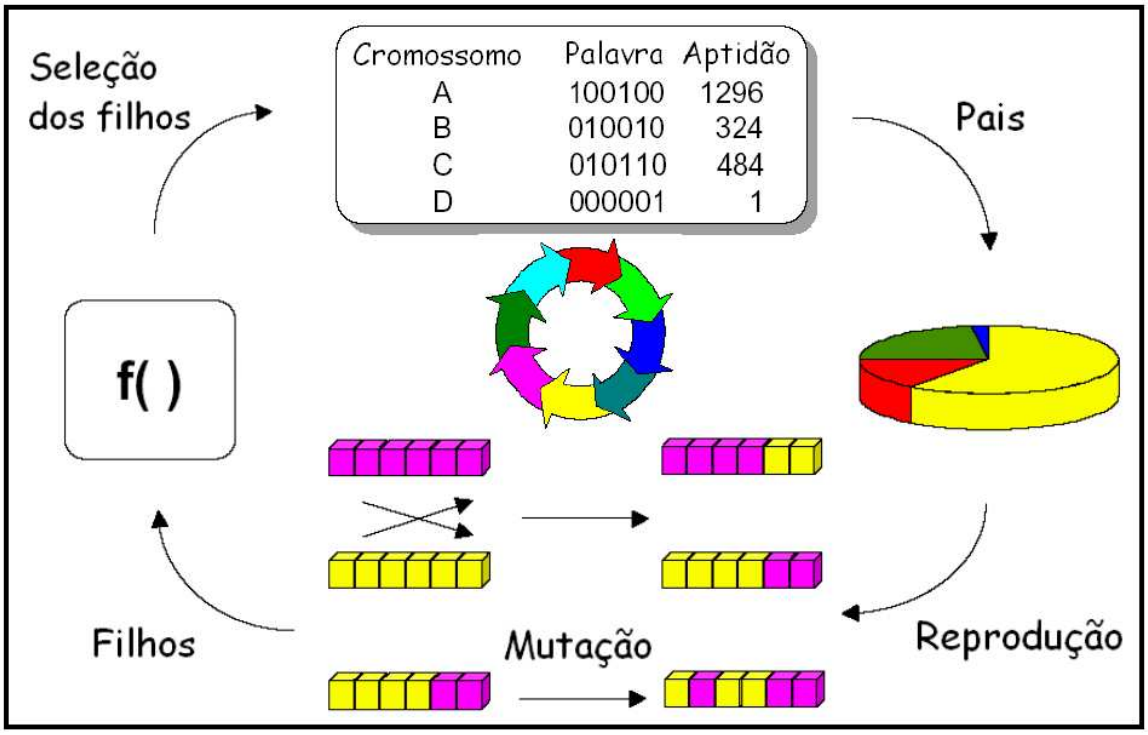


Figura 3.1: Ciclo dos Algoritmos Genéticos

derão a ser os progenitores da geração seguinte, possibilitando assim que as suas características sejam transmitidas às próximas gerações (PALAZZO, 1997).

A Figura 3.1 mostra o ciclo de funcionamento de um Algoritmo Genético. Na prática, um Algoritmo Genético pode ser implementado com o uso de *strings* de *bits* ou caracteres para representar os cromossomos, e com simples operações de manipulação é possível implementar os operadores genéticos. Na Figura 3.2, é mostrada a representação de um cromossomo composto por seis genes através de uma *string* de valores binários.

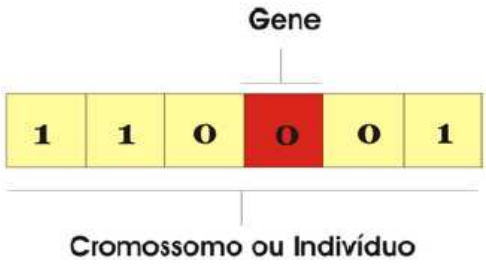


Figura 3.2: Exemplo de representação de um cromossomo de genes binários

3.1.2 Operadores genéticos

Os operadores genéticos são rotinas que transformam a população através de sucessivas gerações, estendendo a busca até se chegar a um resultado satisfatório (GOLDBERG, 1989). Um Algoritmo Genético padrão evolui em sucessivas gerações através do uso de três operadores básicos: seleção, cruzamento e mutação (SHAPIRO, 1999).

Seleção

A idéia principal do operador de seleção em um Algoritmo Genético é oferecer aos melhores indivíduos da população corrente a preferência no processo de reprodução, permitindo que estes indivíduos passem suas características às próximas gerações. Isto ocorre de forma similar à Evolução Natural, onde os indivíduos altamente adaptados ao ambiente possuem mais oportunidades de se reproduzir do que os indivíduos considerados mais fracos e menos adaptados (HOLLAND, 1975).

Nos Algoritmos Genéticos, a seleção se dá através da aptidão (*fitness*) de cada indivíduo (Figura 3.3), que é um valor que mede o seu grau de adaptabilidade ao ambiente, ou seja, a qualidade da solução do problema representada por este indivíduo (GOLDBERG, 1989). Assim,

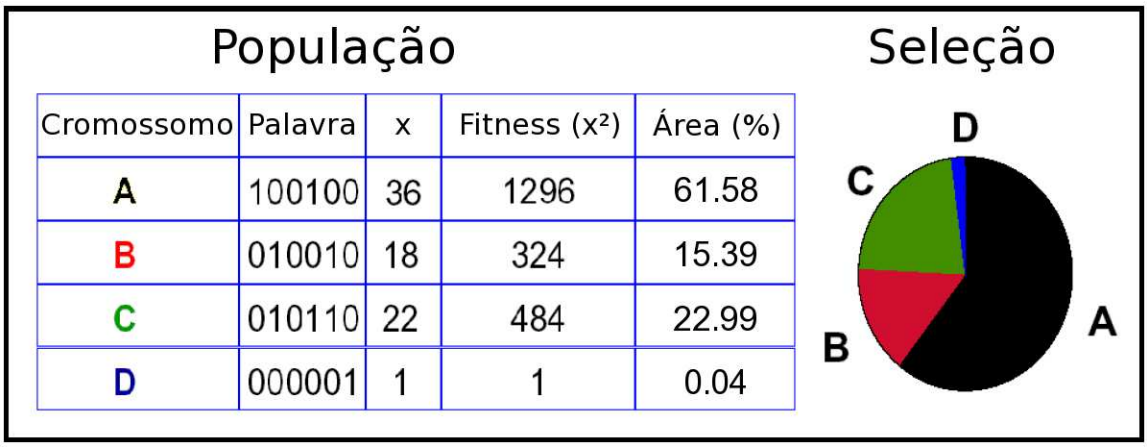


Figura 3.3: Esquema de seleção *roulette wheel*

através de um esquema de seleção específico, são selecionados os melhores indivíduos para darem origem à próxima geração. Um dos esquemas de seleção mais utilizados é o *roulette wheel*, o qual simula uma roleta na qual cada indivíduo recebe uma área proporcional ao seu nível de aptidão, de forma que os indivíduos mais aptos tenham maiores chances de serem selecionados (GOLDBERG, 1989). Esta roleta virtual é "girada" várias vezes, de forma a selecionar os indivíduos que darão origem à próxima geração. A Figura 3.3 ilustra este esquema de seleção.

Cruzamento

Uma das principais características dos Algoritmos Genéticos que os distinguem das demais técnicas de busca aleatória é o operador cruzamento. O cruzamento (*crossover*) é a troca de segmentos entre pares de cromossomos selecionados, com a finalidade de originar novos indivíduos que irão compor a próxima geração. A idéia central do cruzamento é a propagação das características dos indivíduos mais aptos da população. As formas de reprodução mais comuns em Algoritmos Genéticos são o cruzamento em um ponto, o cruzamento em dois pontos e o cruzamento em múltiplos pontos ou uniforme (MITCHELL, 1996).

Na reprodução baseada no cruzamento em um único ponto *(one-point crossover)*, o ponto de quebra do cromossomo é escolhido de forma aleatória sobre o comprimento da *string* que o representa, e a partir desse ponto se realiza a troca de material genético entre os dois indivíduos. Na Figura 3.4 é mostrado um esquema da representação desse tipo de cruzamento.

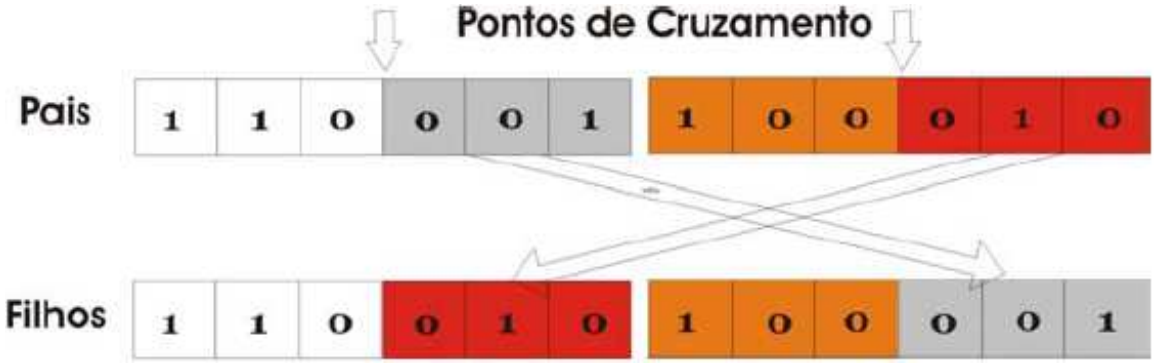


Figura 3.4: Esquema do cruzamento em um único ponto

Na reprodução baseada no cruzamento em dois pontos *(two-point crossover)*, procede-se de maneira similar ao cruzamento em um único ponto, mas a troca de segmentos é realizada a partir de dois pontos, como mostra a Figura 3.5.

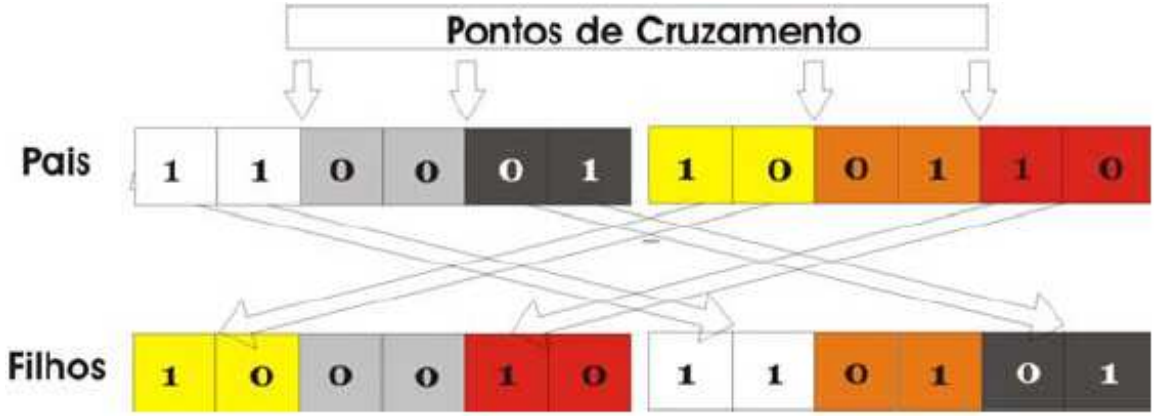


Figura 3.5: Esquema do cruzamento em dois pontos

Na reprodução baseada no cruzamento em múltiplos pontos ou uniforme *(uniform crossover)*, cada gene é criado através da cópia de um dos genes dos pais, escolhido de acordo com uma máscara de cruzamento gerada aleatoriamente. Onde houver 1 na máscara de cruzamento, o gene correspondente será copiado do primeiro pai e onde houver 0 na máscara, o gene copiado será o do segundo pai. O processo é repetido invertendo-se os pais para produzir o segundo descendente. Cada par de indivíduos é criado com uma máscara de cruzamento específica gerada aleatoriamente. A Figura 3.6 mostra de forma visual um esquema do cruzamento uniforme.

A taxa de cruzamento Pc define a probabilidade de ocorrer um cruzamento, sendo que os valores mais utilizados ficam entre $0,5 \leq Pc \leq 0,9$. Se não ocorrer o cruzamento entre um par de indivíduos, os dois filhos gerados serão cópias exatas dos pais.

Mutação

A mutação é vista como o operador responsável pela introdução e manutenção da diversidade genética na população (GOLDBERG, 1989). Ela trabalha alterando arbitrariamente, logo após o cruzamento, um ou mais componentes de uma estrutura escolhida entre os novos indivíduos, fornecendo dessa forma meios para a introdução de material genético novo na

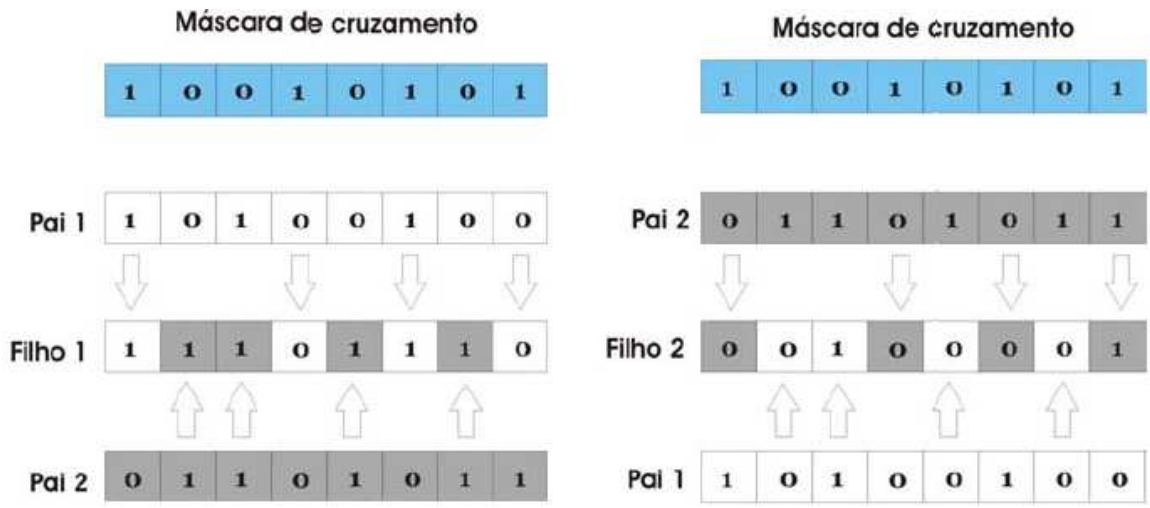


Figura 3.6: Esquema do cruzamento uniforme

população (HOLLAND, 1975). O operador de mutação é aplicado aos indivíduos com uma probabilidade dada por uma taxa de mutação *Pm*. A Figura 3.7 ilustra o processo de mutação.

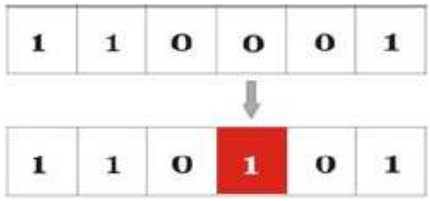


Figura 3.7: Esquema da ocorrência de mutação

A ocorrência de mutação em determinado gene é determinada pela taxa de mutação *Pm*, que usualmente possui um valor pequeno, sendo que os valores mais utilizados ficam entre $0,001 \leq Pm \leq 0,1$. Segundo Back (1996), a performance do GA em termos de convergência tende a decair em populações de tamanho relativamente grande $(n > 200)$ usando grande probabilidade de mutação $(Pm > 0,05)$ e em populações de pequeno tamanho $(n < 200)$ combinadas com pequena probabilidade de mutação $(Pm < 0,02)$.

Elitismo

O elitismo (DE JONG, 1975) é um operador genético que visa impedir que os melhores indivíduos de uma população sejam perdidos devido às operações de cruzamento e mutação. O elitismo opera simplesmente passando os melhores indivíduos da geração atual para a geração seguinte, sem qualquer alteração genética.

3.1.3 Parâmetros genéticos

Os Algoritmos Genéticos possuem diversos parâmetros que influenciam o comportamento dos mesmos e do processo de evolução. Para que se consiga obter uma boa performance, é importante analisar a maneira com que cada um dos parâmetros influencia o comportamento

dos Algoritmos Genéticos, de forma que seja possível configurá-los conforme às necessidades do problema e/ou dos recursos disponíveis. A seguir serão analisados os parâmetros genéticos mais utilizados na prática.

Tamanho da população

O tamanho da população determina o número de cromossomos na população, afetando diretamente o desempenho global e a eficiência dos GAs. Com uma população pequena, o desempenho tende a cair, pois a população fornece uma pequena cobertura do espaço de estados. Uma população grande geralmente fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras (tendência da população evoluir para uma solução sub-ótima). No entanto, o uso de grandes populações exige um maior tempo de processamento.

Taxa de cruzamento

Quanto maior for a taxa de cruzamento, mais rapidamente novas estruturas serão introduzidas população. Mas se a taxa de cruzamento for muito alta, a maior parte da população será substituída, o que pode fazer com que os indivíduos mais aptos sejam perdidos. Com uma taxa de cruzamento muito baixa, a evolução torna-se muito lenta.

Tipo de cruzamento

O tipo de cruzamento a ser utilizado determina a forma como se procederá a troca dos segmentos de informação entre os pares de cromossomos selecionados para cruzamento. O cruzamento do tipo uniforme tende a ser bastante disruptivo, quebrando os blocos de construção de hipóteses (*building block hypothesis*) (MITCHELL, 1996). Já o uso de cruzamento em um único ponto pode tornar o processo de evolução mais lento. O ideal é testar os diversos tipos de cruzamento para o problema em questão e assim verificar qual apresenta um melhor resultado.

Taxa de mutação

A taxa de mutação determina a probabilidade com que uma mutação ocorrerá. A mutação é utilizada para introduzir novas informações na população e também para prevenir que ocorra a convergência prematura. Uma taxa de mutação pequena previne que dada posição fique estagnada em um valor (mínimos locais), e ao mesmo tempo evita que ocorra uma grande variação de uma geração para outra. Com uma taxa de mutação muito alta a busca se torna essencialmente aleatória e aumenta muito a possibilidade de que uma boa solução seja destruída.

3.1.4 Biblioteca GAlib

Atualmente existem várias bibliotecas de software que facilitam a implementação dos Algoritmos Genéticos, disponibilizando diversas rotinas e estruturas de dados. Uma das bibliotecas mais utilizadas é a GAlib, que foi desenvolvida por Matthew Wall do *Massachusetts Institute*

of Technology (MIT). A biblioteca GAlib[1] é uma das mais completas, eficientes conhecidas bibliotecas de software para a simulação de Algoritmos Genéticos, e possui a vantagem de ser uma biblioteca gratuita (*freeware*) e de código aberto (*open source*), baseada na linguagem de programação C++.

3.2 Redes Neurais Artificiais

As Redes Neurais Artificiais (*Artificial Neural Networks* – ANN) surgiram como uma tentativa de se reproduzir o funcionamento do cérebro humano, e assim desenvolver máquinas capazes de realizar muitas funções que antes só eram possíveis de serem realizadas através da intervenção humana. As Redes Neurais possuem a capacidade de aprendizado e de generalização, de forma que a partir dos exemplos analisados elas conseguem extrair as regras gerais que descrevem um problema e assim podem aplicar estas regras para a solução de novos exemplos não analisados anteriormente. Outras características desejáveis que as Redes Neurais possuem são o fato de serem tolerantes a dados incorretos e/ou incompletos e também de poderem lidar com informações quantitativas e qualitativas (HEINEN; OSÓRIO, 2006e).

As Redes Neurais foram originalmente desenvolvidas baseadas nos estudos realizados sobre a forma como o conhecimento é armazenado no cérebro humano e a forma como ocorre o aprendizado (MCCULLOCH; PITTS, 1943). Nestes estudos foi constatado que nos seres humanos o conhecimento é armazenado nas ligações que os neurônios realizam uns com os outros, chamadas de sinapses, e a medida que o aprendizado ocorre mais ligações vão se formando e se fortalecendo entre estes neurônios.

Baseado no funcionamento do neurônio biológico, foi desenvolvido o neurônio artificial, que é uma simplificação matemática que tenta reproduzir as principais características dos neurônios humanos referentes a forma em que se processa a aquisição de conhecimentos (ROSENBLATT, 1959). A Figura 3.8 mostra o esquema de um neurônio artificial.

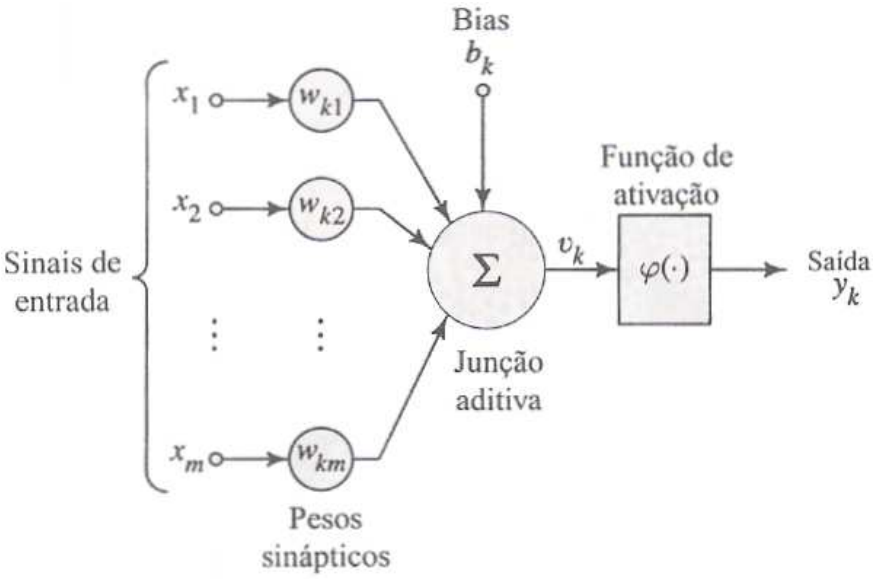


Figura 3.8: Esquema de um neurônio artificial (HAYKIN, 2001)

[1] GAlib – http://www.lancet.mit.edu/ga/

Um neurônio artificial é constituído de entradas $(x_1, x_2, \ldots, x_m)$, que recebem os sinais provenientes do exterior, uma função de ativação $(\varphi(\cdot))$, que realiza uma combinação dos valores obtidos nas entradas, e saídas, que transmitem os sinais recebidos e processados pelo neurônio para o exterior (BRAGA; LUDERMIR; CARVALHO, 2000). Associado a cada entrada de um neurônio artificial existem pesos $(w_{k1}, w_{k2}, \ldots, w_{km})$, que são valores numéricos que representam a força das conexões existentes entre os diversos neurônios. Se um peso tiver um valor elevado, a sua respectiva entrada será amplificada, e se ele tiver um valor próximo de 0, a sua respectiva entrada terá seus valores atenuados.

O tipo de neurônio artificial mais utilizado é o Perceptron, que foi originalmente desenvolvido por Rosenblatt (1959). Em termos matemáticos, a saída y_k de um Perceptron k com m entradas é descrita pela equação:

$$y_k = \varphi(v_k) \,, \tag{3.1}$$

onde $\varphi(\cdot)$ é a função de ativação, que serve para normalizar o valor de saída do neurônio, e v_k é o campo local induzido, calculado pela fórmula:

$$v_k = \sum_{i=1}^{m} w_{ki} x_i + b_k \,, \tag{3.2}$$

onde $x_1, x_2, \ldots, x_m$ são os valores de entrada, $w_{k1}, w_{k2}, \ldots, w_{km}$ são os pesos sinápticos do neurônio k e b_k é o *bias*, que é uma entrada especial que recebe sempre o valor 1 e que possui um peso a ela associado que pode ser ajustado, assim como os demais pesos do neurônio.

A função de ativação $\varphi(\cdot)$ mais utilizada é a *sigmoid*, definida através da função logística:

$$\varphi(v_k) = \frac{1}{1 + e^{-av_k}} \,, \tag{3.3}$$

onde a é o parâmetro de inclinação (*slope*) da função. A Figura 3.9 mostra o gráfico da função logística para $a = 1$. A função *sigmoid* faz com que a saída de um neurônio seja normalizada entre 0 e 1, e devido a sua curvatura especial ela é usada no processo de aprendizado para gerar uma certa estabilidade no sistema, evitando que os pesos variem depois que o valor da saída já se encontre próximo a um dos extremos (RUMELHART; HINTON; WILLIAMS, 1986).

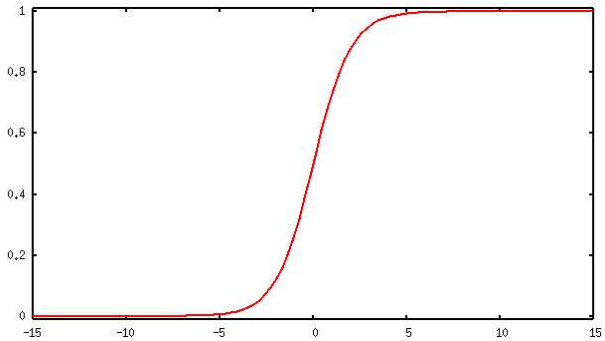


Figura 3.9: Gráfico da função de transferência *sigmoid*

Para que o aprendizado seja possível em uma Rede Neural à base de *perceptrons*, é necessário que seja fornecido um conjunto de exemplos de treinamento com as saídas desejadas para cada exemplo. Este tipo de aprendizado, que é conhecido como aprendizado supervisionado, ocorre em várias épocas, onde a cada época o conjunto inteiro de dados de treinamento

é submetido à Rede Neural para o ajuste dos pesos (OSÓRIO; BITTENCOURT, 2000). Inicialmente os pesos são inicializados aleatoriamente, e a cada época n os pesos do neurônio k são adaptados através da fórmula:

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) \ , \tag{3.4}$$

onde $\Delta w_{ki}(n)$ é a correção do peso $w_{ki}(n)$ do neurônio k, calculada pela *regra delta*:

$$\Delta w_{ki}(n) = \eta \, \delta_k(n) \, x_i(n) \ , \tag{3.5}$$

onde η é a taxa de aprendizado (um valor entre 0 e 1 que determina a velocidade de aprendizado), $x_i(n)$ é o valor da entrada i e $\delta_k(n)$ é o gradiente local, calculado pela fórmula:

$$\delta_k(n) = e_k(n) \, \varphi'(v_k(n)) \ , \tag{3.6}$$

onde $v_k(n)$ é o campo local induzido, calculado pela Fórmula 3.2, $\varphi'(\cdot)$ é a derivada da função de ativação em relação ao argumento, e $e_k(n)$ é o sinal de erro, calculado através da fórmula:

$$e_k(n) = d_k(n) - y_k(n) \ , \tag{3.7}$$

onde $d_k(n)$ é a saída desejada (valor esperado) e $y_k(n)$ é a saída obtida no neurônio, calculada pela da Fórmula 3.1. Para a função *sigmoid* (Fórmula 3.3), a derivada da função de ativação $\varphi'(\cdot)$ em relação à $v_k(n)$ é obtida através da fórmula:

$$\varphi'(v_k(n)) = a y_k(n)[1 - y_k(n)] \ , \tag{3.8}$$

onde a é o parâmetro de inclinação (*slope*) da *sigmoid*.

A medida que o aprendizado se processa, o erro quadrático na saída $[e_k(n)]^2$ vai sendo reduzido, até que atinja um valor mínimo, onde este mínimo pode ser um mínimo local ou o mínimo global. O ideal seria conseguir atingir o mínimo global, mas dependendo da inicialização dos pesos, muitas vezes o aprendizado fica preso em um mínimo local. A Figura 3.10 demonstra esta situação para um neurônio com uma único peso. Em um neurônio com m entradas, a curva de descida do gradiente é um hiperplano de $m+1$ dimensões.

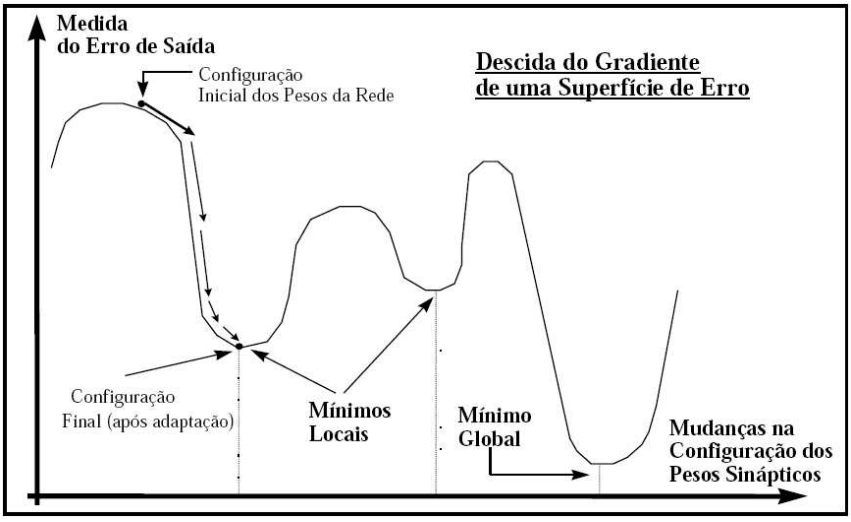


Figura 3.10: Descida do gradiente de uma superfície de erro

Uma Rede Neural constituída de apenas um neurônio, como por exemplo o Perceptron de m entradas descrito acima, só consegue solucionar problemas que sejam linearmente separáveis (HAYKIN, 2001). Para que a solução de problemas não linearmente separáveis seja possível, são necessários vários *perceptrons* combinados em diversas camadas, formando uma Rede Neural do tipo *Multi Layer Perceptron* (MLP).

Em uma rede MLP, os valores desejados $d_k(n)$ só estão disponíveis para os neurônios da camada de saída, de forma que para as demais camadas o erro $e_k(n)$ precisa ser estimado de alguma forma. O algoritmo mais utilizado para ajustar os pesos de uma Rede Neural do tipo MLP é o *back-propagation* (RUMELHART; HINTON; WILLIAMS, 1986), descrito abaixo.

3.2.1 Back-propagation

Um dos algoritmos mais utilizados para o treinamento de Redes Neurais do tipo MLP é o *back-propagation*, proposto inicialmente em Rumelhart, Hinton e Williams (1986) e que torna possível o treinamento de Redes Neurais de múltiplas camadas, e assim ser possível solucionar problemas não linearmente separáveis. A Figura 3.11 mostra o esquema de uma rede MLP.

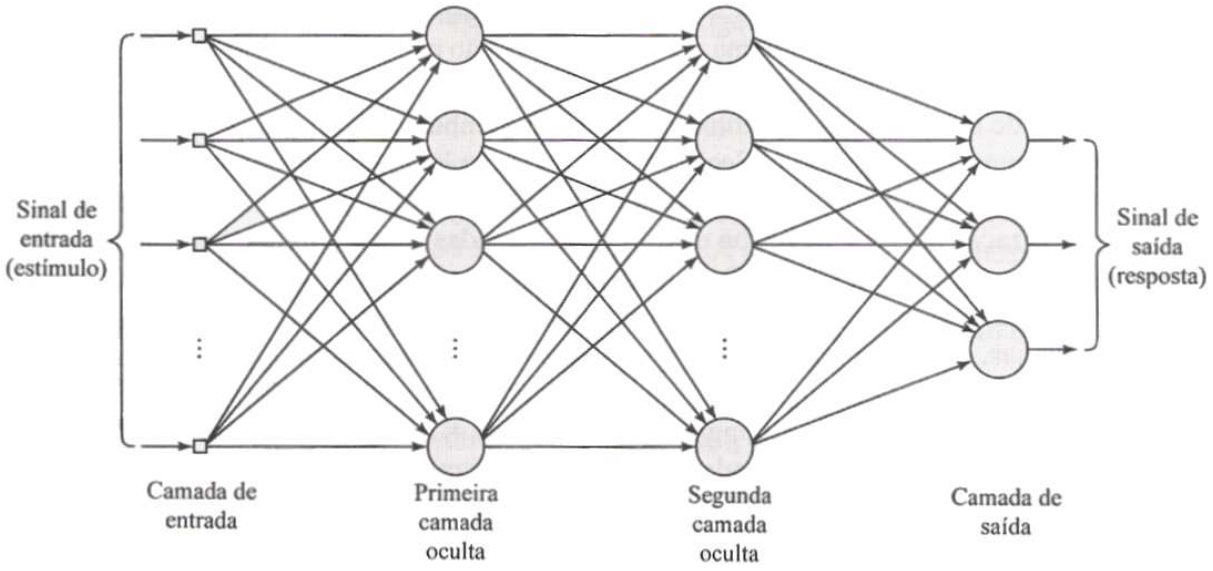


Figura 3.11: Esquema de uma Rede Neural do tipo MLP (HAYKIN, 2001)

A primeira camada, conhecida como camada de entrada, é apenas um *buffer* para os dados de entrada, pois não possui pesos sinápticos nem função de ativação, e as demais camadas da Rede Neural são formadas por neurônios do tipo Perceptron. As conexões entre as diversas camadas ocorrem por meio de pesos sinápticos, sendo que as entradas dos neurônios da camada l são as saídas dos neurônios da camada anterior $l-1$.

O funcionamento das redes MLP com o algoritmo *back-propagation* ocorre da seguinte forma: Inicialmente, todos os pesos sinápticos da rede são inicializados aleatoriamente segundo uma distribuição uniforme. Em seguida, os exemplos de treinamento são apresentados repetidamente à Rede Neural através do ciclo $\{(\mathbf{x}(n),\mathbf{d}(n))\}_{n=1}^{N}$, onde N é o número de épocas de treinamento. Para cada exemplo de treinamento $(\mathbf{x}(n),\mathbf{d}(n))$ são realizados os passos *forward* e *backward*, descritos abaixo. Uma descrição mais detalhada do algoritmo *back-propagation* pode ser encontrada em Haykin (2001).

Passo *Forward*: Considere um exemplo de treinamento $(\mathbf{x}(n), \mathbf{d}(n))$ sendo aplicado à Rede Neural na época n, onde $\mathbf{x}(n)$ é o vetor de entradas aplicado à camada de entrada e $\mathbf{d}(n)$ é o vetor de saídas desejadas apresentado à camada de saída para o cálculo do erro e_j. Para todos os neurônios da Rede Neural, compute o campo local induzido e o valor de saída dos neurônios, camada por camada. O campo local induzido $v_j^{(l)}(n)$ do neurônio j na camada l é calculado através da fórmula:

$$v_j^{(l)}(n) = \sum_{i=0}^{m_0} w_{ji}^{(l)}(n)\, y_i^{(l-1)}(n) \;, \tag{3.9}$$

onde $y_i^{(l-1)}(n)$ é o sinal de saída do neurônio i na camada anterior $l-1$ na época n e $w_{ji}^{(l)}(n)$ é o peso sináptico do neurônio j na camada l que alimenta este neurônio com a saída do neurônio i na camada $l-1$. Para $i=0$, temos $y_0^{(l-1)}(n) = +1$ e $w_{j0}^{(l)}(n) = b_j^{(l)}$ sendo o *bias* aplicado ao neurônio j na camada l. Assumindo o uso da função *sigmoid* para a ativação, o sinal de saída do neurônio j na camada l é obtido através da fórmula:

$$y_j^{(l)} = \varphi_j(v_j(n)) \;. \tag{3.10}$$

Se o neurônio está na primeira camada oculta (ou seja, $l=1$), então atribuímos:

$$y_j^{(0)}(n) = x_j(n) \;, \tag{3.11}$$

onde $x_j(n)$ é o j-ésimo elemento do vetor de entrada $\mathbf{x}(n)$. Se o neurônio j está na camada de saída, (ou seja, $l=L$, onde L é o número de camadas da Rede Neural) então atribuímos:

$$y_j^{(L)} = o_j(n) \;, \tag{3.12}$$

onde $o_j(n)$ é o sinal obtido na saída do neurônio j. Em seguida é calculado o erro nas saídas da Rede Neural através da fórmula:

$$e_j(n) = d_j(n) - o_j(n) \;, \tag{3.13}$$

onde $d_j(n)$ é o j-ésimo elemento do vetor de saídas desejadas $\mathbf{d}(n)$.

Passo *backward*: Calcule os gradientes locais $\delta_j^{(l)}(n)$ da ANN, definidos através das equações:

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\, \varphi_j'(v_j^{(L)}(n)) & \text{para o neurônio } j \text{ na camada de saída } L \\ \varphi_j'(v_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n)\, w_{kj}^{(l+1)}(n) & \text{para o neurônio } j \text{ na camada oculta } l \end{cases} \tag{3.14}$$

onde o apóstrofe em $\varphi_j'(\cdot)$ indica diferenciação em relação ao argumento. Ajuste os pesos sinápticos da rede na camada l de acordo com a regra delta generalizada:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[w_{ji}^{(l)}(n-1)] + \eta\, \delta_j^{(l)}(n)\, y_i^{(l-1)}(n) \;, \tag{3.15}$$

onde η é a taxa de aprendizado e α é a constante de *momentum*, que é um parâmetro que regula a taxa de inércia do aprendizado, que é utilizada para filtrar as variações de alta freqüência na superfície de erro (RUMELHART; HINTON; WILLIAMS, 1986).

Os passos *forward* e *backward* são repetidos iterativamente para cada exemplo de treinamento por *N* épocas, até que o aprendizado seja concluído. O número de neurônios na camada oculta, assim como o número de camadas ocultas, depende do problema que está sendo abordado, de forma que não existe um valor padrão que sirva para a maioria dos casos.

O algoritmo *back-propagation* com *momentum*, descrito acima, costuma ser muito sensível aos valores da taxa de aprendizado η e da constante de *momentum* α, de forma que se estes valores não estiverem configurados adequadamente o aprendizado não irá ocorrer de forma satisfatória. Na prática, são gastas muitas horas de um especialista humano até que se consiga configurar corretamente estes parâmetros e o número de neurônios da camada oculta, e estas configurações são específicas para o problema abordado, de forma que uma configuração que funciona de forma satisfatória para um problema pode não funcionar corretamente para outro (HEINEN; OSÓRIO, 2006e).

Outro fator importante em uma Rede Neural é o grau de generalização. Quando uma Rede Neural é treinada, os pesos vão sendo ajustados lentamente até que ela responda de forma adequada aos exemplos presentes na base de dados. Mas se o aprendizado prosseguir por um número excessivo de épocas, a partir de certo ponto a Rede Neural começará a aprender características que são específicas dos dados de treinamento e que não estão presentes nos dados populacionais. Este problema é conhecido como *overfitting*. Para tentar evitá-lo, costuma-se utilizar duas bases de dados, uma para o treinamento, com a qual o aprendizado é realizado e os pesos sinápticos são ajustados, e uma base de teste de generalização, com a qual a Rede Neural é ativada apenas no passo *forward*, sem que haja ajuste dos pesos. A medida que o aprendizado progride, o erro em ambas as bases de dados vai sendo reduzido, como mostra a Figura 3.12.

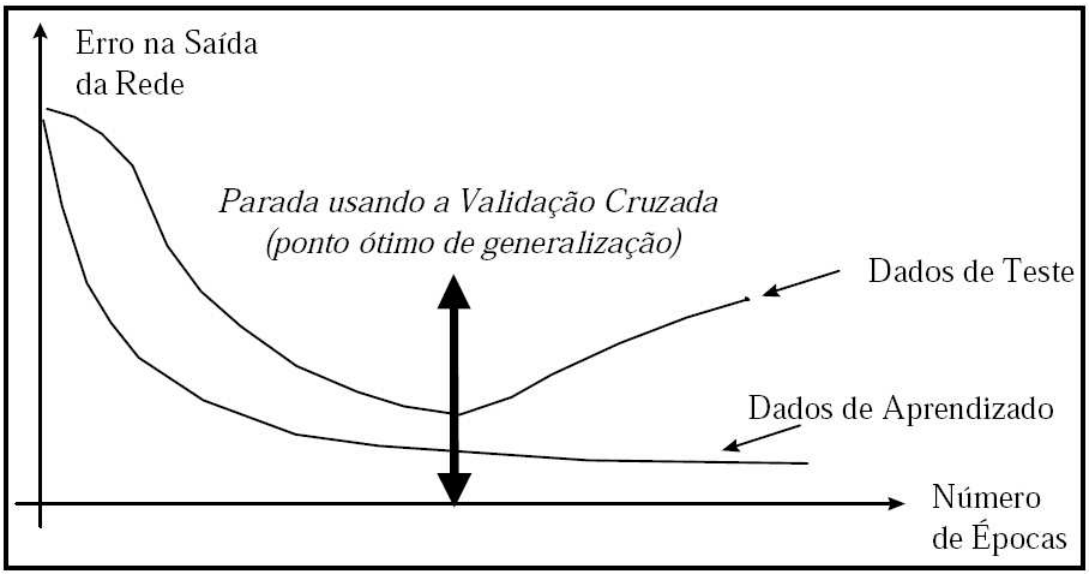


Figura 3.12: Curvas de erro no aprendizado e na generalização

Na base de dados de aprendizado o erro se reduz indefinidamente (podendo inclusive chegar a zero), mas para a base de teste de generalização, o erro diminui até certo ponto, e a partir deste ponto ele tende a começar a subir, indicando a ocorrência de *overfitting*. O ideal é realizar o aprendizado de forma que ele passe do ponto ótimo e que os pesos da melhor época em termos de generalização sejam salvos para uso posterior, pois assim se garante um melhor desempenho do sistema em termos de generalização (HAYKIN, 2001; HEINEN; OSÓRIO, 2005).

3.2.2 Redes Neurais recorrentes e fenômenos temporais

Muitos algoritmos de treinamento das ANNs não são capazes de implementar mapeamentos dinâmicos, como por exemplo o algoritmo *back-propagation* simples (BRAGA; LUDERMIR; CARVALHO, 2000). A principal questão é como estender a estrutura das redes MLP para que assumam um comportamento que varie com o tempo, sendo assim capazes de tratar fenômenos temporais. Segundo Elman (1990), para que uma Rede Neural possa lidar com séries temporais, é preciso que ela possua memória. Esta memória pode ser obtida de duas formas (BRAGA; LUDERMIR; CARVALHO, 2000):

1. Introduzir atraso no tempo, como nas técnicas TDNN *Time Delay Neural Network* (WAIBEL, 1989) e *FIR Multilayer Perceptron* (WAN, 1990);
2. Utilizar redes recorrentes, tais como o *back-propagation through time* (WERBOS, 1990), *real-time recurrent learning* (WILLIAMS; ZIPSER, 1989), redes de Elman (ELMAN, 1990) e redes de Jordan (JORDAN, 1986).

As técnicas de atraso do tempo utilizam elementos de atraso unitário, representados por z^{-1}, que fornecem entradas de iterações passadas à Rede Neural em questão. O atraso no fornecimento de entradas à Rede Neural introduz memória na rede, proporcionando aos neurônios valores de entrada atuais e valores temporalmente anteriores a eles.

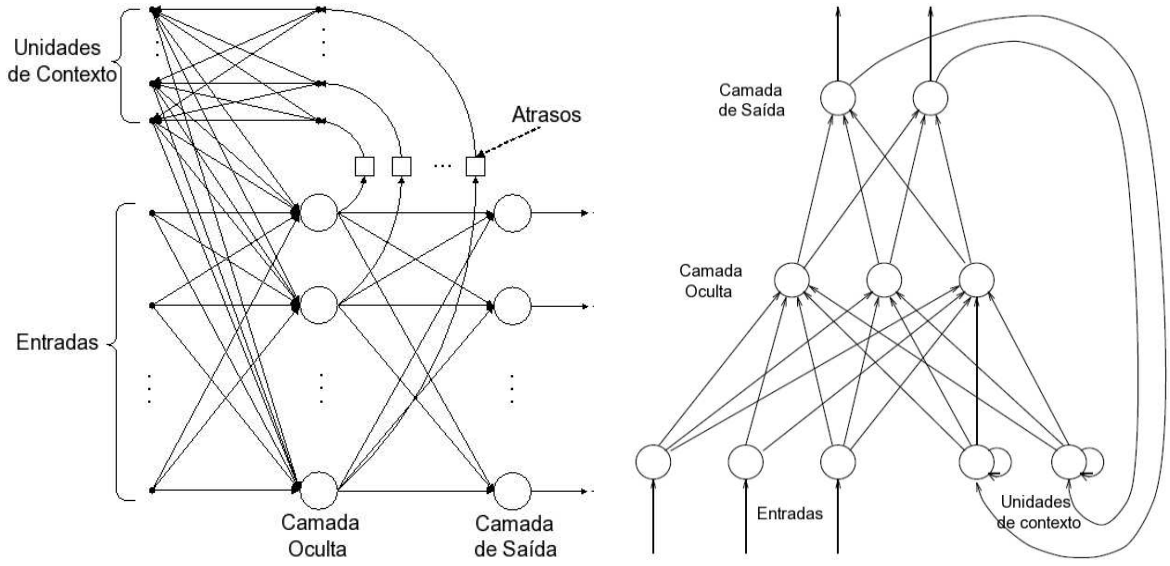
Já as Redes Neurais recorrentes diferem das redes *feedforward* pelo fato de apresentar pelo menos um laço de retro-alimentação (*feedback*). Estas conexões de retro-alimentação fornecem memória à Rede Neural, o que faz com que elas apresentem um comportamento dinâmico não-linear. As redes recorrentes são bastante úteis na predição de séries temporais (BRAGA; LUDERMIR; CARVALHO, 2000).

As redes de Elman (ELMAN, 1990), são Redes Neurais recorrentes onde a realimentação se dá na saída de cada neurônio da camada oculta para todos neurônios da mesma camada. Uma outra camada, chamada de camada de contexto, é utilizada para memorizar os valores de saída da camada oculta no instante anterior. O processamento da rede consiste nos eventos: No instante t (inicial) o sinal é propagado pela rede e as unidades de contexto, inicializadas com o valor 0, não influenciarão na saída da rede, ou seja, na primeira iteração a rede se comportará como uma rede *feedforward*. Ainda na primeira iteração os neurônios ocultos ativarão os neurônios da camada de contexto e esses armazenarão a saída desta iteração que será utilizada no próximo ciclo. O algoritmo *back-propagation* é então aplicado para a correção dos pesos sinápticos, com exceção as sinapses recorrentes, que possuem pesos fixados em 1. No instante de tempo $t+1$ o processo é repetido, mas a partir de agora os neurônios ocultos serão ativados pelas unidades de entrada e pelas unidades de contexto, que possuem o valor de saída dos neurônios ocultos no instante anterior (ELMAN, 1990). A Figura 3.13(a) mostra um esquema de uma rede Elman.

Nas redes de Elman, o valor de saída y de um neurônio k da camada oculta l no instante de tempo t é calculado através da equação:

$$y_k^{(l)}(t) = \varphi\left(\sum_{i=1}^{m} w_{a,ki}^{(l)} x_i^{(l-1)}(t) + \sum_{j=1}^{n} w_{b,kj}^{(l)} y_j^{(l)}(t-1) + b_k^{(l)}\right), \tag{3.16}$$

onde $\varphi(\cdot)$ é a função de ativação, m é o número de neurônios da camada $l-1$, n é o número de neurônios da camada l, $x_i^{(l-1)}(t)$ é o valor de saída do neurônio i da camada anterior $l-1$



(a) Rede Elman (b) Rede Jordan

Figura 3.13: Esquema das Redes Neurais recorrentes de Elman e Jordan

no instante de tempo t, $y_j^{(l)}(t-1)$ é o valor de saída do neurônio k da camada l no instante de tempo $t-1$, $b_k^{(l)}$ é o *bias* do neurônio k da camada l, $w_{a,ki}^{(l)}$ é o peso da conexão sináptica existente entre os neurônios k (da camada l) e i (da camada $l-1$), e $w_{b,kj}^{(l)}$ é o peso da conexão sináptica existente entre os neurônio k e a unidade de contexto j, que guarda o valor de saída do neurônio j da camada l no instante de tempo $t-1$.

Nas redes de Jordan (JORDAN, 1986), as saídas da Rede Neural são copiadas para as unidades de contexto. Além disso, as unidades de contexto são localmente recorrentes, como mostra a Figura 3.13(b). A grande diferença em termos de topologia entre as duas redes é que a recorrência na rede de Elman é feita da camada oculta para as entradas, enquanto na rede de Jordan a recorrência é feita das saídas para as entradas (BRAGA; LUDERMIR; CARVALHO, 2000). Os pesos que ligam as saídas com unidades de contexto são fixados em 1 não variam durante o aprendizado.

Em uma rede Jordan, o valor de saída v de um neurônio k da primeira camada oculta l no instante de tempo t é calculado através da equação:

$$v_k^{(l)}(t) = \varphi\left(\sum_{i=1}^{m} w_{a,ki}^{(l)} x_i(t) + \sum_{q=1}^{o} w_{b,kq}^{(l)} y_q^{(l)}(t-1) + b_k^{(l)}\right), \tag{3.17}$$

onde o é o número de neurônios da camada de saída, $x_i(t)$ é o valor da entrada i da Rede Neural no instante t, $y_q^{(l)}(t-1)$ é o valor da saída q da Rede Neural no instante $t-1$, $w_{a,ki}^{(l)}$ é o peso da conexão sináptica existente entre os neurônios k (da primeira camada oculta l) e i (da camada de entrada $l-1$), e $w_{b,kq}^{(l)}$ é o peso da conexão sináptica existente entre os neurônio k e a unidade de contexto q, que guarda o valor de saída q da Rede Neural no instante de tempo $t-1$.

Em suma, as arquiteturas de Elman e Jordan são caracterizadas por um conjunto extra de unidades de processamento, chamadas de unidades de contexto, nas quais a Rede Neural

armazena os valores de ativação da camada oculta ou da camada de saída. Estas conexões de *feedback* possuem possuem os pesos fixados em 1. Os valores armazenados nas unidades de contexto no instante de tempo t são utilizados como entradas no instante de tempo $t+1$, e permitem que a Rede Neural apresente um comportamento dinâmico (HAYKIN, 2001).

3.3 Sistemas híbridos – ANN e GA

A área de Aprendizado de Máquina possui diversas técnicas, e cada uma delas possui diversas vantagens e desvantagens, bem como pontos fracos e limitações. Assim, não existe uma técnica de Aprendizado de Máquina ideal que funcione de forma satisfatória para todas as classes de problemas existentes (MITCHELL, 1997), de forma que para melhorar o desempenho muitas vezes é necessário que duas ou mais técnicas sejam combinadas em um sistema híbrido. O objetivo dos sistemas híbridos é a combinação de várias técnicas de Aprendizado de Máquina, de forma que o desempenho delas em conjunto seja superior ao desempenho das técnicas aplicadas individualmente (REZENDE, 2003). Nesta seção serão descritas duas formas de combinar as Redes Neurais Artificiais e os Algoritmos Genéticos, para assim poder tirar vantagens de ambas as técnicas.

Na Seção 3.2 foi descrito que os maiores pontos críticos das Redes Neurais Artificiais são a configuração da rede (número de camadas ocultas, número de neurônios em cada camada oculta e a forma de conexão entre as diversas camadas) e dos principais parâmetros de aprendizado (taxa de aprendizado η, a constante de *momentum* α, faixa de inicialização dos pesos, dentre outros), de forma que se estes não estiverem corretamente configurados, o aprendizado não será realizado de forma satisfatória (HAYKIN, 2001). Assim, geralmente são necessárias muitas horas de um especialista humano para a configuração manual da Rede Neural e dos diversos parâmetros de aprendizado. Uma alternativa viável é a utilização de Algoritmos Genéticos para a otimização da topologia e dos parâmetros da rede (ou seja, é criada uma população de topologias e parâmetros de ANNs), de forma que estas tarefas não sejam mais realizadas manualmente, o que também garante melhores resultados (TSAI; CHOU; LIU, 2006). Esta alternativa é biologicamente fundamentada, uma vez que o sistema nervoso da maioria dos seres vivos também é resultado da evolução natural (REZENDE, 2003).

Outra forma de se utilizar os Algoritmos Genéticos em conjunto com as Redes Neurais é fazendo com que os pesos da Rede Neural sejam evoluídos através do uso de Algoritmos Genéticos ao invés do uso das técnicas de aprendizado tradicionais (*back-propagation*, RPROP, etc) (MAN; TANG; KWONG, 1999). De fato, uma das principais limitações das redes MLP treinadas com o algoritmo *back-propagation* é a necessidade de uma base de treinamento com as saídas desejadas, o que nem sempre é possível de ser obtido. Em problemas de robótica autônoma, por exemplo, não é comum se ter informações locais para a correção dos pesos, mas apenas uma medida que informa o desempenho de cada solução em relação as outras (REEVE, 1999). Esta medida de desempenho não é suficiente para o cálculo do erro através do algoritmo *back-propagation*, mas geralmente é suficiente para a definição de uma função de *fitness* a ser utilizada nos Algoritmos Genéticos.

Assim, a combinação das duas técnicas permite que se utilize todo o poder de representação das Redes Neurais Artificiais como um aproximador universal de funções (HAYKIN, 2001) em problemas onde não é possível se obter de antemão informações locais para a descida do gradiente. Esta alternativa permite que se expanda em muito a classe de problemas que podem

ser resolvidos utilizando as Redes Neurais Artificiais (MAN; TANG; KWONG, 1999). Além disso, segundo Edelman (1987), o uso de GAs para a evolução dos pesos de uma ANN utilizada no controle do caminhar é uma alternativa biologicamente plausível, pois a evolução natural desempenhou um papel preponderante na formação dos CPGs (Seção 2.5) da maioria dos seres vivos (PFEIFER; SCHEIER, 1999).

3.4 Método de Powell

O método de Powell (1964) é uma técnica de minimização multidimensional que se constitui no protótipo da maioria dos métodos direcionais de otimização não-linear. Ele determina e especifica o conjunto de vetores que estipulam as direções nas quais o mínimo ótimo deve ser procurado. Este conjunto de vetores inclui vetores unitários como membros. Usando-se uma técnica de minimização unilateral e começando a partir de um ponto correspondente a uma solução inicial, movimentos são feitos ao longo da primeira direção até o seu ponto mínimo, a partir do qual a solução é investigada ao longo da segunda direção até se encontrar o mínimo nela, e assim por diante. O ciclo de iterações sobre o conjunto de direções é repetido até que não se possa fazer outros movimentos, o que significa ter se atingido as regiões mais baixas do vale de mínima, indicando que a função objetiva foi completamente minimizada (ACTON, 1970). A Figura 3.14 ilustra este processo.

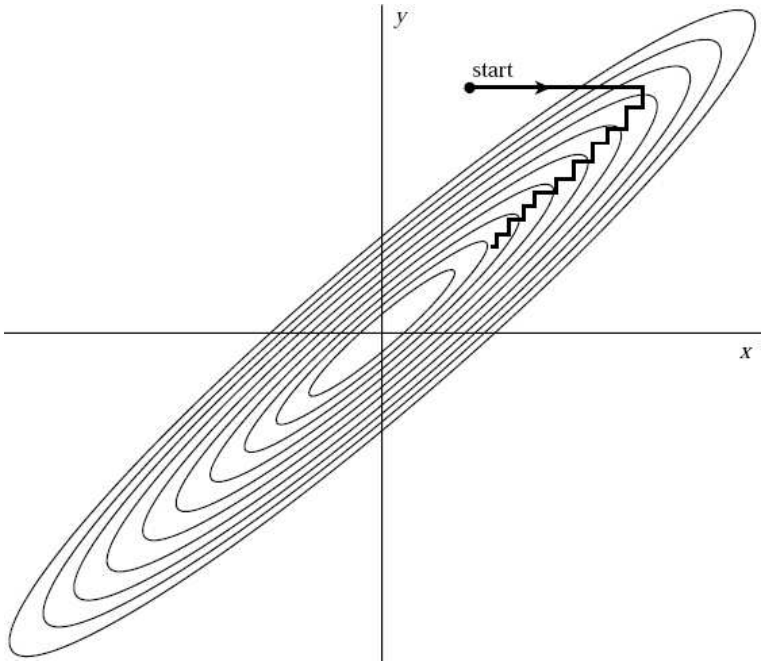


Figura 3.14: Minimização realizada pelo método de Powell (PRESS et al., 1992)

Os vetores direcionais podem ser, eventualmente, atualizados no final de cada iteração. Existem situações numa otimização multidimensional em que a segunda derivada da função em uma direção pode ser bem maior que nas outras. Quando isto ocorre, ciclos repetidos sobre todos os N vetores unitários são necessários para se conseguir algum progresso (muitas vezes insignificante) na busca do ponto ótimo. Assim sendo, torna-se necessário obter outros componentes para o conjunto de vetores direcionais, mais eficientes que as direções ortogonais. O

processo de minimização se inicia procurando a solução mínima ao longo de cada direção ortogonal, variando-se apenas um parâmetro por etapa. Os resultados da iteração anterior são usados para melhorar a eficiência da investigação durante a próxima iteração. Se algumas condições pré-estabelecidas são satisfeitas, a direção anterior ao longo da qual a função objetivo teve seu maior decréscimo é descartada em favor de uma nova direção (ACTON, 1970).

Na prática, o método de Powell para em duas situações. Primeira, se algumas direções de busca não ganham melhorias de modo algum, então as direções de busca subseqüentes não estarão conjugadas. A segunda situação é que depois de algumas iterações, as direções de busca tendem a tornar-se paralelas por causa da imprecisão numérica ou por causa da natureza não quadrática da função que está sendo minimizada. Provavelmente a mais simples modificação para contornar essa situação, e muitas vezes a mais prática, é inicializar o processo com buscas unidirecionais toda vez que o processo de otimização convergir lentamente. O método de Powell será viável para problemas de otimização pequenos ou quando o custo de uma avaliação de função for pequeno (PRESS et al., 1992).

O controle do caminhar utilizando técnicas de Aprendizado de Máquina, como as descritas neste capítulo, vem sendo aplicado em robôs móveis com bastante sucesso. O próximo capítulo descreve diversos trabalhos relativos ao controle do caminhar de robôs com pernas.

4 Trabalhos relacionados

Nesta seção serão descritas diversas abordagens do estado da arte utilizadas para a configuração do caminhar em robôs dotados de pernas. A Seção 4.1 traz um breve histórico das pesquisas envolvendo robôs que caminham. A Seção 4.2 descreve alguns trabalhos recentes realizados utilizando robôs de seis pernas. A Seção 4.3 descreve alguns trabalhos que utilizam robôs de quatro pernas. A Seção 4.4 descreve alguns trabalhos recentes envolvendo robôs bípedes. A Seção 4.5 descreve a área de vida artificial e alguns trabalhos desenvolvidos nesta área. Por último, a Seção 4.6 traz as considerações finais do capítulo.

4.1 História dos robôs caminhantes

Uma das primeiras referências históricas sobre a construção de artefatos caminhantes data de 1893, quando L. A. Rygg patenteou o projeto de um cavalo mecânico (Figura 4.1). Este cavalo, que nunca chegou a ser construído, utilizava engrenagens para fazer com que as patas descrevessem trajetórias elípticas. Segundo Raibert (1986), até meados do século vinte a pesquisa de máquinas que caminham (*walk machines*) era baseada no uso de engrenagens que descreviam trajetórias fixas, o que impossibilitava adaptação do caminhar em tempo real.

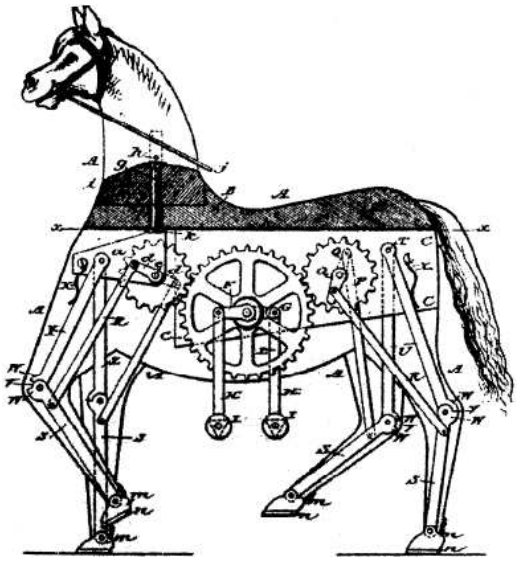


Figura 4.1: Projeto de um cavalo mecânico de L. A. Rygg (RAIBERT, 1986)

Em 1966, McGhee e Frank criaram o “Phoney Pony” (falso pônei) (MCGHEE, 1968, 1976), que foi o primeiro robô com pernas a ser controlado via computador. O “Phoney Pony” (Fi-

gura 4.2) possuía quatro pernas idênticas, que eram controladas através de um autômato finito. A construção deste robô foi muito importante para a área, pois permitiu que fossem identificados os diversos problemas envolvidos na tarefa de construir máquinas que caminham.

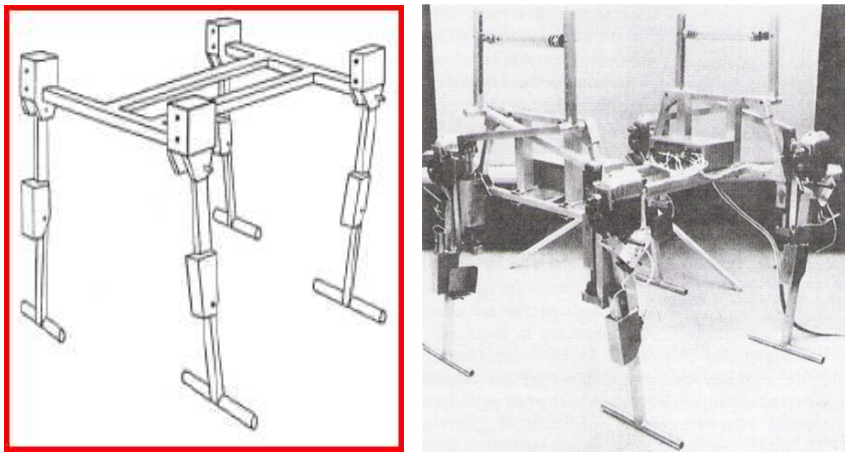


Figura 4.2: Esquema do "Phoney Pony" de McGhee e Frank (BEKEY, 2005)

Em 1968, R. Mosher construiu o *General Electric Walking Truck* (Figura 4.3), que era um veículo que possuía quatro pernas, controladas manualmente. Este veículo era muito difícil de ser conduzido, pois o operador precisava controlar cada uma das pernas de forma individual (BEKEY, 2005).

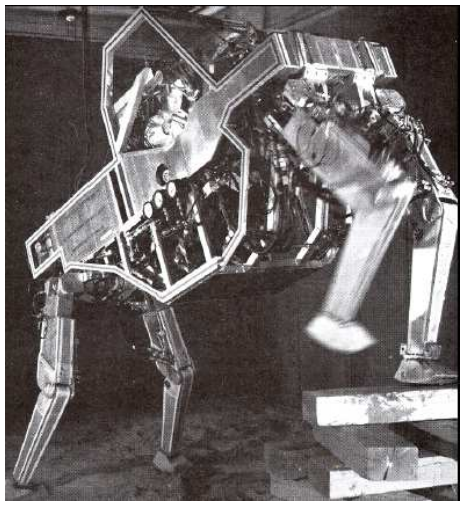


Figura 4.3: General Electric Walking Truck (BEKEY, 2005)

Nos anos 80, Marc Raibert realizou diversas pesquisas relativas à estabilidade dinâmica do caminhar, primeiro em robôs com uma única perna (*monopod*) (RAIBERT; BROWN; CHEPPONIS, 1984), e depois em robôs de duas, quatro e seis pernas (RAIBERT, 1986). O controle das articulações foi realizado através de um autômato, que tratava todas as pernas como se fossem uma única perna virtual. A Figura 4.4 mostra dois modelos de robôs desenvolvidos por Raibert, um *monopod* e um *tetrapod*.

Nos anos 90, foi desenvolvido o robô Dante (LEMONICK, 1994), que realizou a exploração do interior de um vulcão situado na Antártida. Este robô, originalmente concebido por um grupo

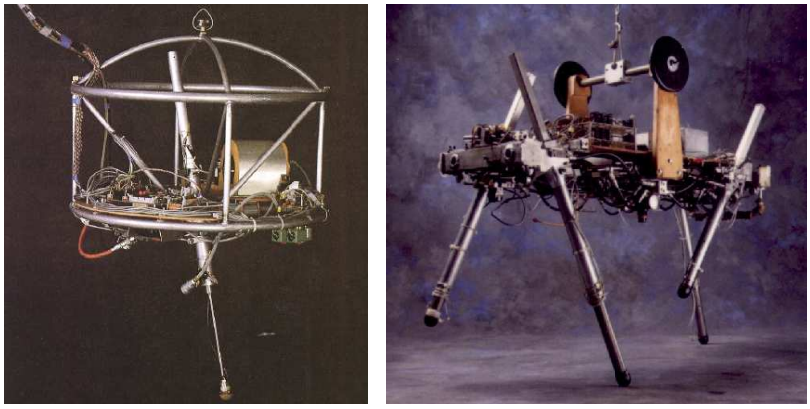


Figura 4.4: Robôs com estabilidade dinâmica (RAIBERT, 1986)

da NASA para a exploração de outros planetas, foi concebido para resistir as condições mais extremas: frio, calor e terrenos acidentados. A Figura 4.5 mostra o robô Dante.

Figura 4.5: Robô Dante (LEMONICK, 1994)

Através deste breve histórico é possível vislumbrar as dificuldades da tarefa em questão, bem como as tentativas de solucionar estas dificuldades que vem sendo pesquisadas a várias décadas. Na próxima seção são descritos trabalhos recentes envolvendo robôs de seis pernas.

4.2 Hexapods

Os robôs de seis pernas, chamados de *hexapods*, são mais fáceis de serem controlados que os robôs de duas e quatro pernas, pois conseguem manter a estabilidade estática com somente a metade das pernas em contato com o chão. Nesta seção são descritos diversos trabalhos realizados utilizando *hexapods*.

Em (ZYKOV; BONGARD; LIPSON, 2004), foram utilizados Algoritmos Genéticos para a evolução do caminhar de um robô de nove pernas (Nonaped), dispostas como mostra a Figura 4.6(a). Este robô é composto de três seções que se movem umas em relação às outras

através do uso de 12 atuadores pneumáticos. As nove pernas do robô são fixas nas seções e não possuem articulações. O controle do robô é realizado através de uma tabela de estados, onde para cada estado é definida duração do mesmo e a posição de cada atuador pneumático (0 = contraído; 1 = expandido). A tabela de estados foi evoluída através do uso de Algoritmos Genéticos, que foram aplicados inicialmente em um robô simulado (Figura 4.6(b)), e após a evolução o controlador foi portado para o robô real onde foram realizados ajustes finais.

(a) Robô real (b) Robô simulado

Figura 4.6: Robô Nonaped utilizado em (ZYKOV; BONGARD; LIPSON, 2004)

Para a configuração do caminhar, Parker (PARKER; RAWLINS, 1996; PARKER; BRAUN; CYLIAX, 1997; PARKER; MILLS, 1998) desenvolveu um novo tipo de Algoritmo Genético, chamado de Algoritmo Genético Cíclico (*Cyclic Genetic Algorithms* – CGA), que opera de forma similar aos Algoritmos Genéticos tradicionais, mas difere destes pelo fato dos genes em cada cromossomo representarem tarefas ao invés de particularidades. No CGA, cada cromossomo é dividido em uma parte de inicialização, que é executada apenas uma vez, e uma parte iterativa, que é repetida continuamente durante o caminhar. A avaliação do *fitness* é realizada gene por gene de forma analítica, sem o uso de simulação. Em (TOTH; PARKER, 2003), é descrita a utilização do CGA para o controle do caminhar em um robô *hexapod* real (Figura 4.7) com dois graus de liberdade por perna, controlados através de 12 servo-motores.

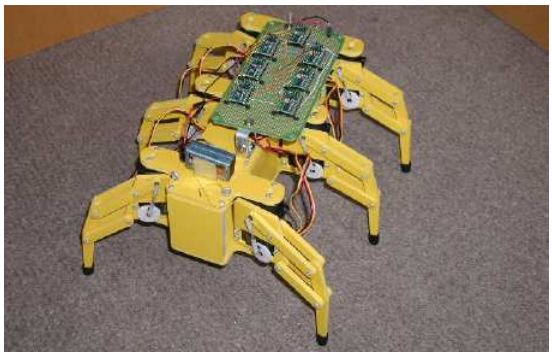


Figura 4.7: Robô Lynxmotion Hexapod II utilizado em (TOTH; PARKER, 2003)

Em (WEINGARTEN et al., 2004), o caminhar do robô HRex (Figura 4.8) foi definido através de um modelo de máquina de estados cujos parâmetros foram evoluídos através de uma versão modificada do algoritmo *Nelder-Mead descent* (NELDER; MEAD, 1965). O robô HRex

possui seis pernas curvas que possuem apenas um grau de liberdade no plano *sagittal*, e cada uma das pernas pode girar em até 360°.

Figura 4.8: Robô HRex utilizado em (WEINGARTEN et al., 2004)

Porta (2000) propõe uma versão modificada do algoritmo de Aprendizado por Reforço *Q-learning*, chamada de ρ*-learning*. Na versão proposta, o algoritmo *Q-Learning* é adaptado a tarefa de escolher o melhor subconjunto de informações sensoriais a ser utilizado no planejamento das ações do robô, e não no controle do caminhar propriamente dito. A Figura 4.9 mostra o robô Genghis-II, utilizado nos experimentos.

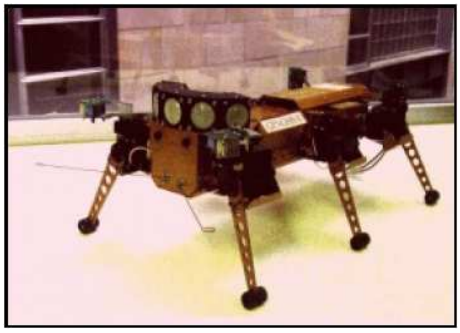


Figura 4.9: Robô Genghis-II utilizado em (PORTA, 2000)

Devido ao número elevado de pernas dos robôs *hexapod*, em todos os experimentos descritos acima foram utilizados robôs com articulações simplificadas (apenas um ou dois graus de liberdade por perna) e alguma forma de simplificação do sistema de controle, visando assim reduzir o espaço de estados e tornar o aprendizado mais fácil de ser realizado.

4.3 Tetrapods

Ultimamente os robôs de quatro pernas, chamados de *tetrapods*, vem ganhando bastante destaque, devido principalmente ao surgimento no mercado de robôs comerciais como o Sony Aibo (FUJITA, 2001). Nesta seção serão descritas diversas pesquisas realizadas visando a configuração do caminhar de robôs *tetrapod*.

Em (BUSCH et al., 2002), foi utilizada Programação Genética (*Genetic Programming* – GP) para definição do caminhar em diversos modelos de robôs simulados através do pacote de

software DynaMechs[1]. Nesta abordagem, a Programação Genética é implementada através de uma representação linear dos genomas, onde cada indivíduo da população é composto de uma seqüência de instruções para o controle dos robôs. A Figura 4.10 mostra alguns modelos de robôs utilizados nas simulações.

Figura 4.10: Robôs simulados utilizados em (BUSCH et al., 2002)

Em (REEVE; HALLAM, 2005), foram testadas diversas arquiteturas de Redes Neurais Artificiais para o controle do caminhar de robôs simulados de quatro pernas, como por exemplo o da Figura 4.11. As arquiteturas de Redes Neurais testadas foram redes *sigmoidais* padrão (RUMELHART; HINTON; WILLIAMS, 1986), redes oscilatórias de primeira ordem do tipo *Continuous Time Recurrent Neural Networks* – CTRNN (BEER, 1995), redes de segunda ordem (TAGA, 1995) e redes de terceira ordem (EKEBERG, 1993). Para o ajuste dos pesos das Redes Neurais foram utilizados Algoritmos Genéticos com o esquema de seleção do tipo *tournament*, *crossover* em um ponto com probabilidade de 0,8 e a taxa de mutação foi de 0,1. A função de *fitness* utilizada foi a velocidade média do robô durante cada experimento. O artigo destaca que as redes de terceira ordem obtiveram um melhor desempenho que as demais redes em relação ao poder de representação, pois foi possível obter formas rápidas de caminhar utilizando menos neurônios na camada oculta.

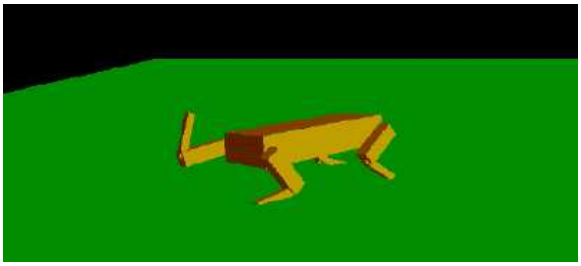
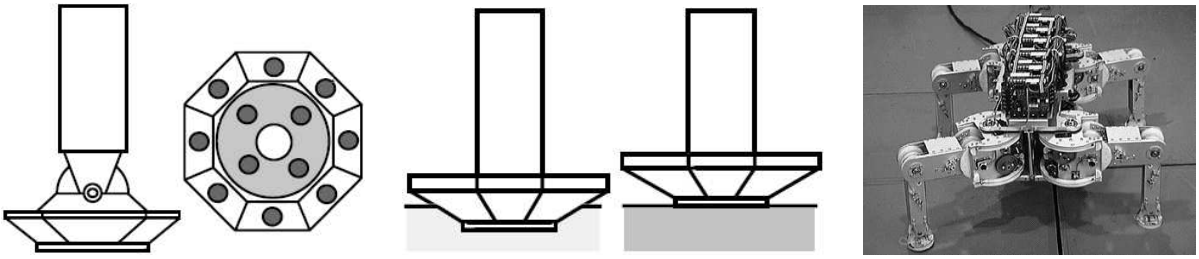


Figura 4.11: Exemplo de robô simulado utilizado em (REEVE; HALLAM, 2005)

Em (SHIMADA et al., 2002), foi definido um sistema de controle para um robô quadrúpede com quatro graus de liberdade em cada uma das pernas, que é capaz de caminhar em terreno macio. Neste sistema, o movimento das juntas é realizado através do uso de osciladores neurais que imitam o comportamento de geradores centrais de padrões (*central pattern generator* – CPG). Conforme visto na Seção 2.5, um CPG é uma espécie de gerador rítmico biológico que está presente na maioria dos sistemas de locomoção de diversos animais (MATSUOKA, 1987). Para que o caminhar em superfícies macias fosse possível, as patas do robô foram desenvolvidas

[1]DynaMechs – http://dynamechs.sourceforge.net/

em um formato especial, como mostra a Figura 4.12(a), e são dotadas de diversos sensores de pressão. Assim, quando o robô caminha em superfícies sólidas, apenas os sensores centrais da pata são acionados, mas quando ele caminha em superfícies macias, os sensores centrais são acionados juntamente com os sensores laterais, como mostra a Figura 4.12(b). A Figura 4.12(c) mostra o robô Taitan-VII, que foi utilizado nos experimentos.



(a) Esquema das patas (b) Terrenos macio e duro (c) Robô Taitan-VIII

Figura 4.12: Esquema do robô utilizado em (SHIMADA et al., 2002)

Em (HOLLAND; SNAITH, 1992), foi utilizado Aprendizado por Reforço para a configuração do caminhar em um robô dotado de quatro pernas idênticas, com dois graus de liberdade em cada perna. Neste robô, a movimentação das juntas ocorre por meio de atuadores pneumáticos ("pistões"), e assim cada uma das juntas do robô possui apenas dois estados, um com o atuador pneumático contraído e outro com o atuador expandido. Para a otimização do caminhar, foi utilizado o Aprendizado por Reforço com um versão modificada do algoritmo *Q-learning* (SUTTON; BARTO, 1998). A Figura 4.13 mostra o robô Igor, utilizado nos experimentos.

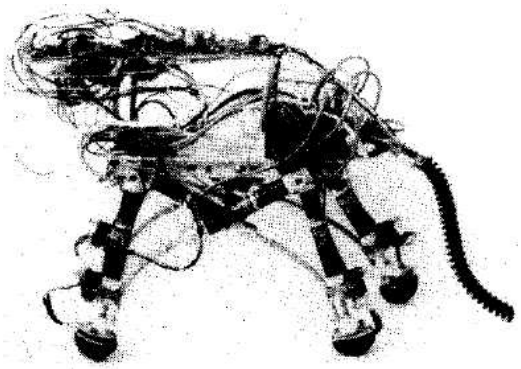
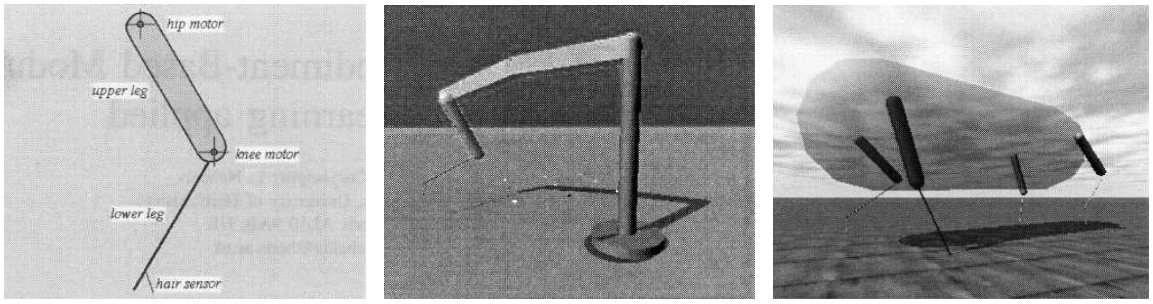


Figura 4.13: Robô Igor utilizado em (HOLLAND; SNAITH, 1992)

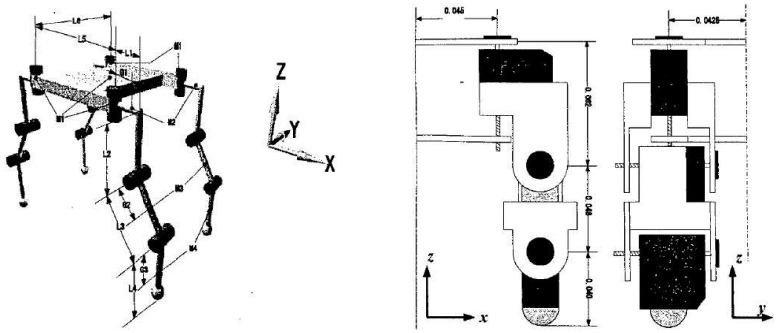
Em (JACOB; POLANI; NEHANIV, 2005), o Aprendizado por Reforço foi utilizado para a configuração do caminhar em um robô quadrúpede com dois graus de liberdade em cada perna simulado através da biblioteca *Open Dynamics Engine* (ODE). A Figura 4.14(a) mostra o esquema de uma das pernas do robô. Devido ao fato do Aprendizado por Reforço não ser eficiente quando o espaço de estados é contínuo e/ou possui grande dimensionalidade (HAYKIN, 2001), o aprendizado foi realizado inicialmente em apenas uma perna, utilizando para isto um suporte de treinamento, como mostra a Figura 4.14(b). Após a perna ter sido treinada individualmente, o aprendizado foi realizado em robô simulado de quatro pernas (Figura 4.14(c)), visando sincronizar o movimento das diversas pernas.



(a) Esquema de uma perna (b) Suporte de treinamento (c) Robô simulado

Figura 4.14: Detalhes do robô utilizado em (JACOB; POLANI; NEHANIV, 2005)

Em (MURAO; TAMAKI; KITAMURA, 2001), foi utilizado um robô quadrúpede com três graus de liberdade em cada perna, onde os movimentos de cada junta durante o caminhar são definidos através do uso de uma função senoidal. Para a otimização dos parâmetros desta função senoidal, foi utilizado o Aprendizado por Reforço modular com o algoritmo *stochastic gradient ascent*. A Figura 4.15(a) mostra um esquema do robô utilizado nas simulações, e a Figura 4.15(b) mostra um esquema de uma das pernas do robô. O aprendizado foi realizado inteiramente no robô real.



(a) Esquema das juntas (b) Esquema de uma perna

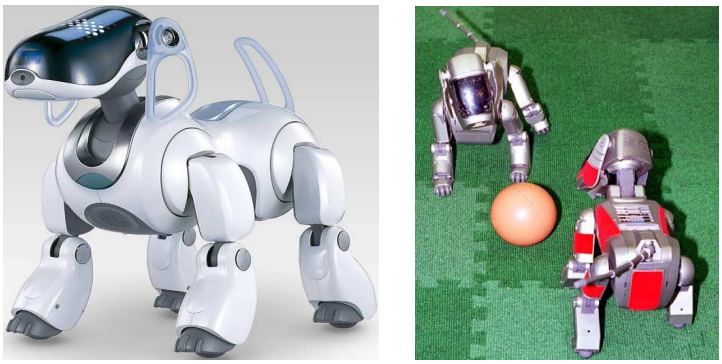
Figura 4.15: Esquema do robô utilizado em (MURAO; TAMAKI; KITAMURA, 2001)

4.3.1 Robôs Aibo

Um dos fatores que tem trazido destaque à área de robôs dotados de pernas é a realização da RoboCup[2], que é um campeonato mundial de futebol de robôs. Na RoboCup, existe uma categoria que é disputada por robôs do tipo Sony Aibo (Figuras 4.16(a) e 4.16(b)), que são robôs que possuem 20 graus de liberdade (3 em cada perna), microfones stereo, uma câmera CCD, um sensor de distância infravermelho, um giroscópio, dois acelerômetros e *bumpers* embaixo das patas. A movimentação do robô é realizada por um módulo de locomoção que controla o caminhar do robô através de um conjunto de parâmetros especificado pelo usuário (FUJITA,

[2]RoboCup – http://www.robocup.org

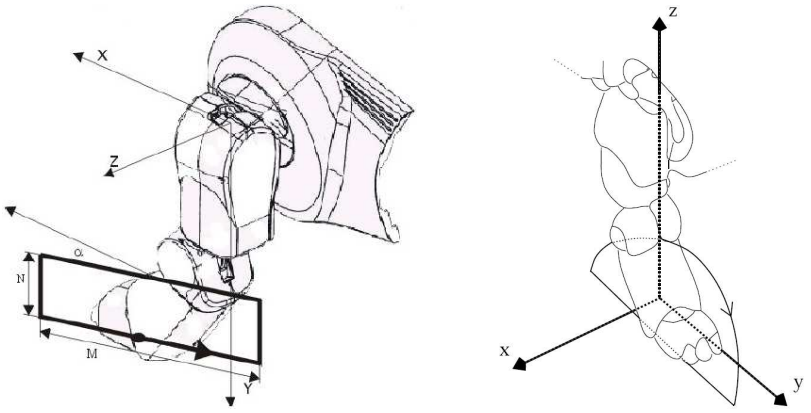
2001). A forma de caminhar original dos robôs Aibo não é muito rápida, mas como no futebol a velocidade é um fator importante, várias pesquisas vem sendo realizadas para acelerar o máximo possível o caminhar destes robôs. Nesta seção serão descritos alguns trabalhos que apontam nesta direção.



(a) (b)

Figura 4.16: Modelos de robôs Sony Aibo (GOLUBOVIC; HU, 2002)

Em (GOLUBOVIC; HU, 2002, 2003), foi utilizada uma abordagem de alto nível para o aumento da velocidade dos robôs, que visa definir a trajetória percorrida pelas patas do robô (*endpoints*) através de um retângulo (Figura 4.17(a)). Os parâmetros deste retângulo são otimizados através de Algoritmos Genéticos, e a função de *fitness* utilizada leva em conta não somente a distância percorrida pelo robô, mas também a taxa de instabilidade, que é calculada através das leituras realizadas por um giroscópio interno presente nos robôs Aibo. O uso da taxa de instabilidade visa tornar o caminhar obtido não somente veloz, mas também estável.



(a) Retângulo (b) Meia-elipse

Figura 4.17: Trajetória das patas do robô (GOLUBOVIC; HU, 2002; KOHL; STONE, 2004a)

Em (KOHL; STONE, 2004a), a trajetória das patas foi definida através de uma meia-elipse (Figura 4.17(b)), e para a otimização desta meia-elipse foram utilizadas as técnicas *hill climbing* (PRESS et al., 1992), *downhill simplex*, também chamado de Amoeba (PRESS et al., 1992),

Algoritmos Genéticos (GOLDBERG, 1989) e Aprendizado por Reforço com o algoritmo *policy gradient* (SUTTON; BARTO, 1998). Uma comparação entre as diversas técnicas foi realizada, e as técnicas *hill climbing* e *policy gradient* obtiveram um melhor desempenho que as demais. Em (KOHL; STONE, 2004b), os mesmos autores focaram o aumento da velocidade dos robôs Aibo utilizando apenas Aprendizado por Reforço através do algoritmo *policy gradient*. Eles destacam que um dos pontos fracos do uso deste algoritmo é que a velocidade do robô só é aumentada quando o aprendizado parte de uma configuração realizada manualmente. Quando os parâmetros da meia-elipse são inicializados aleatoriamente, o comportamento do robô se torna muito instável e as velocidades obtidas são muito baixas.

Também utilizando uma meia-elipse para a definição do caminhar, Rofer (2005) utiliza algoritmos evolucionários para a otimização dos parâmetros desta meia-elipse, mas com a preocupação de acelerar o caminhar não apenas em linha reta, mas em todas as direções. Para isto, foi utilizado um sensor do tipo odômetro de forma que a distância percorrida pudesse ser calculada com maior precisão.

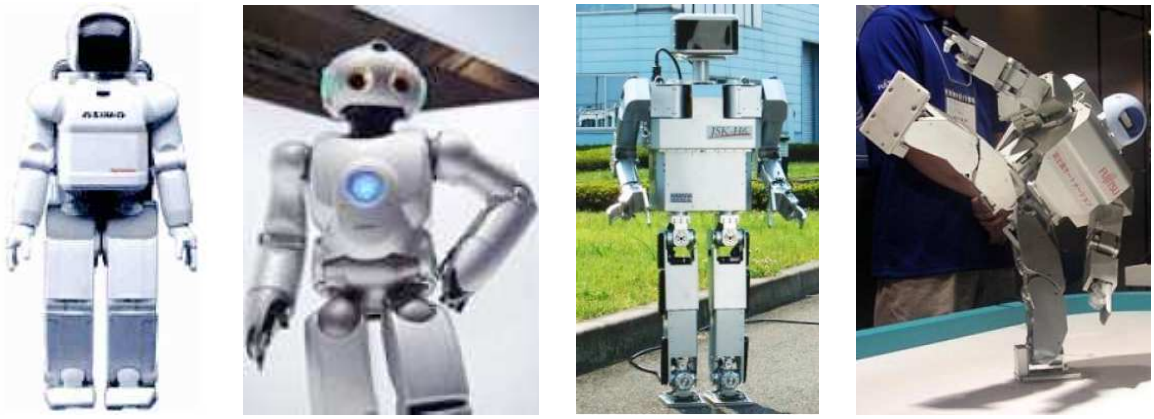
Já em (CHERNOVA; VELOSO, 2004), foi utilizada uma abordagem diferente para a locomoção rápida do robô, onde a trajetória do corpo é representada através de um modelo de aceleração que mantém a direção desejada do movimento. Nesta abordagem, ao invés da trajetória dos *endpoins* (patas) ser definida precisamente em todos os momentos do caminhar, ela foi definida apenas para os momentos em que as patas estão em contato com o chão. Na parte em que a pata está no ar a única restrição imposta foi de que a pata não colida com o chão ou com as outras partes do robô. Para a otimização dos parâmetros do caminhar, foram utilizados Algoritmos Genéticos com populações sobrepostas (*overlapping populations*), juntamente com uma técnica de radiação, que visa evitar a convergência prematura de todos os indivíduos para algum mínimo local. A evolução dos parâmetros foi realizada em quatro robôs reais do tipo Aibo de forma paralela, reduzindo assim o tempo de aprendizado.

4.4 Bípedes

Recentemente, várias indústrias de tecnologia tem desenvolvido robôs bípedes humanóides que conseguem caminhar de forma bastante estável, dentre os quais podemos destacar o Honda Asimo (Figura 4.18(a)), o Sony SDR-4X (Figura 4.18(b)), o Kawada Industries H6 (Figura 4.18(c)) e o Fujitsu HOAP-2 (Figura 4.18(d)). Todos estes robôs utilizam para caminhar o cálculo do *Zero Moment Point* - ZMP, que é o ponto sobre a superfície do chão onde as forças reativas e gravitacionais atuam (SHIN, 1996). Em outras palavras, o ZMP é o ponto de equilíbrio, de forma que enquanto o centro de gravidade do robô permanecer sobre este ponto ele se manterá estável (LINGYUM; ZENGQI, 2004). A Figura 4.19 mostra um esquema do triângulo de suporte formado pelo ZMP.

A desvantagem desta técnica é que um caminhar controlado pelo ZMP tende a ser até vinte vezes mais dispendioso em termos de gastos de energia do que o caminhar utilizado pelos seres humanos (VAUGHAN, 2003b), que é chamado de *controlled falling*. Isto se deve ao fato do ser humano conseguir aproveitar melhor a energia cinética e potencial durante o caminhar (HEINEN; OSÓRIO, 2006g).

Em (WOLFF; NORDIN, 2002) é utilizada Programação Genética para configurar o caminhar do robô Elvina, mostrado na Figura 4.20(a). Este robô possui cinco graus de liberdade em



(a) Honda Asimo (b) Sony SDR-4X (c) Kawada H6 (d) Fujitsu HOAP-2

Figura 4.18: Robôs bípedes modernos

cada perna, um em cada braço, um na cabeça e um no torso. A função de *fitness* utilizada levou em conta a distância percorrida pelo robô e a altura da cabeça deste, através da fórmula:

$$f = W\left(1 - \frac{h_{start}}{h_{stop}}\right) - (d_{left} + d_{right}), \tag{4.1}$$

onde h_{start} é a altura inicial da cabeça do robô, h_{stop} é a altura final da cabeça, d_{left} é a posição inicial do robô e d_{right} é a posição ao final do período de avaliação.

A representação utilizada para a Programação Genética foi de uma *virtual register machine* (VRM) com genomas representados linearmente, e o esquema de seleção foi do tipo *tournament*. A evolução foi realizada inteiramente no robô real, o que consumiu seis meses de simulações e muitos componentes do robô foram danificados e tiveram de ser substituídos. Já em (WOLFF; NORDIN, 2003a, 2003b), os mesmos autores utilizam um modelo do robô Elvina simulado em ODE (Figura 4.20(b)) para fazer a evolução da Programação Genética, o que segundo eles facilitou muito a tarefa de aprendizado.

Em (WYETH; KEE; YIK, 2003), a trajetória das pernas do robô bípede GuRoo foi definida através de uma tabela de estados (*key frames*) que define para cada estado as posições nos eixos

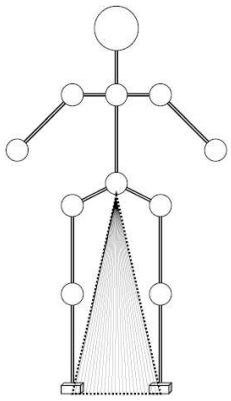
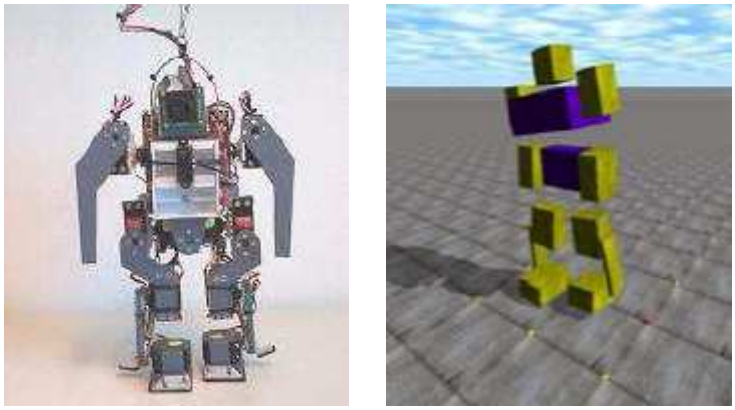


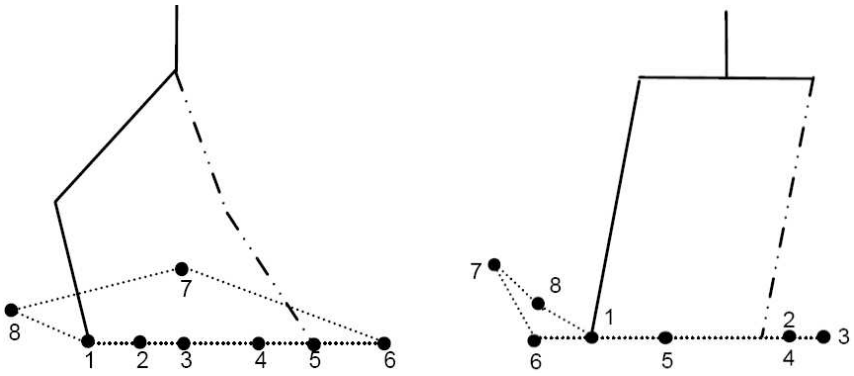
Figura 4.19: Triângulo de suporte do ZMP (VAUGHAN, 2003b)



(a) Robô real (b) Robô simulado

Figura 4.20: Robô Elvina utilizado em (WOLFF; NORDIN, 2002)

x, *y* e *z* dos *endpoints* do robô. Este tipo de caminhar é conhecido como *locus based gait*. A Figura 4.21 mostra exemplos de *key frames* nos planos *saggital* e frontal.

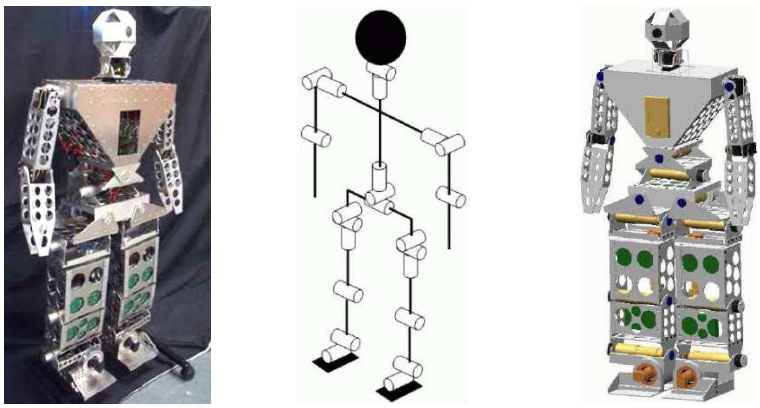


(a) Plano *saggital* (b) Plano frontal

Figura 4.21: Exemplo de *key frames* gerados pela técnica de Wyeth, Kee e Yik (2003)

A posição dos pés do robô em cada um dos *key frames* é calculada considerando o *Zero Moment Point* - ZMP, que é o ponto sobre o chão no qual as forças gravitacionais se anulam, de forma que se o centro de gravidade do robô permanecer sobre este ponto ele se mantém equilibrado (LINGYUM; ZENGQI, 2004). O robô GuRoo possui no total 23 graus de liberdade, dispostos como mostra a Figura 4.22(b). Em (ROBERTS; KEE; WYETH, 2003), os mesmos autores utilizam Algoritmos Genéticos para melhorar o caminhar do robô GuRoo, de forma a torná-lo mais estável. As simulações foram realizadas inicialmente em um robô simulado através do pacote de software Dynamechs (Figura 4.22(c)) e depois foram realizadas utilizando o robô real (Figura 4.22(a)).

Em (ENDO; MAENO; KITANO, 2003), foram utilizados Algoritmos Genéticos (GA) para a evolução da morfologia dos robôs. Na maioria das abordagens em robótica autônoma, a estrutura e a forma de controle são projetadas separadamente, mas segundo os autores elas deveriam ser feitas em conjunto, pois a estrutura do robô afeta o sistema de controle, de forma que

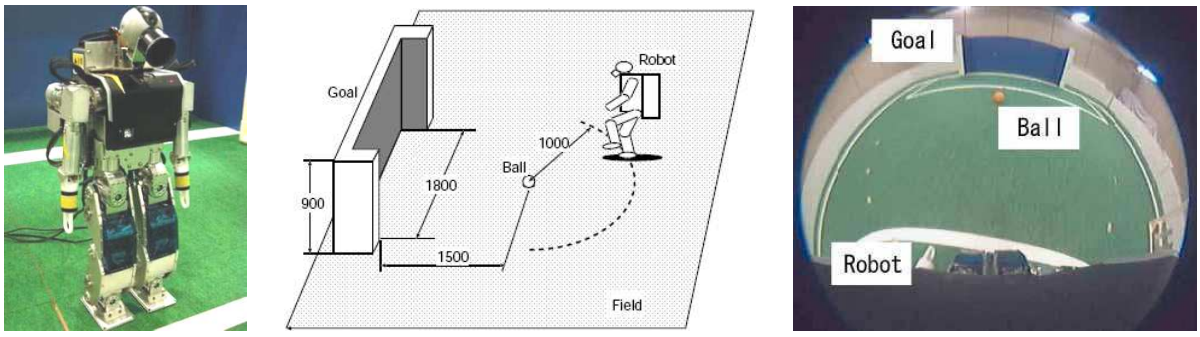


(a) Robô real (b) Juntas (c) Robô simulado

Figura 4.22: Robô GuRoo (WYETH; KEE; YIK, 2003; ROBERTS; KEE; WYETH, 2003)

mudanças na morfologia do robô poderiam facilitar a tarefa de caminhar. Na abordagem desenvolvida, foi utilizada Computação Evolucionária para definir a estrutura do robô, e o controle foi realizado através de redes oscilatórias cujos pesos foram atualizados utilizando GAs.

Em (ASADA et al., 2003), é utilizado Aprendizado por Reforço para que o robô HOAP-1 (Figura 4.23(a)) se locomova até uma bola e chute esta em direção ao gol, como mostra a Figura 4.23(b). O controle do caminhar foi realizado através do uso de um CPG implementado através de osciladores neurais, e o algoritmo *Q-learning* (SUTTON; BARTO, 1998) foi utilizado para otimizar os parâmetros dos osciladores. Nesta abordagem, a trajetória dos *endpoints* foi definida através do uso de uma meia-elipse. Em (OGINO et al., 2004), os mesmos autores descrevem em mais detalhes o uso da informação visual no planejamento das ações deste robô. A Figura 4.23(c) mostra a visão do robô, que é obtida através de uma câmera CCD com lentes do tipo *fish-lens*.



(a) HOAP-1 (b) Esquema da tarefa (c) Visão do robô

Figura 4.23: Abordagem utilizada em (ASADA et al., 2003; OGINO et al., 2004)

Em (TEDRAKE; ZHANG; SEUNG, 2004), foi utilizado Aprendizado por Reforço com o algoritmo *stochastic policy gradient* para a configuração do caminhar em um robô inspirado em um *passive dynamic walker* - PDW. Um PDW é um tipo de robô desenvolvido por McGeer (1990), que é capaz de caminhar em uma superfície inclinada utilizando apenas a energia

potencial do declive. A Figura 4.24(a) mostra um exemplo de PDW. O robô utilizado nos experimentos de Tedrake, Zhang e Seung (2004) (Figura 4.24(b)) possui uma junta passiva no quadril e dois graus de atuação (juntas com servo-motores) em cada pé. Este robô possui os joelhos fixos e os pés grandes e curvos, o que aumenta a estabilidade e facilita a tarefa de caminhar.

(a) Exemplo de PDW (b) Robô utilizado

Figura 4.24: Robô utilizado em (TEDRAKE; ZHANG; SEUNG, 2004)

Em (HITOMI et al., 2005), também foi utilizado um robô inspirado em um PDW, mas ao contrário do anterior, este possui joelhos móveis e os pés pequenos e cilíndricos, como mostra a Figura 4.25. A otimização do caminhar também foi realizada utilizando Aprendizado por Reforço com o algoritmo *stochastic policy gradient*, e para facilitar a tarefa do caminhar, o aprendizado foi realizado primeiro com os joelhos do robô travados e depois com eles destravados, o que permitiu a redução do espaço de estados.

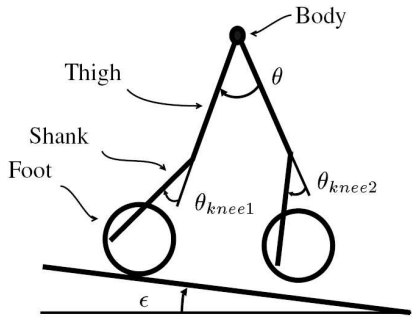


Figura 4.25: Esquema do robô utilizado em (HITOMI et al., 2005)

Em (VAUGHAN, 2003b, 2003a; VAUGHAN; DI PAOLO; HARVEY, 2004), foi proposto e desenvolvido um robô bípede simulado, que também segue os princípios dos PDWs, mas com alguns aperfeiçoamentos: possui joelhos, os pés são pequenos e planos, se desloca tanto em superfícies inclinadas quanto planas, e não é passivo: o robô possui motores angulares nas juntas. A Figura 4.26 mostra um o robô simulado caminhando em uma superfície plana. Este robô foi simulado com a biblioteca ODE, e o controlador utilizou Redes Neurais com os pesos evoluídos através de Algoritmos Genéticos.

Pode-se notar que apesar dos grandes avanços que vem ocorrendo ultimamente na área de robótica, ainda existem obstáculos que precisam ser superados até que se consiga desenvolver um robô bípede que caminhe de forma estável em superfícies irregulares e que seja eficiente em

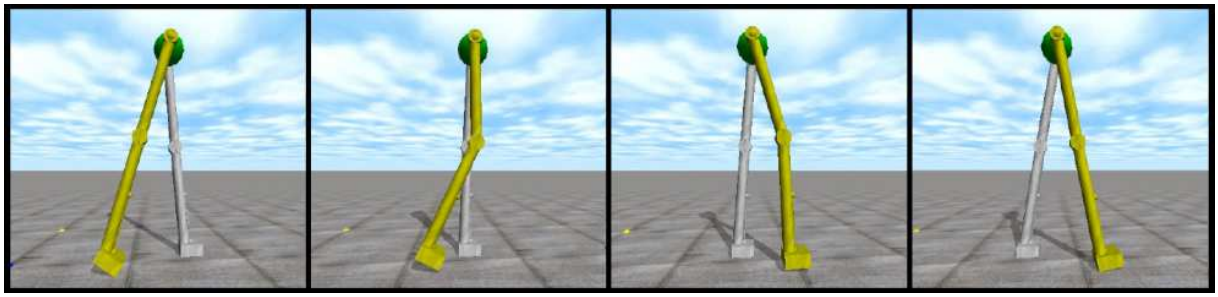


Figura 4.26: Robô bípede simulado em (VAUGHAN, 2003a)

termos de: (i) gastos de energia; (ii) autonomia; (iii) robustez; (iv) facilidade de treinamento e/ou configuração; (v) adaptabilidade a novas situações. A Figura 4.31 traz um esquema dos diversas diversas abordagens do estado da arte descritas neste capítulo.

4.5 Vida artificial

Vida artificial (*Artificial Life* - ALife) é o nome dado à disciplina que estuda a vida natural através da tentativa de recriar fenômenos biológicos em computadores ou outros meios artificiais (LANGTON, 1995). Complementa a abordagem analítica tradicional da biologia com uma abordagem sintética onde, ao invés de estudar os fenômenos biológicos através de ver como funcionam os organismos vivos já constituídos, cria um sistema que se comporta como um organismo vivo (LEVY, 1992; PFEIFER; SCHEIER, 1999). As tentativas de recriar os fenômenos biológicos de maneira artificial podem resultar não só na melhor compreensão teórica dos fenômenos estudados, como também em aplicações práticas dos princípios biológicos nas áreas de robótica, medicina, nanotecnologia e engenharia.

Um dos pioneiros na área de vida artificial foi Karl Sims, que em (SIMS, 1994a, 1994b) utilizou um ambiente tridimensional baseado em física para a evolução de criaturas virtuais. Nestas criaturas, o sistema de controle neural foi evoluído em conjunto com a morfologia, o que é biologicamente plausível, pois segundo Pfeifer e Scheier (1999), na natureza o sistema nervoso evoluiu em conjunto com a morfologia dos seres vivos. O genótipo utilizado é constituído por um grafo direcionado, que determina as conexões entre os diversos segmentos da criatura virtual. A Figura 4.27 mostra algumas criaturas virtuais de Karl Sims evoluídas para caminhar (a função de *fitness* é a distância percorrida).

Quando a morfologia é evoluída em conjunto com o sistema de controle, o espaço de busca cresce exponencialmente, o que dificulta a evolução e exige um maior poder computacional. Para manter a complexidade em níveis aceitáveis, são necessárias restrições no modelo. Nas criaturas de Karl Sims, uma das restrições impostas é o formato dos segmentos – as criaturas são constituídas de corpos rígidos (cubos) de tamanhos variados. Um modelo sem restrições dificilmente irá convergir para uma solução aceitável (PFEIFER; SCHEIER, 1999). Já um modelo com muitas restrições reduzirá os graus de liberdade, e em conseqüência haverá menos variabilidade nas criaturas evoluídas.

Outro trabalho importante na área de vida artificial é descrito em (EGGENBERGER, 1996, 1997), onde o desenvolvimento de criaturas virtuais foi modelado a nível celular: as células se multiplicam e se diferenciam de forma similar ao que ocorre no crescimento dos seres vivos.

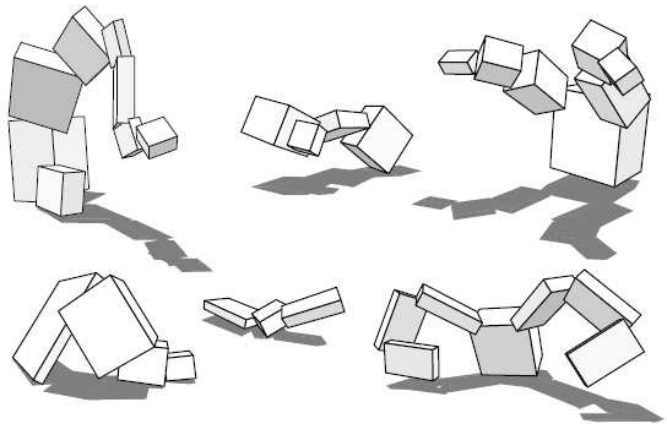
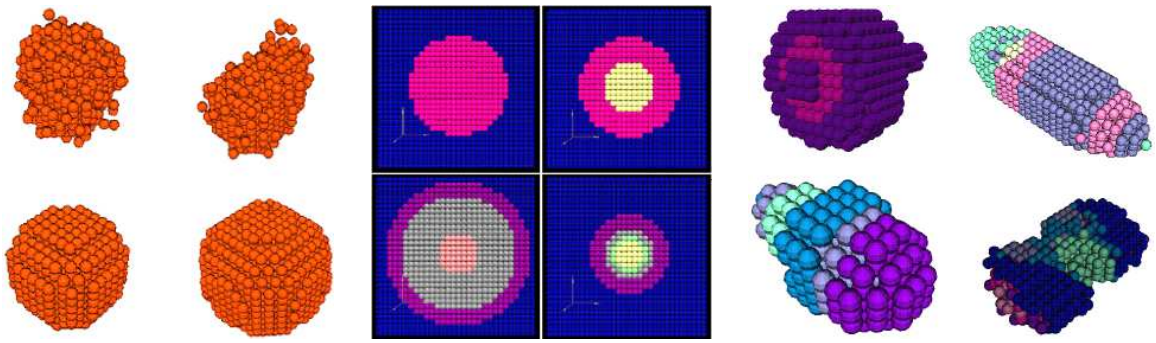


Figura 4.27: Criaturas virtuais de Sims (1994b)

Quando as criaturas virtuais são especificadas neste nível, o espaço de busca, e conseqüentemente a complexidade computacional, se torna muito grande, o que praticamente inviabiliza a evolução de criaturas multi-celulares como os mamíferos e os seres humanos. A Figura 4.28 mostra o processo de evolução celular de Peter Eggenberger. A Figura 4.28(a) mostra o processo de divisão e agrupamento das células, a Figura 4.28(b) mostra a diferenciação celular e a Figura 4.28(c) mostra alguns exemplos de criaturas evoluídas. Pode-se notar que estas criaturas são bastante simples e não possuem muita relação com os seres vivos.



(a) Multiplicação (b) Diferenciação celular (c) Criaturas virtuais

Figura 4.28: Processo de evolução e diferenciação celular (EGGENBERGER, 1997)

Um trabalho na área de vida artificial que foca no deslocamento de criaturas virtuais com pernas é descrito em (NIKOVSKI, 1995). Neste trabalho, foram utilizados Algoritmos Genéticos para a evolução do caminhar em criaturas de seis pernas simuladas (Figura 4.29). Na abordagem utilizada, a dinâmica do caminhar foi definida através de um conjunto de equações diferenciais de primeira ordem, e os parâmetros de cada uma das equações utilizadas foram evoluídos através do uso de Algoritmos Genéticos. O tipo de caminhar utilizado foi o *metachronal wave*, comum em muitos insetos, que é um tipo de caminhar bastante econômico em termos de energia, pois apenas uma das patas é afastada do chão a cada instante de tempo.

Outro trabalho desenvolvido recentemente é o de Bonnasse-Gahot (2005), no qual foram utilizadas Redes Neurais recorrentes para o controle das articulações das criaturas virtuais. Para a otimização dos pesos da ANN, foram utilizados Algoritmos Genéticos, e o ambiente virtual

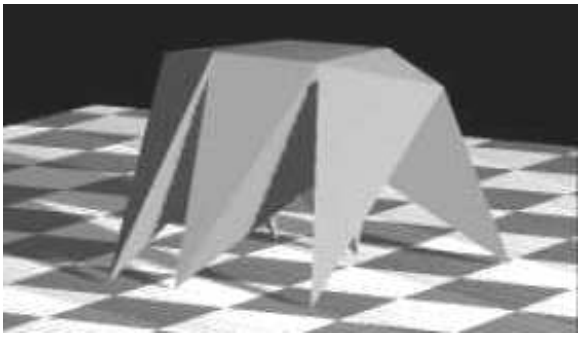


Figura 4.29: Exemplo de criatura virtual utilizada em (NIKOVSKI, 1995)

foi construído utilizando a biblioteca ODE. A Figura 4.30 mostra uma criatura artificial utilizada nos experimentos, que se desloca de forma similar às serpentes.

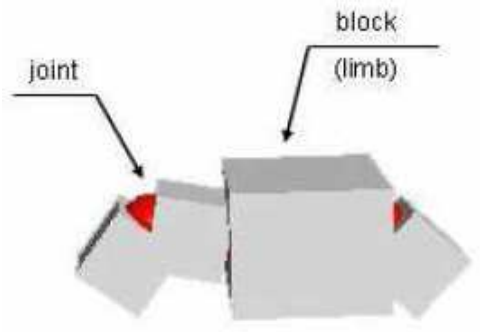


Figura 4.30: Exemplo de criatura virtual artificial (BONNASSE-GAHOT, 2005)

4.6 Considerações finais

Neste capítulo foram descritas diversas técnicas do estado da arte na área de controle do caminhar de robôs com pernas e na área de vida artificial. Ao longo do capítulo foram mostrados diversos modelos de robôs e criaturas, e foram descritas as técnicas de Aprendizado de Máquina utilizadas em cada uma das abordagens.

A partir desta pesquisa do estado-da-arte, foi possível a elaboração do modelo de simulação e sua implementação, conforme é apresentado no próximo capítulo. Além da elaboração do modelo, foi desenvolvido um ambiente virtual de simulação tridimensional baseado em física para os robôs com pernas. O objetivo de se desenvolver o ambiente de simulação é possibilitar o desenvolvimento de pesquisas mais abrangentes sobre diversos modelos de controle de robôs com pernas, permitindo assim uma visão mais ampla dos problemas relacionados com a implementação e controle destes tipos de robôs.

Além do ambiente virtual de simulação, também foram propostos e implementados diferentes modelos de controle de robôs com pernas, baseados em ferramentas que se utilizam principalmente de Algoritmos Genéticos, Redes Neurais Artificiais e autômatos finitos (máquinas de estados), de modo a verificar a robustez destas técnicas quando aplicadas à configuração automática do caminhar de robôs com pernas. Estes modelos de controle propostos e implementados são descritos no próximo capítulo.

Nome do robô / simulador	Citação	Leg	GL	Robô	Controle	ML
Mechanical Horse	(RYGG, 1893)	4	3	P	Engrenagens	- // -
"Phoney Pony"	(MCGHEE; FRANK, 1968)	4	2	R	Autômato	- // -
GE Walking Truck	(MOSHER, 1968)	4	3	R	Manual	- // -
Monopod	(RAIBERT, 1984)	1	3	R	Autômato	- // -
Dante II	(LEMONICK, 1994)	8	2	R	Autômato	- // -
Nonaped / ODE	(ZYKOV; BONGARD; LIPSON, 2004)	9	2	R/S	Autômato	GA
Lynxmotion Hexapod II	(TOTH; PARKER, 2003)	6	2	R	CGA	CGA
HRex	(WEINGARTEN et al., 2004)	6	1	R	Autômato	*Nelder-Mead*
Genghis-II	(PORTA, 2000)	6	2	R	Autômato	*Q-Learning*
Dynamechs	(BUSCH et al., 2002)	1-6	2	S	GP	GP
DynaMechs	(REEVE; HALLAM, 2005)	4	2	S	ANN	GA
Taitan-VIII	(SHIMADA et al., 2002)	4	4	R	CPG	ANN
Igor	(HOLLAND; SNAITH, 1992)	4	2	R	ANN	*Q-Lerning*
ODE	(JACOB; POLANI; NEHANIV, 2005)	4	2	S	CPG	RL
MOBILE	(MURAO; TAMAKI; KITAMURA, 2001)	4	3	R/S	Funções senoidais	RL
Aibo	(GOLUBOVIC; HU, 2002, 2003)	4	3	R	Retângulo	GA
Aibo	(KOHL; STONE, 2004a, 2004b)	4	3	R	Meia-elipse	GA, RL
Aibo	(ROFER, 2005)	4	3	R	Meia-elipse	GA
Aibo	(CHERNOVA;VELOSO, 2004)	4	3	R	Funções cíclicas	GA
Elvina / ODE	(WOLFF; NORDIN, 2002, 2003a, 2003b)	2	5	R/S	VRM	GP
GuRoo / Dynamechs	(WYETH; KEE; YIK, 2003)	2	6	R/S	ZMP / *key frames*	GA
SD/FAST	(ENDO; MAENO; KITANO, 2003)	2	5	R/S	ANN	GA
HOAP-1	(ASADA et al., 2003)	2	5	R	CPG / ANN	*Q-Learning*
Toddler	(TEDRAKE; ZHANG; SEUNG, 2004)	2	3	R	PDW	RL
ODE	(HITOMI et al., 2005)	2	2	S	PDW	RL
ODE	(VAUGHAN, 2003a, 2003b)	2	5	S	PDW / ANN	GA

Figura 4.31: Trabalhos do estado da arte pesquisados

5 Modelo proposto

Neste capítulo são descritas as diversas técnicas de controle propostas para o problema em questão. Para que fosse possível avaliar as diferentes técnicas, foi desenvolvido um modelo, chamado de LegGen (HEINEN; OSÓRIO, 2006b, 2006c, 2006d, 2006f, 2006i, 2006j, 2007a, 2007b) (*Legged Gait Generator*), que realiza a simulação do caminhar de robôs com pernas. Este modelo foi implementado através de um protótipo, que utiliza as bibliotecas ODE e GAlib, descritas anteriormente.

O simulador LegGen permite que sejam utilizados diversos modelos de robôs, definidos através de arquivos texto, e diversas estratégias de controle, que serão descritas na Seção 5.2. Assim, é possível realizar um estudo comparativo entre as diversas técnicas de controle e entre diferentes modelos de robôs. Além disto, o simulador LegGen permite inclusive a evolução dos sistemas de controle em conjunto com a morfologia do robô.

Este capítulo está estruturado da seguinte forma: a Seção 5.1 descreve o simulador LegGen, seus diversos módulos e a relação entre eles; a Seção 5.2 descreve as estratégias de controle propostas; a Seção 5.3 descreve o Algoritmo Genético utilizado, bem como as funções de *fitness* propostas; a Seção 5.4 descreve os principais modelos de robôs utilizados; e por último, a Seção 5.5 descreve a evolução do controle em conjunto com a morfologia do robô.

5.1 Simulador LegGen

O simulador LegGen é composto de diversos módulos, mostrados na Figura 5.1. Cada um destes módulos é fracamente acoplado com os demais, o que permite, que um deles seja alterado sem a necessidade de se modificar os outros módulos. As próximas seções descrevem as características de cada um dos módulos da Figura 5.1 em detalhes.

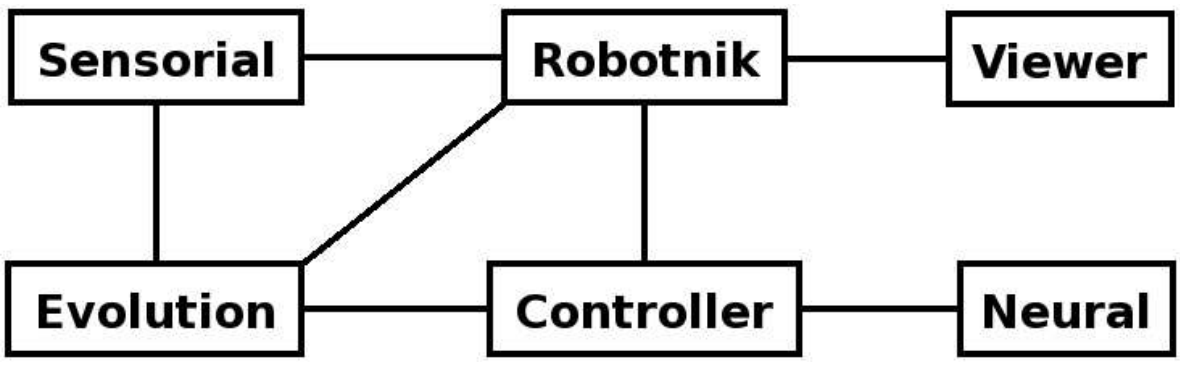


Figura 5.1: Módulos do LegGen

5.1.1 Robotnik

Este módulo é responsável pela criação do robô, bem como do ambiente virtual, através do uso da biblioteca ODE. Este módulo atua da seguinte forma:

1. As especificações do robô (número de pernas e segmentos, tamanhos, limites das juntas, etc.) são lidas a partir de um arquivo texto (Seção B.2 do Anexo B);
2. A partir destas especificações, o robô é criado (corpos e juntas) no ambiente virtual;
3. Os parâmetros de controle, que dependem da estratégia adotada, são carregados;
4. A simulação física do caminhar é realizada, e ao final o módulo Sensorial é chamado para calcular os estimadores.

5.1.2 Sensorial

Este módulo é responsável pela leitura dos dados sensoriais durante a simulação, e pelo cálculo dos estimadores ao final desta. As principais informações coletadas (Seção 5.3.1) são:

- A distância percorrida pelo robô (Fórmula 5.17);
- Taxa de instabilidade, estimada através de um giroscópio simulado (Fórmula 5.20);
- Índice dos *bumpers* (Fórmula 5.19);

A Seção 5.3.1 descreve estas informações em maiores detalhes, bem como a composição das mesmas na função de *fitness*.

5.1.3 Controller

Este módulo é responsável pelo controle das juntas do robô durante o caminhar. Ele é composto por diversos sub-módulos, como mostra a Figura 5.2.

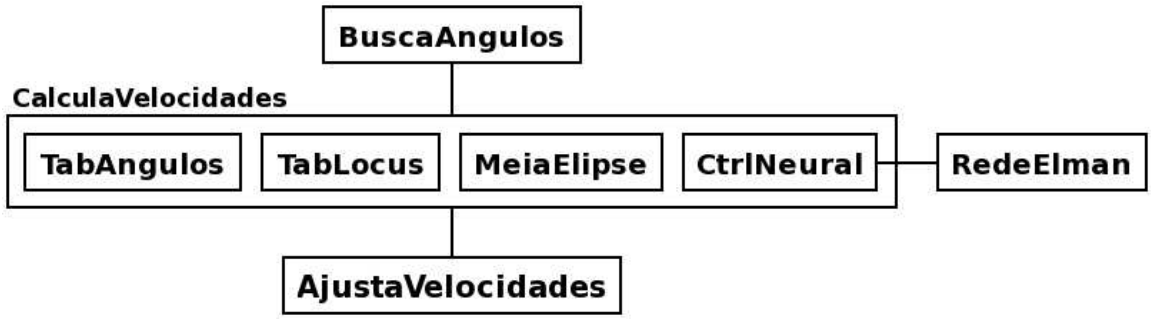


Figura 5.2: Sub-módulos de controle do LegGen

As funções de cada um destes sub-módulos são:

- BuscaAngulos: lê os ângulos atuais de cada uma das juntas do robô;
- CalculaVelocidades: calcula as velocidades a serem aplicadas nos motores angulares, de acordo com a estratégia de controle adotada (Seção 5.2);
- AjustaVelocidades: ajusta as velocidades dos motores angulares do robô;
- RedeElman: este sub-módulo, que implementa uma Rede Neural do tipo Elman (vide Seção 5.1.6), só é utilizado pelo controlador neural.

A Seção 5.2 descreve as quatro estratégias de controle utilizadas pelo simulador LegGen, que correspondem aos quatro sub-módulos de controle (TabAngulos, TabLocus, MeiaElipse e CtrlNeural) da Figura 5.2.

5.1.4 Evolution

Este módulo é responsável pela evolução dos parâmetros do controlador adotado através do uso de Algoritmos Genéticos. Sua implementação é baseada na biblioteca GAlib, descrita na Seção 3.1.4. O processo avaliação dos indivíduos ocorre da seguinte forma:

1. O módulo Robotnik cria o robô no ambiente virtual;
2. Os parâmetros do GA são lidos a partir de um arquivo (Seção B.1 do Anexo B);
3. A evolução é realizada utilizando a biblioteca GAlib e os parâmetros lidos anteriormente;
4. Ao final da evolução, os parâmetros do melhor indivíduo são salvos em um arquivo (Seção B.3 do Anexo B).

O tipo de Algoritmo Genético utilizado, bem como as funções de *fitness* adotadas, são descritos detalhadamente na Seção 5.3.

5.1.5 Viewer

Este módulo é responsável pela visualização dos resultados em um ambiente gráfico tridimensional. Para isto, é utilizado o visualizador padrão da biblioteca ODE, chamado de *drawstuff*. Para acelerar a evolução, este módulo é utilizado apenas na visualização das soluções finais, e não durante a evolução. Isto permite que o relógio de simulação seja acelerado em relação ao tempo real.

5.1.6 Neural

Este módulo, que é utilizado apenas pelo controlador neural, implementa a Rede Neural do tipo Elman que é utilizada no controle das articulações do robô. A Seção 5.2.5 descreve este módulo em maiores detalhes.

Como foi descrito anteriormente, o simulador LegGen implementa quatro estratégias de controle, e todas elas tem seus parâmetros evoluídos através da Algoritmos Genéticos. A Próxima seção descreve as estratégias de controle implementadas.

5.2 Estratégias de controle

Esta seção descreve várias estratégias possíveis de serem utilizadas no controle do caminhar de robôs com pernas. A primeira estratégia, descrita na Subseção 5.2.1, serve apenas como referência, de modo que ela não foi implementada na protótipo por apresentar muitas desvantagens em relação as demais. As Subseções 5.2.2, 5.2.3, 5.2.4 e 5.2.5 descrevem, respectivamente, as quatro estratégias de controle que foram implementadas no protótipo: TabAngulos, TabLocus, MeiaElipse e CtrlNeural.

5.2.1 Autômato baseado em uma tabela de estados

Esta estratégia de controle consiste na utilização de um autômato (máquina de estados) representado por uma tabela que determina, para cada estado, a duração do mesmo e os valores de ativação dos motores angulares das juntas do robô. A Tabela 5.1 ilustra um exemplo de autômato de controle para um robô com oito juntas. Nesta tabela, cada linha representa um estado do autômato (Estados 01, 02, 03 e 04). A segunda coluna (Tempo) representa os tempos de duração dos estados em segundos, e as demais colunas (J1 a J8) representam os valores de ativação dos motores angulares de cada uma das juntas do robô.

Tabela 5.1: Autômato representado por uma tabela de estados

Estado	Tempo	J1	J2	J3	J4	J5	J6	J7	J8
01	1,45s	-0,715	0,541	0,174	-0,401	0,907	-0,506	-0,401	0,907
02	1,65s	-0,471	1,483	-1,012	-0,558	0,785	-0,226	-0,558	0,785
03	1,20s	-0,401	0,907	-0,506	-0,715	0,541	0,174	-0,715	0,541
04	1,40s	-0,558	0,785	-0,226	-0,471	1,483	-1,012	-0,471	1,483

No exemplo da Tabela 5.1, a configuração do caminhar consiste em determinar os valores mais adequados para as diversas células da tabela. Se o robô for simétrico, como é o caso da maioria dos robôs existentes (e de grande parte dos seres vivos também), apenas metade da tabela precisa ser preenchida, pois a outra metade é obtida facilmente trocando-se os valores das juntas do lado esquerdo com os valores das juntas do lado direito. Pode-se observar que, mesmo assim, ainda existem muitos parâmetros (campos na tabela) a serem configurados, e é isso que torna difícil a configuração manual do caminhar em robôs com pernas.

A principal desvantagem desta estratégia de controle é que nenhum *feedback* do mundo externo é levado em conta no planejamento das ações, ou seja, é uma estratégia de controle puramente deliberativa (HEINEN, 1999, 2002), que atua em malha aberta (*open loop*) sem o uso de sensores. Em um robô real, esta estratégia se mostra pouco robusta, pois pequenas diferenças no comportamento dos atuadores, no nível de carga da bateria, a fricção dos componentes ou obstáculos externos podem alterar a resposta das juntas, fazendo com que o robô não se comporte da forma esperada (BOEING; HANHAM; BRäUNL, 2004).

5.2.2 Autômato baseado em uma tabela de ângulos

Uma abordagem alternativa (TabAngulos), implementada no protótipo, consiste na utilização de um autômato similar ao anterior, mas no qual se define, ao invés dos tempos e velocidades, o ângulo desejado ao final do estado para cada uma das juntas do robô. Nesta abordagem, também utilizada em (BONGARD; PFEIFER, 2002), o controlador continuamente realiza a leitura dos ângulos de cada uma das juntas, para verificar se elas já atingiram os valores desejados. Em robôs reais, os ângulos das juntas podem ser obtidos através da leitura de sensores (*encoders*) instalados nas mesmas (DUDEK; JENKIN, 2000; BEKEY, 2005).

Na abordagem descrita acima, o controle do caminhar é realizado da seguinte forma: inicialmente o controlador verifica se as juntas já atingiram os ângulos desejados. As juntas que não tiverem atingido são movimentadas no sentido correspondente (os motores são ativados), e quando todas as juntas tiverem atingido os seus respectivos ângulos (com uma pequena margem

de tolerância), o autômato passa para o próximo estado. Se alguma das juntas não tiver atingido o ângulo desejado após um limite de tempo, o estado é avançado independente desta. Em um robô real, esta situação pode ocorrer devido a um dano no robô ou a presença de obstáculos no ambiente.

Para que haja sincronia nos movimentos, é importante que todas as juntas atinjam os ângulos desejados praticamente ao mesmo tempo, o que é possível com a aplicação de uma velocidade angular específica para cada uma das juntas, calculada através da fórmula:

$$V_{ij} = Vr_j(\alpha_{ij} - \alpha_{ij-1}), \tag{5.1}$$

onde V_{ij} é a velocidade aplicada ao motor da junta i no estado j, α_{ij} é o ângulo da junta i no estado j, α_{ij-1} é o ângulo da junta i ao final do estado anterior $(j-1)$, e Vr_j é a velocidade referencial do estado j, utilizada para controlar a velocidade do conjunto. Esta velocidade referencial deve ser definida na tabela de controle para cada um dos estados, e também para um estado extra, chamado de estado 0, no qual o robô precisa passar da posição original para a posição definida pelos ângulos do primeiro estado do autômato.

A Tabela 5.2 ilustra um exemplo de autômato de controle para um robô com oito juntas (a velocidade referencial do estado 0 foi omitida por questões de simplificação). Nesta tabela, cada linha representa um estado do autômato (Estados 01, 02, 03 e 04). A segunda coluna da tabela representa a velocidade referencial Vr_j, e as demais colunas (α_1 a α_8) representam os ângulos desejados ao final do estado para cada uma das juntas do robô. Neste trabalho, utilizou-se Algoritmos Genéticos para a configuração automática dos diversos campos da Tabela 5.2, implementados através da biblioteca GAlib.

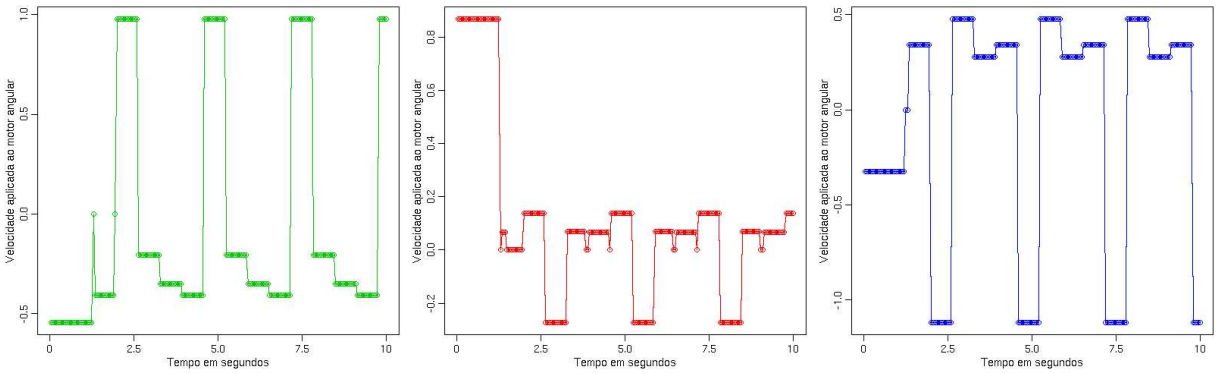
Tabela 5.2: Autômato representado por uma tabela de ângulos

Estado	Vr_j	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8
01	2,985	1,262	-0,418	1,359	0,210	1,359	0,210	1,262	-0,418
02	1,654	0,625	0,261	1,378	-0,698	1,378	-0,698	0,625	0,261
03	2,432	1,359	0,210	1,262	-0,418	1,262	-0,418	1,359	0,210
04	1,129	1,378	-0,698	0,625	0,261	0,625	0,261	1,378	-0,698

Para reduzir o espaço de busca, foram utilizados alelos do tipo real, que somente geram valores dentro dos limites de atuação de cada uma das juntas. Estes alelos foram discretizados em intervalos de 2,5°. Com relação às velocidades relativas, os alelos utilizados geram valores dentro do intervalo [0,25; 1,75]. Para simplificar a busca no espaço de estados, duas estratégias foram utilizadas:

- Os robôs caminham utilizando o passo trote, no qual duas pernas diagonalmente opostas se movimentam ao mesmo tempo (a perna frontal-esquerda se move em conjunto com a perna traseira-direita, por exemplo). Assim, se todas as pernas do robô forem iguais, como é o caso dos robôs modelados, os mesmos ângulos podem ser aplicados em pernas diagonalmente opostas, o que reduz o número de parâmetros a serem otimizados;
- Como os robôs são simétricos (as juntas do lado direito são idênticas as do lado esquerdo), o GA precisa otimizar somente os ângulos de metade dos estados do autômato. A outra metade da tabela é preenchida trocando-se os valores das juntas do lado direito com os do lado esquerdo, e vice-versa. Seguindo este princípio, somente a metade das velocidades referenciais precisam ser otimizadas.

Assim, um autômato como o da Tabela 5.2, que possui 37 parâmetros, pode ser otimizado por um GA de apenas 11 genes, o que simplifica a busca no espaço de estados. Em relação aos robôs de seis pernas, estes possuem um passo conhecido como *tripod* (tripé), que é equivalente ao trote (apenas a metade das patas fica em contato com o chão), e apresenta as mesmas vantagens deste em termos de redução do espaço de estados. A Figura 5.3 mostra valores de ativação (velocidade dos motores angulares) típicos desta estratégia de controle.



(a) Tornozelo (b) Joelho (c) Virilha

Figura 5.3: Velocidades aplicadas aos motores angulares

Essa estratégia de controle se mostra um pouco mais robusta do que a anterior, pois apesar de ainda ser uma estratégia deliberativa (HEINEN, 1999, 2002), ela utiliza *feedback* proprioceptivo (*encoders*) no controle das juntas do robô.

5.2.3 Autômato baseado em uma tabela de posições

Outra abordagem para o controle do caminhar, conhecida como *locus based gait* (WYETH; KEE; YIK, 2003) (TabLocus), consiste na utilização de um autômato no qual é determinado, para cada estado, a posição em que cada um dos *endpoints* ("pés", "patas", extremidades inferiores das pernas) devem se encontrar, ao final do estado, em relação aos eixos x, y e z. Esta abordagem é útil principalmente em robôs bípedes, onde o número de graus de liberdade por extremidade (número de juntas) costuma superar o número de dimensões. Já em robôs com menos de quatro graus de liberdade por perna, esta abordagem não é tão eficiente, pois não haverá uma redução do espaço de estados (número de parâmetros).

Como os robôs utilizados neste trabalho tem graus de liberdade apenas no plano *sagittal* (eixo z), as posições dos *endpoints* precisam ser determinadas apenas nos eixos x e y, o que torna esta abordagem interessante em robôs com mais de dois graus de liberdade por perna, que é o caso dos robôs HexaL3J (Figura 5.8(a)) e do TetraL3J (Figura 5.8(b)), descritos na Seção 5.4. A Tabela 5.3 mostra um exemplo de autômato de controle para um robô de quatro pernas. As simplificações utilizadas na estratégia anterior também podem ser utilizadas aqui, ou seja, devido ao passo trote e a simetria dos robôs, apenas $1/4$ das posições da Tabela 5.3 precisam ser otimizadas.

É importante ressaltar que os valores da Tabela 5.3 representam posições, e não valores de junta (ângulos e/ou velocidades). Para se determinar os ângulos desejados de cada junta, é ne-

Tabela 5.3: Autômato representado por uma tabela de posições

		Perna 1		Perna 2		Perna 3		Perna 4	
Estado	Tempo	x	y	x	y	x	y	x	y
01	1,15s	1,412	-3,325	1,750	-2,854	-0,350	-3,325	0,713	-2,821
02	0,77s	-0,175	-2,275	0,712	-1,925	-0,175	-2,275	0,712	-1,925
03	1,22s	0,525	-3,325	-0,175	-2,864	1,412	-3,325	1,750	-2,854
04	0,92s	-0,350	-3,325	0,713	-2,821	0,525	-3,325	-0,175	-2,864

cessário calcular a cinemática inversa. Este cálculo não precisa ser realizado continuamente em tempo real, pois para cada genoma, os ângulos podem ser calculados antes da simulação física e gravados em uma tabela similar a da abordagem anterior (Tabela 5.2). Na implementação realizada, calculou-se algebricamente a cinemática direta dos quatro modelos robôs com o software R[1] e utilizou-se a implementação descrita em (PRESS et al., 1992) do método de Powell (BRENT, 1973) (*Powell's Direction Set*) para calcular a cinemática inversa.

Esse cálculo da cinemática inversa ocorre da seguinte forma: o método de Powell procura minimizar o valor de retorno da função fp, que é chamada iterativamente por este método. Ela recebe como entrada um vetor v_α com os ângulos das juntas da perna para a qual se está calculando a cinemática inversa (perna atual), e retorna um valor numérico calculado pela equação:

$$fp = (x_d - x_o)^2 + (y_d - y_o)^2 + (\alpha_o + \pi/2)^2 + r_j \tag{5.2}$$

onde x_d e y_d são as posições desejadas para o *endpoint* nos eixos x e y, e $\pi/2$ é o ângulo que torna o *endpoint* paralelo ao corpo do robô. x_o e y_o são as posições do *endpoint* obtidas com os ângulos atuais (v_α), calculadas através das equações:

$$x_o = \sum_{i=1}^{n} \cos\left(\sum_{j=1}^{i} \alpha_j\right) s_i \tag{5.3}$$

$$y_o = \sum_{i=1}^{n} \sin\left(\sum_{j=1}^{i} \alpha_j\right) s_i \tag{5.4}$$

onde n é o número de juntas da perna atual, α_j é o ângulo atual da junta j (recebido do vetor v_α) e s_i é o comprimento do segmento da perna que se inicia na junta i (nos robôs modelados, o número de partes por perna é igual ao número de juntas). Na Fórmula 5.2, α_o é o ângulo do *endpoint* em relação à origem, calculado através da fórmula:

$$\alpha_o = \sum_{i=1}^{n} \alpha_i \tag{5.5}$$

onde α_i é o ângulo atual recebido do vetor v_α. A variável rj, é calculada através das equações:

$$r_j = \sum_{i=1}^{n} \begin{cases} (\alpha_i - \alpha_{i_{min}})^2 & \text{se} \quad \alpha_i < \alpha_{i_{min}} \\ (\alpha_i - \alpha_{i_{max}})^2 & \text{se} \quad \alpha_i > \alpha_{i_{max}} \\ 0 & \text{se} \quad \alpha_{i_{min}} \leq \alpha_i \leq \alpha_{i_{max}} \end{cases} \tag{5.6}$$

onde $\alpha_{i_{min}}$ é o ângulo mínimo e $\alpha_{i_{max}}$ é o ângulo máximo da junta i. A Fórmula 5.6 serve para restringir os valores dos ângulos nos intervalos máximos e mínimos de cada junta.

[1] R Foundation for Statistical Computing – http://www.r-project.org/

Assim, o método de Powell recebe um vetor v_α com os ângulos atuais da perna, e vai alterando estes valores até que o erro residual (valor de retorno de fp) seja menor que um determinado limite $ftol$, ou até que o número de iterações ultrapasse um valor máximo $iter$. Nos experimentos realizados, o valor utilizado em $ftol$ foi $1e-6$ e o valor de $iter$ foi 200. Ao final das iterações, o método de Powell irá retornar os ângulos que fazem com que o *endpoint* fique o mais próximo possível da posição desejada, e o mais paralelo ao solo possível, levando também em conta as restrições das juntas. Se uma determinada posição for impossível de ser atingida, isto pode ser detectado através do valor de retorno fp na última iteração, pois neste caso o valor será maior que $ftol$.

No simulador LegGen, os diversos campos da Tabela 5.3 também foram configurados automaticamente através do uso de Algoritmos Genéticos. Para tornar a busca no espaço de estados mais eficiente, utilizou-se alelos limitando os valores gerados pelo Algoritmo Genético, de modo a evitar a geração de coordenadas que estivessem fora da área de alcance dos *endpoints*. Assim, os alelos foram calculados através das seguintes equações:

$$a_x = \left\{ -\frac{C}{2}; +\frac{C}{2} \right\} \tag{5.7}$$

$$a_y = \left\{ -\frac{C}{20}; -\frac{C}{2} \right\} \tag{5.8}$$

onde C é o comprimento da perna do robô, e a_x e a_y são os alelos, que na implementação realizada foram discretizados com intervalos de 0,01.

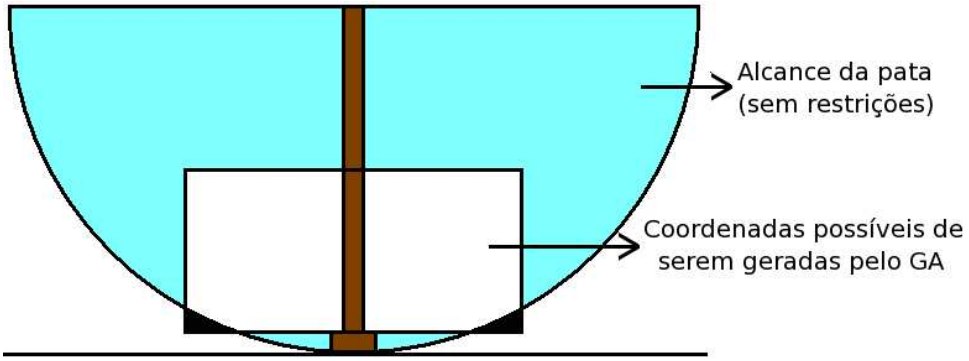


Figura 5.4: Relação entre as coordenadas geradas e as possíveis

Os valores de a_x e a_y são relativos ao ponto de origem da perna (ponto que prende ela ao corpo). Percebe-se que estas equações definem uma área no formato de um retângulo, dentro do qual podem ser gerados pontos (posições desejadas em x e y). Como pode ser visto pela Figura 5.4, nem todos os pontos possíveis de serem gerados são válidos. Nestes casos, os indivíduos com um fp maior que determinado limite podem ser previamente descartados, sem a necessidade de se realizar a simulação física. Os valores de ativação típicos desta estratégia de controle são similares aos da Figura 5.3.

5.2.4 Controle baseado em funções cíclicas

Outra forma de se controlar o movimento das pernas de um robô é modelar o deslocamento dos *endpoints* através de uma função cíclica (MeiaElipse), como por exemplo uma meia-elipse

(Figura 4.17(b)). Neste caso, não é utilizado um autômato de controle, pois os movimentos são controlados através de diversas equações. Para a geração da meia-elipse, as posições dos *endpoints* no eixo x são obtidas através da fórmula:

$$x(t) = \begin{cases} x_0 - \frac{l}{2} + \frac{2lt}{T} & \text{se } t < T/2 \\ x_0 + \frac{l\cos(\alpha)}{2} & \text{se } t \geq T/2 \end{cases}, \tag{5.9}$$

onde $x(t)$ é a posição do *endpoint* em x no tempo t, x_0 é a origem (centro) da elipse no eixo x, T é o tempo de um ciclo (um passo completo), t é o tempo atual, l é a largura da elipse, e α é o ângulo do *endpoint* em relação ao centro da elipse, calculado através da fórmula:

$$\alpha = \frac{2\pi}{T}\left(t - \frac{T}{2}\right). \tag{5.10}$$

As posições do *endpoint* em relação ao eixo y são obtidas através da fórmula:

$$y(t) = \begin{cases} y_0 & \text{se } t < T/2 \\ y_0 + h\sin(\alpha) & \text{se } t \geq T/2 \end{cases}, \tag{5.11}$$

onde $y(t)$ é a posição do *endpoint* em y no tempo t, y_0 é a origem (centro) da elipse no eixo y, e h é a altura da elipse. Quando $t >= T$, t é setado para 0 e um novo passo se inicia.

Nos robôs modelados (Figura 5.8), a posição do centro da meia-elipse em z não precisa ser calculada, pois o robô possui movimentos apenas no plano *sagittal*. Assim, como os robôs são simétricos, cada par de pernas (esquerda e direita) pode utilizar os mesmos parâmetros nas elipse, o que reduz ainda mais a busca no espaço de estados. A cinemática inversa foi novamente calculada utilizando o método de Powell, e após a obtenção dos ângulos desejados, a velocidade aplicada a cada um dos motores angulares é calculada através da Fórmula 5.1, descrita anteriormente.

Com o uso da meia-elipse, os únicos parâmetros que precisam ser otimizados são a posição do centro da elipse de cada perna em relação aos eixos x e y, a altura e a largura de cada elipse, o tempo de duração total do ciclo e tempo de duração da parte em que a perna está em contato com o solo. A Tabela 5.4 mostra um exemplo de tabela de configuração da meia-elipse. As posições x_0 e y_0 são relativas à origem da perna do robô (ponto de ligação com o corpo), e Δt é o passo de simulação física utilizado nos experimentos (50 milisegundos).

Tabela 5.4: Parâmetros da elipse

Parâmetro	**Valor**
T	4,00 segundos
Δt	0,05 segundos
x_0	3,50 centímetros
y_0	-30,00 centímetros
l	15,00 centímetros
h	8,50 centímetros

Para a otimização dos parâmetros da meia elipse, foram utilizados Algoritmos Genéticos com alelos do tipo real, de forma a evitar que fossem geradas coordenadas impossíveis de serem

atingidas pelos *endpoints*. Os intervalos utilizados nos alelos são os seguintes:

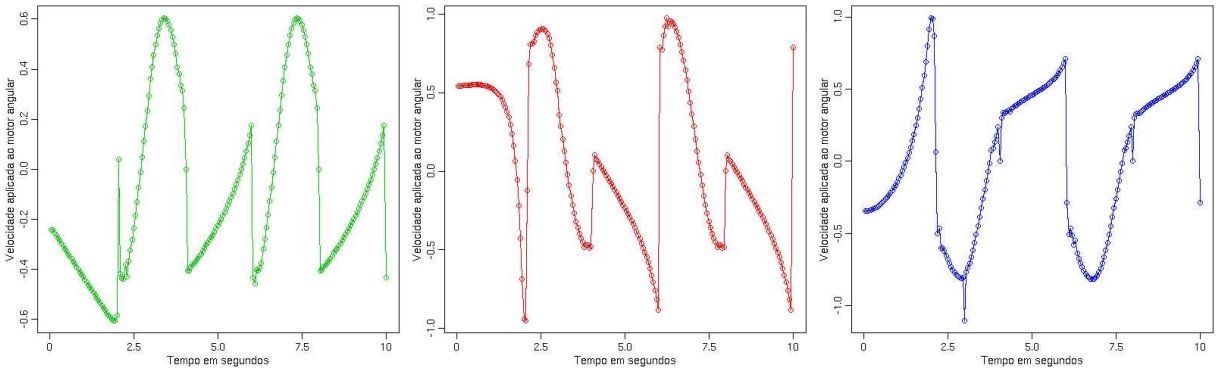
$$\begin{aligned} T &= \{\, 0; 10 \,\} \\ x_0 &= \{\, -C/2; C/2 \,\} \\ y_0 &= \{\, -C; -C/2 \,\} \\ h &= \{\, 0; C/2 \,\} \\ l &= \{\, 0; 3C/2 \,\} \end{aligned} \tag{5.12}$$

onde C é o comprimento da perna do robô. Os alelos utilizados não foram discretizados, de forma que podem assumir qualquer valor real dentro da faixa especificada.

A principal vantagem de se utilizar uma função cíclica para o controle das juntas de um robô é que ela reduz a busca no espaço de estados, facilitando o aprendizado. As principais desvantagens desta abordagem são:

- A meia-elipse restringe os movimentos possíveis, dificultando o caminhar em superfícies irregulares;
- A cinemática inversa precisa ser calculada em tempo real de forma rápida e eficiente.

De fato, o controle baseado na elipse é muito similar ao controle baseado em engranagens mecânicas, utilizado nas primeiras máquinas caminhantes (Seção 4.1) (RAIBERT, 1986). A Figura 5.5 mostra valores de ativação dos motores angulares típicos desta estratégia de controle.



(a) Tornozelo (b) Joelho (c) Virilha

Figura 5.5: Velocidades aplicadas aos motores angulares

5.2.5 Controle neural

Outra forma de se controlar o deslocamento de robôs com pernas é através do uso de Redes Neurais Artificiais (CtrlNeural). Esta abordagem possui uma limitação: não é possível se obter de antemão informações locais para o cálculo do gradiente e a correção dos erros, e isto impede a utilização dos algoritmos de aprendizado supervisionado tradicionais (*back-propagation* e similares). Por isto, é necessário que se utilize alguma outra técnica de Aprendizado de Máquina para a otimização dos pesos da Rede Neural, como os Algoritmos Genéticos ou o Aprendizado por Reforço (RL).

Neste trabalho foram utilizados Algoritmos Genéticos para a evolução dos pesos sinápticos. A vantagem de se utilizar GAs para o ajuste dos pesos é que eles não necessitam de informações locais para a correção dos erros, ou seja, eles não necessitam de uma base com os dados de treinamento. Além disto, o aprendizado ocorre através da interação do robô com o ambiente, o que está de acordo com o princípio da aprendizagem, descrito na Seção 2.7.5. Com relação ao Aprendizado por Reforço, que inicialmente chegou a ser cogitado para ser utilizado (HEINEN, 2006), este foi descartado devido ao fato de que o espaço de busca do problema em questão é continuo e de alta dimensinalidade. Segundo Haykin (2001), a convergência da maioria dos algoritmos de RL só é garantida se o espaço de busca for discreto.

As vantagens de se utilizar uma ANN no controle do caminhar são: (i) elas são mais robustas em relação a situações novas e inesperadas; (ii) possuem um alto grau de generalização; (iii) fornecem uma arquitetura em conformidade com o princípio de processos paralelos fracamente acoplados, descrito na Seção 2.7.3.

Como entradas da ANN, foram utilizados os ângulos atuais das juntas do robô, normalizados entre -1 (α_{mim}) e 1 (α_{max}). Na saída da ANN, são obtidos os ângulos desejados para as juntas no instante $t+1$, normalizados entre -1 e 1. Após alguns testes preliminares, optou-se por utilizar as Redes Neurais recorrentes do tipo Elman (ELMAN, 1990), que são Redes Neurais do tipo *multi layer perceptron* (MLP) que possuem conexões de realimentação (*feedback*) na camada oculta. Estas conexões permitem que as redes de Elman aprendam a reconhecer e gerar padrões temporais. A Figura 5.6 mostra um diagrama das redes de Elman. A função de ativação utilizada foi a tangente hiperbólica.

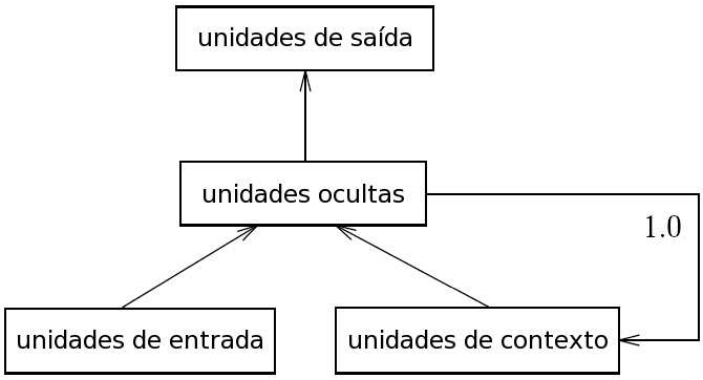


Figura 5.6: Diagrama das redes de Elman

Uma Rede Neural para o controle do robô TetraL3J (Figura 5.8(b)), por exemplo, possui 4 entradas (metade das juntas devido ao passo trote) e 4 saídas. O número de neurônios ocultos é estipulado durante o aprendizado conforme a necessidade, e o número de unidades de contexto é sempre igual ao número de neurônios ocultos. Nesta estratégia de controle, não foi possível reduzir o número de variáveis devido a simetria do robô, pois ao contrário de um autômato, a ANN utilizada não possui um número finito de estados.

O LegGen permite que o usuário defina o intervalo de valores possíveis para os pesos sinápticos. Nos experimentos realizados, foi utilizado o intervalo $[-1;1]$, que se mostrou bastante adequado para a representação do problema em questão. Na ANN utilizada, a camada de entrada foi totalmente conectada à camada oculta, e a camada oculta foi totalmente conectada à camada de saída. Em adição, cada neurônio da camada oculta foi conectado a si mesmo de forma recorrente. Assim, a ANN utilizada não segue exatamente a arquitetura Elman padrão,

pois nesta arquitetura a camada oculta é totalmente conectada a si mesma de forma recorrente.

O controle neural é realizado da seguinte forma: a cada passo de simulação (50 milisegundos), os ângulos das juntas são normalizados entre -1 e 1 e em seguida são enviados para a Rede Neural. A função utilizada na normalização é a seguinte:

$$\alpha_n = 2\left(\frac{\alpha_a - \alpha_{mim}}{\alpha_{max} - \alpha_{min}}\right) - 1 \tag{5.13}$$

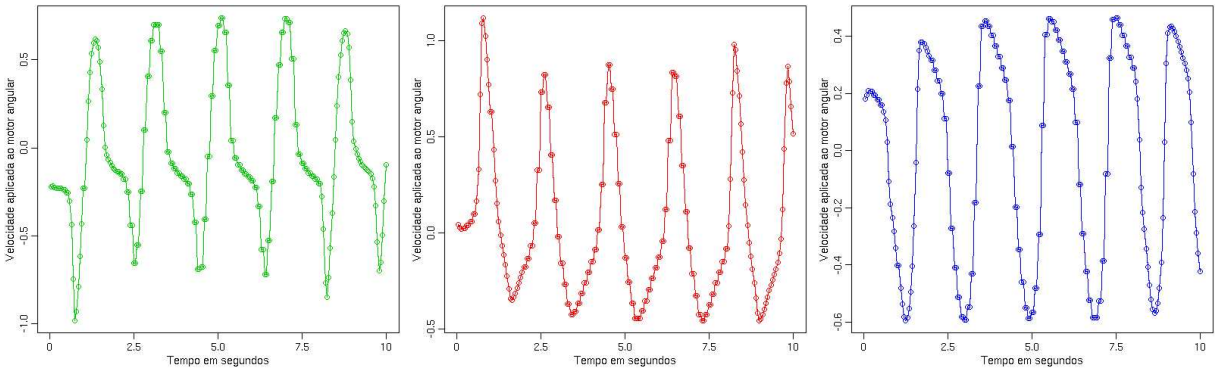
onde α_a é o ângulo atual, α_n é o ângulo normalizado, α_{min} é o limite mínimo e α_{max} é o limite máximo da junta. Após a ativação da ANN, as saídas são restauradas através da fórmula:

$$\alpha_a = \frac{(\alpha_n + 1)(\alpha_{max} - \alpha_{min})}{2} + \alpha_{min} \tag{5.14}$$

obtendo-se assim os ângulos desejados para os motores angulares no instante de tempo $t+1$. A velocidade aplicada a cada um dos motores angulares é calculada através da fórmula:

$$V_{ij} = \frac{\alpha_{ij} - \alpha_{ij-1}}{\Delta t} \tag{5.15}$$

onde Δt é o passo de simulação físico (50 milisegundos nos experimentos realizados). A Figura 5.7 mostra valores de ativação típicos desta estratégia de controle.



(a) Tornozelo (b) Joelho (c) Virilha

Figura 5.7: Velocidades aplicadas aos motores angulares

Durante a fase de aprendizado, o LegGen gera populações de Redes Neurais, e evolui os pesos sinápticos de acordo com o critério de *fitness* utilizado. Na próxima seção será descrito o GA utilizado pelo simulador LegGen na evolução do caminhar dos robôs modelados.

5.3 Algoritmo Genético utilizado

Em todas as estratégias de controle descritas acima, foram utilizados Algoritmos Genéticos para a otimização dos parâmetros de cada uma das técnicas – evolução da tabela de ângulos, da tabela de posições, dos parâmetros da elipse ou dos pesos sinápticos de uma Rede Neural. O

Algoritmo Genético utilizado pelo LegGen foi implementado através da biblioteca de software GAlib, descrita na Seção 3.1.4. Foi utilizado o Algoritmo Genético proposto por De Jong (1975) com populações sobrepostas (*overlapping populations* - *GASteadyStateGA*), e foram adotados genomas do tipo real (números de ponto flutuante – *GARealGenome*). Para se reduzir o espaço de busca, foram utilizados alelos (*GARealAlleleSetArray*) para limitar o conjunto de valores gerados para cada atributo, conforme descrito nas seções anteriores. A Tabela 5.5 descreve os parâmetros da biblioteca GAlib utilizados nas simulações. Uma descrição completa de cada um destes parâmetros pode ser encontrada na documentação da GAlib.

Tabela 5.5: Elementos da GAlib utilizados

Parâmetro	**Tipo**
GA	GASteadyStateGA
Genome	GARealGenome
Alleles	GARealAlleleSetArray
Scaling	GASigmaTruncationScaling
Initializer	GARealGenome::UniformInitializer
Comparator	GARealGenome::ElementComparator
Selector	GASRSSelector
Crossover	GARealGenome::OnePointCrossover
Mutation	GARealGaussianMutator

O tipo de cruzamento (*Crossover*) escolhido foi o cruzamento em um ponto (*OnePointCrossover*), que segundo Mitchell (1996) é menos disruptivo que o cruzamento em dois pontos (*TwoPointCrossover*) e o uniforme (*UniformCrossover*). O método de escala (*Scaling*) do *fitness* utilizado foi o *sigma truncation* (GOLDBERG, 1989) (*GASigmaTruncationScaling*), que permite que o *fitness* assuma valores negativos. O esquema de seleção adotado (*Selector*) foi o *stochastic remainder sampling selector* (*GASRSSelector*), que segundo Goldberg (1989) possui um desempenho superior ao esquema da roleta (*GARouletteWheelSelector*). Com relação ao tamanho do genoma, este é proporcional à estratégia de controle utilizada:

- TabAngulos: 11 genes do tipo real, discretizados em intervalos de $2,5^\circ \times \pi/180$;
- TabLocus: 11 genes do tipo real, discretizados em intervalos de $0,01$;
- MeiaElipse: 6 genes do tipo real não discretizados;
- CtrlNeural (3 ocultos): 40 genes do tipo real, contínuos no intervalo $[-1;1]$;

5.3.1 Função de *fitness*

Para a avaliação do *fitness* de cada um dos indivíduos, procurou-se utilizar uma função que valorizasse não apenas os indivíduos mais velozes, mas também aqueles que se deslocassem de maneira estável. Várias tentativas foram feitas até que se chegasse a uma função de *fitness* satisfatória. Nesta seção serão descritas diversas funções de *fitness* utilizadas pelo LegGen.

A avaliação do *fitness* de cada indivíduo é realizada da seguinte forma:

1. O robô é colocado na orientação e na posição inicial do ambiente virtual;
2. O genoma é lido, e a partir dele é configurado o sistema de controle do robô;
3. A simulação física é realizada por um tempo determinado (sessenta segundos simulados nos experimentos realizados);

4. Durante a simulação física, são capturadas e armazenadas informações sensoriais relativas ao caminhar;
5. O *fitness* é calculado e retornado para a GAlib.

A forma como o sistema de controle é configurado depende da estratégia de controle utilizada (TabAngulos, TabLocus, MeiaElipse ou CtrlNeural). Para o cálculo do *fitness*, foram testadas e utilizadas diversas funções. A primeira função utilizada foi a distância percorrida pelo robô em relação ao eixo x, calculada através da equação:

$$F = D \tag{5.16}$$

onde D é a distância percorrida pelo robô, calculada através da fórmula:

$$D = Px_1 - Px_0 \tag{5.17}$$

onde Px_0 é a posição inicial e Px_1 é a posição final robô em relação ao eixo x. Utilizando esta função de *fitness*, os indivíduos que conseguirem se deslocar para a frente serão recompensados, e os indivíduos que se deslocarem para trás serão punidos, recebendo um *fitness* negativo. Note que o robô deve se deslocar alinhado ao eixo x e em direção a valores crescentes de x.

Esperava-se que, utilizando esta função de *fitness*, os indivíduos selecionados para a reprodução seriam os que apresentassem um caminhar mais estável, pois assim o risco de sofrer quedas seria menor, o que permitiria que eles se deslocassem por mais tempo do que os indivíduos que tombassem. Mas devido ao fato de que nas gerações iniciais praticamente todos os indivíduos sofrem alguma queda, os indivíduos recompensados não eram os que conseguiam parar em pé, mas sim os que conseguiam tombar mais violentamente para frente, e desta forma a equação 5.16 não conduzia à melhor solução.

Para tentar eliminar esses problemas, optou-se por utilizar uma função de *fitness* que levasse em conta informações sensoriais para tornar o aprendizado mais eficiente. Um dos tipos de sensores mais baratos utilizados em robótica são os *bumpers*, que são sensores de pressão, que se instalados embaixo das patas de um robô, podem indicar quando cada uma das patas está em contato com o chão. Assim, resolveu-se simular *bumpers* embaixo de cada uma das patas dos robôs simulados, de forma que fosse possível descobrir, durante o curso da simulação, quantas patas o robô mantinha em contato com o chão a cada instante de tempo. O cálculo do *fitness* foi então realizado através da seguinte equação:

$$F = \frac{D}{1+B} \tag{5.18}$$

onde B (Bumpers) é um índice relacionado com o percentual de tempo que as patas entram em contato com o chão, calculado através da equação:

$$B = \frac{P}{4} \sum_{i=1}^{P} \left(\frac{n_i}{N} - \frac{1}{2} \right)^2 \tag{5.19}$$

onde P é o número de *endpoints* (patas), n_i é a quantidade de amostras sensoriais nas quais o *endpoint i* estava em contato com o solo, e N é o número total de leituras sensoriais realizadas. Nesta função de *fitness*, o valor de B tenderá a zero quando o robô mantiver as patas no chão por aproximadamente 50% do tempo, que é o comportamento desejado durante o caminhar. Já se o robô mantiver todas as patas no chão durante o período de simulação, o valor de B será igual a

1. O mesmo ocorrerá se o robô mantiver todas as patas no ar durante o período de simulação.

Utilizando a Equação 5.18, o aprendizado se tornou bem mais eficiente, mas muitas das soluções obtidas não eram totalmente estáveis - a altura do corpo variava muito durante o caminhar e o robô balançava algumas vezes. Assim, além do *bumpers*, resolveu-se simular sensores do tipo giroscópio, que também podem ser encontrados em alguns tipos de robôs móveis. Durante o caminhar, leituras do giroscópio simulado vão sendo realizadas, e ao final é calculada a taxa de instabilidade do robô G (Gyro) através da fórmula (GOLUBOVIC; HU, 2003):

$$G = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2 + \sum_{i=1}^{N}(y_i - \overline{y})^2 + \sum_{i=1}^{N}(z_i - \overline{z})^2}{N}} \tag{5.20}$$

onde N é o número de amostras coletadas, x_i, y_i e z_i são os dados coletados pelo giroscópio simulado no tempo i, e $\overline{x}$, $\overline{y}$ e $\overline{z}$ são as médias das leituras do giroscópio, calculadas através das equações:

$$\overline{x} = \frac{\sum_{i=1}^{N} x_i}{N}, \quad \overline{y} = \frac{\sum_{i=1}^{N} y_i}{N}, \quad \overline{z} = \frac{\sum_{i=1}^{N} z_i}{N} \tag{5.21}$$

A função de *fitness* F é então calculada através da equação:

$$F = \frac{D}{1 + B + a \times G} \tag{5.22}$$

onde a é uma constante que serve para alterar a influência de G na função de *fitness*. Se a for pequeno, serão preferidas as soluções mais velozes, e se a for grande, as soluções mais estáveis é que receberão destaque. Após vários experimentos, optou-se por utilizar $a = 10$, pois este valor garante um bom compromisso entre a distância percorrida D e a taxa de instabilidade G.

Após a incorporação da taxa de instabilidade G, analisou-se se era possível dispensar as leituras dos *bumpers*, fazendo com que a função de *fitness* ficasse:

$$F = \frac{D}{1 + a \times G} \tag{5.23}$$

Durante a simulação, se um robô afastar as quatro patas do chão ao mesmo tempo por mais de um segundo, a simulação deste indivíduo será imediatamente interrompida, pois é muito provável que este robô tenha sofrido alguma queda, e portanto não há necessidade de continuar a simulação deste indivíduo até o final do tempo estabelecido.

Para a validação das quatro funções de *fitness* propostas (Fórmulas 5.16, 5.18, 5.23 e 5.22), foram realizados diversos experimentos estatisticamente válidos, que permitiram que fosse selecionada a função de *fitness* mais eficiente para a tarefa em questão. Estes experimentos são descritos na Seção 6.1 do Capítulo 6.

5.4 Robôs modelados

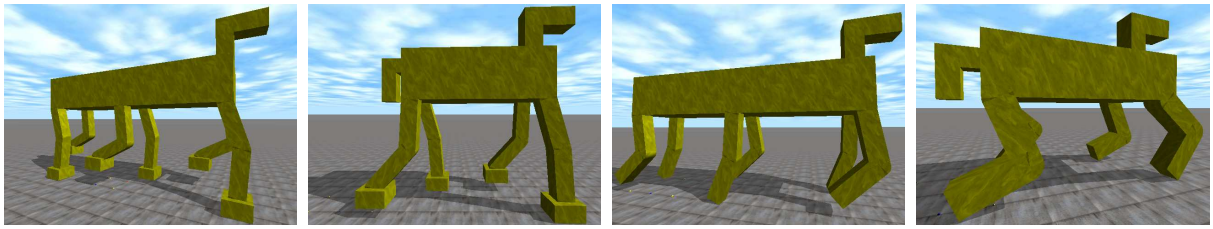
No LegGen, a biblioteca *Open Dynamics Engine* (ODE) foi selecionada para a criação do ambiente virtual e a modelagem dos robôs. Conforme consta em sua documentação, a biblioteca

ODE possui uma complexidade computacional de ordem $O(n^2)$, onde n é o número de corpos presentes no mundo físico simulado. Desta forma, para manter a velocidade da simulação em um nível aceitável, é necessário modelar os corpos presentes no ambiente da forma mais simples possível. Por este motivo, todos os robôs utilizados nos experimentos foram modelados com objetos simples, como retângulos e cilindros, e possuem apenas as articulações necessárias para a tarefa de caminhar. Para manter o projeto dos robôs o mais simples possível, foram utilizadas juntas do tipo *hinge*, que se movimentam apenas em torno do eixo z (plano *sagittal*) em relação ao robô (o mesmo eixo de rotação do nosso joelho), pois as pesquisas realizadas neste trabalho se restringem apenas ao caminhar em linha reta.

Inicialmente foram modelados e testados diversos tipos de robôs, até que se chegou aos quatro modelos principais, mostrados na Figura 5.8. O modelo da Figura 5.8(a), chamado de HexaL3J, possui seis pernas e três partes por perna. As partes que entram em contato com o solo (pés ou patas), chamadas de *endpoints*, são mais largas que o restante das pernas, de modo a dar um maior apoio ao robô. O modelo da Figura 5.8(b), chamado de TetraL3J, é similar ao da Figura 5.8(a), mas possui apenas quatro pernas. Nestes dois robôs, as patas são mantidas paralelas em relação ao corpo do robô, de forma que o suporte das patas em contato com o solo garanta a sustentação do robô. Para que isto ocorra, durante a simulação os ângulos das juntas das patas (α_p) são calculados a cada instante de tempo através da fórmula:

$$\alpha_p = -\sum_{i=1}^{n} \alpha_i, \tag{5.24}$$

onde α_i é o ângulo da junta i e n é o número de juntas da perna.



(a) HexaL3J (b) TetraL3J (c) HexaL2J (d) TetraL2J

Figura 5.8: Modelos de robôs utilizados nas simulações

O modelo de robô da Figura 5.8(c), chamado de HexaL2J, é similar ao da Figura 5.8(a), mas possui apenas duas articulações por perna, ou seja, o robô não possui patas. O modelo da Figura 5.8(d), chamado de TetraL2J, possui quatro pernas e duas articulações por perna. A Tabela 5.6 mostra as dimensões dos robôs em centímetros.

A tabela Tabela 5.7 mostra os limites máximos e mínimos das juntas dos robôs da Figura 5.8 Os robôs de seis pernas (HexaL3J e HexaL2J) possuem todas as pernas idênticas, de forma que as juntas das pernas centrais são idênticas às demais.

Em relação aos robôs bípedes, após um estudo de diversos trabalhos do estado da arte (HEINEN, 2006; HEINEN; OSÓRIO, 2006g), foi decidido que eles ficariam fora do escopo deste trabalho no que diz respeito a simulação e controle implementados no protótipo, podendo vir a ser utilizados em trabalhos futuros.

Tabela 5.6: Dimensões dos robôs simulados

	Corpo			**Coxa**			**Canela**			**Pata**		
Robô	*x*	*y*	*z*	*x*	*y*	*z*	*x*	*y*	*z*	*x*	*y*	*z*
HexaL3J	80,0	15,0	30,0	5,0	15,0	5,0	5,0	15,0	5,0	8,5	5,0	9,0
TetraL3J	45,0	15,0	25,0	5,0	15,0	5,0	5,0	15,0	5,0	8,5	5,0	9,0
HexaL2J	80,0	15,0	30,0	5,0	15,0	5,0	5,0	15,0	5,0	-	-	-
TetraL2J	45,0	15,0	25,0	5,0	15,0	5,0	5,0	15,0	5,0	-	-	-

Tabela 5.7: Limites mínimos e máximos das juntas

	Pernas frontais			**Pernas traseiras**		
Robô	*Quadril*	*Joelho*	*Tornozelo*	*Quadril*	*Joelho*	*Tornozelo*
HexaL3J	[-60°;15°]	[0°;120°]	[-90°;30°]	[-60°;15°]	[0°;120°]	[-90°;30°]
TetraL3J	[-60°;15°]	[0°;120°]	[-90°;30°]	[-60°;15°]	[0°;120°]	[-90°;30°]
HexaL2J	[-60°;15°]	[0°;120°]	–	[-60°;15°]	[0°;120°]	–
TetraL2J	[-60°;15°]	[0°;120°]	–	[-60°;15°]	[0°;120°]	–

5.5 Evolução da morfologia

Segundo Pfeifer e Scheier (1999), na natureza a evolução do controle (sistema nervoso) não ocorre após a morfologia (formato do corpo) estar completa. Pelo contrário, este é um processo que ocorre em conjunto ao longo da evolução. Esta estratégia é muito utilizada na área de vida artificial (SIMS, 1994a, 1994b; EGGENBERGER, 1996, 1997).

Na Seção 5.4 foram descritos quatro modelos de robôs a serem utilizados nos experimentos. Estes robôs foram modelados de forma empírica, inspirados em animais de quatro patas com algumas simplificações estruturais. Ao se realizar a co-evolução da morfologia e dos parâmetros de controle, é possível que sejam descobertos novos modelos de robôs, sem equivalentes na natureza, que se mostrem mais eficientes na tarefa em questão (PFEIFER; SCHEIER, 1999).

Assim, o simulador LegGen original foi estendido, de forma que permitisse a evolução da morfologia em conjunto com os sistemas de controle do robô. Segundo Pfeifer e Scheier (1999), para que haja convergência nas soluções, é necessário que sejam impostas restrições ao modelo. As restrições que foram impostas são:

- Os robôs evoluídos são compostos de quatro pernas, com três segmentos por perna;
- Todos os segmentos são modelados através de caixas de tamanhos variados;
- Os pontos de conexão das juntas e os limites das mesmas são fixos.

Em outras palavras, apenas o tamanho dos segmentos é evoluído, e não a quantidade e a forma de conexão dos mesmos. Assim, foram incluídos novos genes no genoma original, para que fosse possível evoluir os parâmetros da morfologia. Estes novos genes foram definidos através alelos do tipo real, podendo assumir valores entre 0 e 2,5, com exceção das dimensões do corpo, que podem assumir valores entre 0 e 5. Estes alelos foram discretizados em intervalos de 0,01. Cada segmento do robô foi codificado utilizando três valores (dimensões em *x*, *y* e *z*).

Após a implementação do protótipo do LegGen, diversos experimentos foram realizados a fim de validar o modelo proposto, bem como realizar comparações entre as técnicas de controle utilizadas. O próximo capítulo descreve os resultados obtidos nestes experimentos.

6 Experimentos e resultados

Neste capítulo são descritos diversos experimentos realizados com o protótipo do modelo proposto, bem como os resultados obtidos nestes experimentos. Inicialmente, a Seção 6.1 descreve os experimentos realizados para se descobrir a função de *fitness* mais adequada para o problema em questão. Em seguida, a Seção 6.2 descreve os experimentos realizados para determinar o modelo de robô mais eficiente, dentre os propostos na Seção 5.4. A Seção 6.3 descreve os experimentos que avaliam as diversas estratégias de controle propostas. Por último, a Seção 6.4 descreve os experimentos realizados nos quais a morfologia foi evoluída em conjunto com os parâmetros da estratégia de controle do robô.

6.1 Escolha da função de *fitness*

Na Seção 5.3.1 do capítulo anterior, foram descritas quatro funções de *fitness* possíveis de serem utilizadas no problema em questão. A fim de se verificar a eficiência de cada uma destas funções, foram realizados dez experimentos distintos com cada uma delas, e os resultados obtidos são mostrados na Tabela 6.1 (os valores de F foram omitidos para se evitar comparações errôneas). O modelo de robô utilizado foi o TetraL3J (Figura 5.8(b)), e foi utilizada a estratégia de controle TabAngulos (Seção 5.2.2). Em experimentos preliminares, foi constatado que os resultados obtidos utilizando outros modelos de robôs são equivalentes aos da Tabela 6.1.

Tabela 6.1: Experimentos realizados para a escolha da função de *fitness*

	D			$D/(1+B)$			$D/1+a\times G)$			$D/(1+B+a\times G)$		
E	D	B	G	D	B	G	D	B	G	D	B	G
1	40,7	0,027	0,219	36,3	0,043	0,259	30,7	0,002	0,085	39,7	0,009	0,152
2	39,2	0,093	0,344	40,6	0,016	0,189	27,5	0,001	0,117	28,3	0,001	0,090
3	35,3	0,072	0,329	48,2	0,006	0,242	24,4	0,003	0,062	38,5	0,011	0,130
4	49,6	0,094	0,273	42,4	0,015	0,212	39,8	0,014	0,178	34,9	0,007	0,126
5	39,0	0,014	0,188	42,2	0,027	0,282	35,8	0,017	0,144	27,8	0,000	0,092
6	48,7	0,043	0,321	41,9	0,061	0,243	26,9	0,000	0,088	33,4	0,006	0,120
7	48,6	0,044	0,248	29,5	0,025	0,154	31,3	0,014	0,126	36,1	0,001	0,117
8	37,7	0,021	0,281	28,3	0,004	0,149	33,8	0,005	0,127	25,2	0,002	0,084
9	44,7	0,062	0,218	38,3	0,009	0,211	30,9	0,005	0,101	29,6	0,001	0,074
10	43,1	0,026	0,340	47,1	0,009	0,289	30,4	0,002	0,139	26,8	0,003	0,098
μ	**42,7**	**0,050**	**0,276**	**39,5**	**0,021**	**0,223**	**31,2**	**0,006**	**0,117**	**32,0**	**0,004**	**0,108**
σ	**5,1**	**0,029**	**0,057**	**6,6**	**0,018**	**0,049**	**4,5**	**0,006**	**0,034**	**5,1**	**0,004**	**0,024**

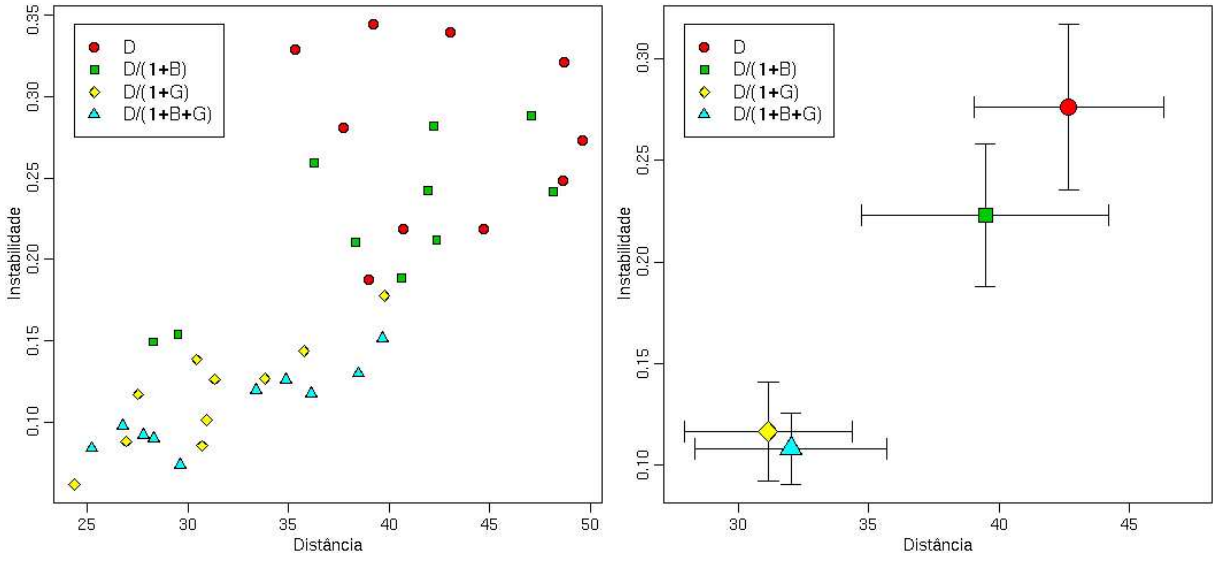
A primeira coluna (**E**) representa o índice de cada experimento. As demais colunas representam, respectivamente, os resultados obtidos utilizando as Fórmulas 5.16, 5.18, 5.23 e 5.22 como *fitness*. As sub-colunas desta tabela representam, respectivamente, a distância percorrida pelo robô (D) em 30 segundos, o índice dos *bumpers* (B), calculado através da Fórmula 5.19, e a taxa de instabilidade (G), calculada através da Fórmula 5.20. A duas últimas linhas da tabela trazem a média (μ) e o desvio padrão (σ) de cada coluna.

A Tabela 6.2 mostra os valores dos parâmetros do Algoritmo Genético utilizados nestes experimentos. O parâmetro *gaNpReplacement*, utilizado apenas em Algoritmos Genéticos do tipo GASteadyStateGA (vide Seção 5.3), representa o percentual de indivíduos da geração atual a serem substituídos na próxima geração.

Tabela 6.2: Parâmetros do Algoritmo Genético

Parâmetro	**Descrição**	**Valor**
gaNpCrossover	Taxa de cruzamentos	0,80
gaNpMutation	Taxa de mutação	0,08
gaNpReplacement	Taxa de substituição	1,00
gaNpopulationSize	Tamanho da população	150
gaNnGenerations	Número de gerações	300

O objetivo dos experimentos da Tabela 6.1 é determinar qual a função de *fitness* que apresenta a melhor relação entre velocidade e instabilidade, pois não basta que um robô se desloque de forma rápida, ele precisa ser estável para não sofrer quedas. A Figura 6.1 mostra dois gráficos que relacionam a distância percorrida D e a taxa de instabilidade G. A Figura 6.1(a) mostra os experimentos de forma individual, e a Figura 6.1(b) mostra a média e o intervalo de confiança (*Confidence Interval* – CI) dos resultados obtidos. Estes gráficos deixam claro que existe uma relação direta entre D e G, ou seja, as soluções mais velozes são as mais instáveis.



(a) Resultados individuais

(b) Média e intervalo de confiança

Figura 6.1: Relação entre a distância percorrida D e a taxa de instabilidade G

Observando os gráficos da Figura 6.1, nota-se que os experimentos realizados utilizando a Fórmula 5.16 como *fitness* se encontram em um dos extremos (maiores valores de D e G). A Figura 6.2 mostra um exemplo de caminhar evoluído utilizando esta função de *fitness*. O intervalo entre a captura de cada uma destas imagens foi de meio segundo, de forma que a Figura 6.2 ilustra 4 segundos de caminhada a uma taxa de 2 fps (*frames* por segundo).

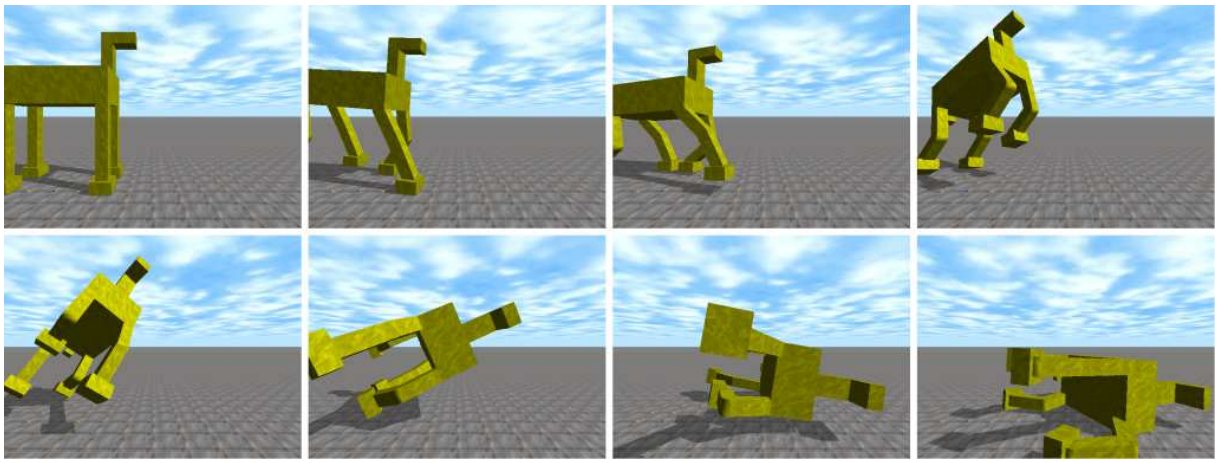


Figura 6.2: Caminhar evoluído utilizando a Fórmula 5.16 como *fitness* (2fps)

Analisando a Figura 6.2, se percebe que o robô não aprendeu realmente a caminhar, mas sim a saltar para frente. Os demais experimentos realizados utilizando a Fórmula 5.16 como *fitness* produzem resultados similares a este. O problema desta função de *fitness* é que ela privilegia inicialmente soluções instáveis, nas quais o robô simplesmente tomba para frente, em detrimento de soluções mais estáveis, que poderiam manter o robô equilibrado. Embora existam modelos robôs que podem se deslocar saltando (BEKEY, 2005), este não é o objetivo do trabalho em questão.

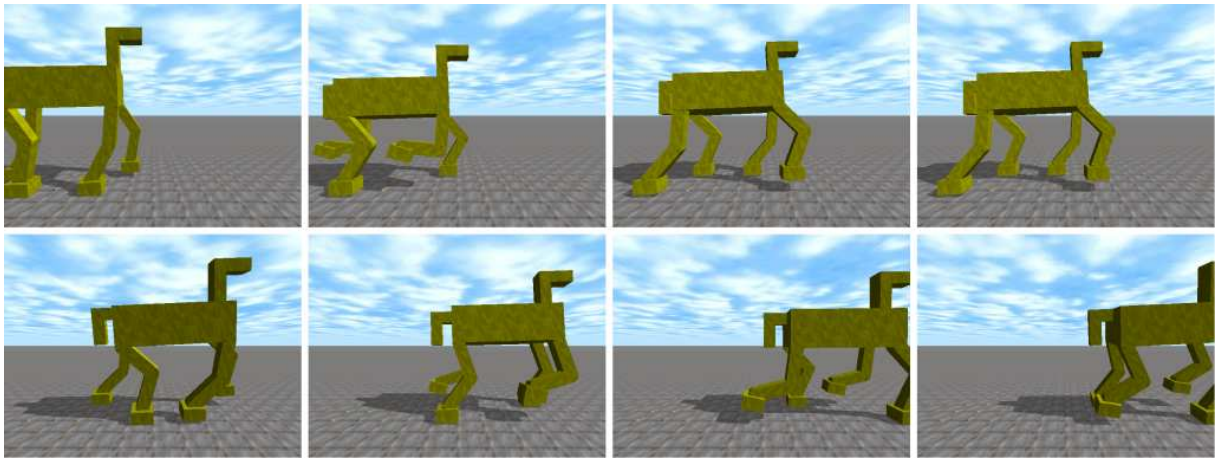


Figura 6.3: Caminhar evoluído utilizando a Fórmula 5.18 como *fitness* (1fps)

A Figura 6.3 mostra um exemplo de caminhar obtido utilizando a Fórmula 5.18 como *fitness*, e a Figura 6.4 mostra um exemplo de caminhar obtido utilizando a Fórmula 5.22. O intervalo de captura entre as imagens é de um segundo (1fps), de forma que as Figuras 6.3 e 6.4 ilustram 8 segundos de caminhada cada.

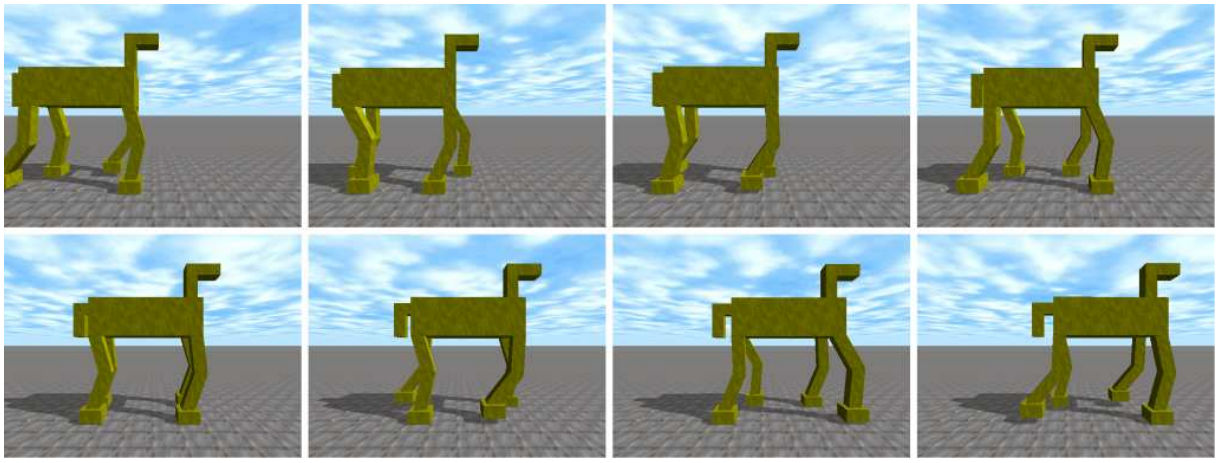


Figura 6.4: Caminhar evoluído utilizando a Fórmula 5.22 como *fitness* (1fps)

Percebe-se que o caminhar da Figura 6.3 está mais próximo de um "galope" (mais veloz e instável), enquanto que o caminhar da Figura 6.4 é mais parecido com um "trote" suave (um pouco mais lento e estável). A Figura 6.5 mostra um gráfico de *boxplot*[1] e o intervalo de confiança (CI) a 95% dos valores de D/G, referentes aos experimentos descritos na Tabela 6.1. Apesar das diferenças não serem significativas do ponto de vista estatístico, percebe-se pela Figura 6.5 que a Fórmula 5.22 apresentou a melhor relação entre D e G. Assim, esta fórmula foi selecionada para ser utilizada como função de *fitness* nos demais experimentos.

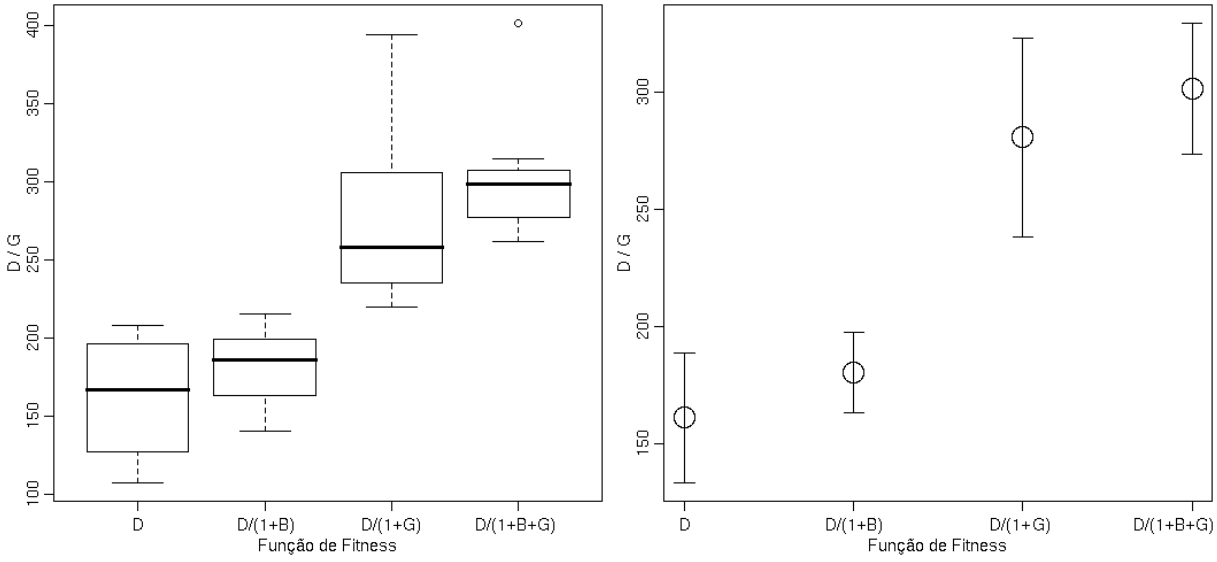


Figura 6.5: Gráfico de *boxplot* e CI dos experimentos da Tabela 6.1

É importante ressaltar que o valor de B não foi utilizado nos gráficos porque ele é mais importante nas primeiras gerações, para punir os indivíduos que não conseguem se deslocar de forma coordenada. Nas últimas gerações sua importância é bastante reduzida, pois como a maioria dos indivíduos já consegue caminhar de forma satisfatória, o valor de B tende a zero.

[1]Um gráfico de *boxplot* (LAW; KELTON, 2000) é uma ferramenta estatística que serve para a visualização de distribuições de probabilidade. Ele mostra a mediana (linha forte ao centro), o 1°e o 3°quartil (limites do retângulo), o 10°e o 90°percentil (retas mais afastadas) e os extremos (pequenos círculos isolados).

6.2 Escolha do modelo de robô

Após a escolha da função de *fitness*, esta seção descreve os experimentos realizados para determinar qual o modelo de robô, dentre os mostrados na Figura 5.8, é mais eficiente na tarefa em questão. A idéia é selecionar um robô que consiga caminhar de forma satisfatória utilizando o menor número de juntas possível, pois quanto mais simples for o robô, mais barato se torna a construção futura do mesmo. Desta forma, foram realizados dez experimentos distintos com cada modelo de robô, e os resultados são mostrados na Tabela 6.3.

Tabela 6.3: Experimentos realizados para selecionar o modelo de robô

	HexaL3J			**TetraL3J**			**HexaL2J**			**TetraL2J**		
E	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*
1	17,84	49,6	0,174	15,72	39,7	0,152	14,83	26,1	0,076	11,85	23,6	0,098
2	17,80	47,5	0,166	14,88	28,3	0,090	15,34	27,9	0,079	12,18	20,7	0,069
3	16,48	39,4	0,138	16,66	38,5	0,130	16,10	24,4	0,050	4,45	11,0	0,145
4	18,34	41,1	0,124	15,39	34,9	0,126	15,96	29,4	0,082	5,72	12,5	0,118
5	16,63	46,6	0,178	14,49	27,8	0,092	13,58	26,9	0,096	11,17	15,0	0,034
6	16,55	50,9	0,205	15,16	33,4	0,120	16,18	29,0	0,078	10,34	15,3	0,047
7	17,70	36,9	0,108	16,60	36,1	0,117	15,43	25,4	0,065	10,02	19,4	0,089
8	13,51	38,6	0,185	13,70	25,2	0,084	16,71	29,8	0,077	9,58	16,0	0,067
9	15,72	34,2	0,116	17,03	29,6	0,074	13,31	24,8	0,086	7,46	11,8	0,058
10	15,80	30,9	0,095	13,51	26,8	0,098	16,71	35,4	0,109	10,78	21,3	0,097
μ	**16,64**	**41,6**	**0,149**	**15,31**	**32,0**	**0,108**	**15,41**	**27,9**	**0,080**	**9,36**	**16,7**	**0,082**
σ	**1,42**	**6,8**	**0,038**	**1,22**	**5,1**	**0,024**	**1,19**	**3,3**	**0,016**	**2,62**	**4,4**	**0,034**

A primeira coluna (**E**) representa o índice do experimento. As demais colunas representam, respectivamente, os resultados obtidos nos experimentos utilizando os robôs HexaL3J (Figura 5.8(a)), TetraL3J (Figura 5.8(b)), HexaL2J (Figura 5.8(c)) e TetraL2J (Figura 5.8(d)). As sub-colunas desta tabela representam, respectivamente, o valor do *fitness* F, calculado através da Fórmula 5.22, a distância percorrida pelo robô D em 30 segundos, e a taxa de instabilidade G, calculada através da Fórmula 5.20. As duas últimas linhas trazem a média (μ) e o desvio padrão (σ) de cada uma das colunas. O controle foi realizado utilizando um autômato baseado em uma tabela de ângulos (TabAngulos), e os parâmetros utilizados no Algoritmo Genético são os mesmos da Tabela 6.2.

A Figura 6.6 mostra o gráfico de *boxplot* e o intervalo de confiança (CI) a 95%, dos valores de F dos experimentos da Tabela 6.3. Observando-se os resultados da Tabela 6.3 e os gráficos da Figura 6.6, percebe-se que o modelo de robô que obteve o pior desempenho foi o TetraL2J (Figura 5.8(d)). A Figura 6.7 mostra um exemplo de caminhada realizada por este robô, no qual se pode notar que o TetraL2J apresenta sérias dificuldades para se locomover. Com relação aos demais modelos, embora o desempenho dos robôs de seis pernas (HexaL3J e HexaL2J) seja ligeiramente melhor que o desempenho do robô TetraL3J, a diferença nos resultados não é significativa do ponto de vista estatístico. Assim, o robô selecionado para ser utilizado nos próximos experimentos foi o TetraL3J (Figura 5.8(b)), pois este consegue se deslocar de forma similar aos demais utilizando apenas quatro pernas, o que o torna mais simples e econômico em relação aos gastos de energia e em relação aos custos de *hardware*.

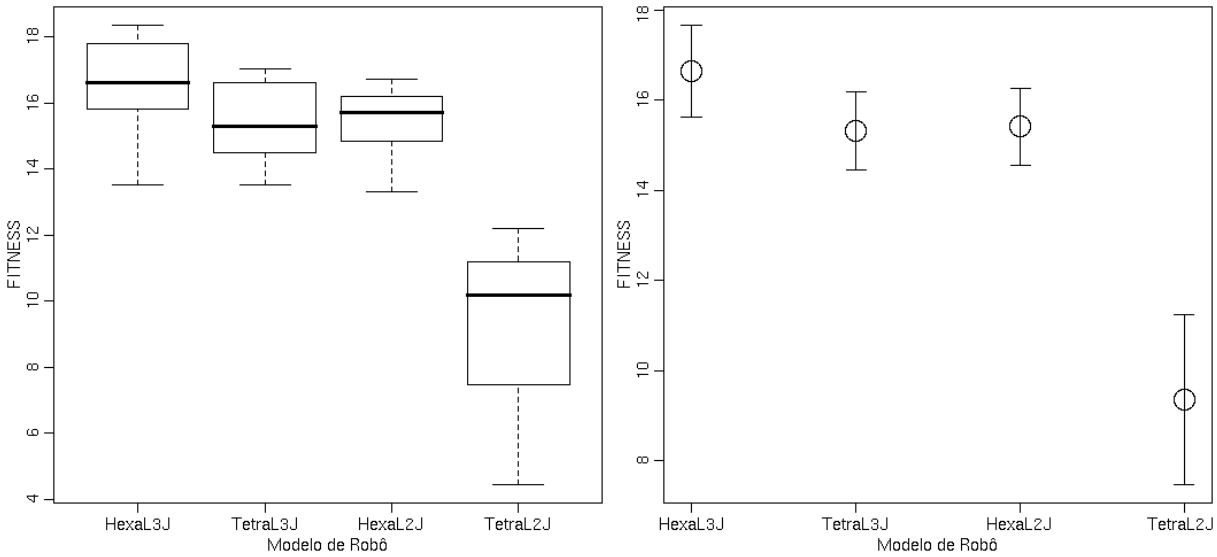


Figura 6.6: Gráfico de *boxplot* e CI dos experimentos da Tabela 6.3

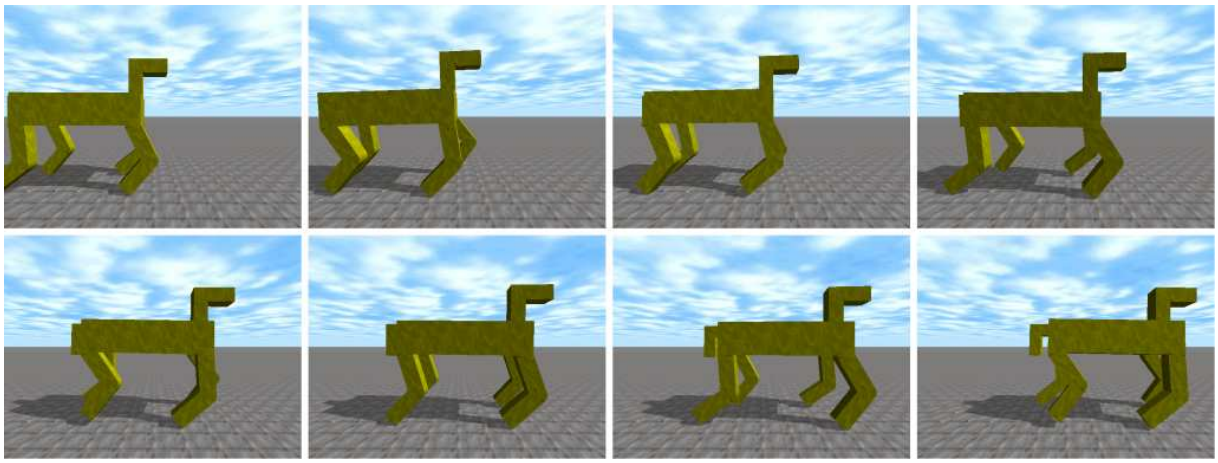


Figura 6.7: Caminhar realizado pelo robô TetraL2J (1fps)

A título de exemplificação, a Figura 6.8 mostra um exemplo de caminhada realizada pelo robô HexaL3J (Figura 5.8(a)), e a Figura 6.9 mostra um exemplo de caminhada realizada pelo robô HexaL2J (Figura 5.8(c)). Percebe-se que devido ao passo *tripod* (tripé), estes robôs conseguem se deslocar de forma rápida e eficiente, pois possuem estabilidade estática.

6.3 Avaliação das estratégias de controle

Esta seção descreve os experimentos realizados visando avaliar o desempenho das diversas estratégias de controle propostas no Capítulo 5. Nestes experimentos, foi utilizada a Fórmula 5.22 como função de *fitness*, e o modelo de robô utilizado foi o TetraL3J. Os parâmetros utilizados no Algoritmo Genético são os mesmos da Tabela 6.2, com exceção do tamanho da população e do número de gerações, que foram respectivamente aumentados para 350 e 700. Estas alterações foram realizadas para reduzir a influência do tamanho do espaço de estados nos resultados. A Tabela 6.4 mostra os resultados obtidos nestes experimentos.

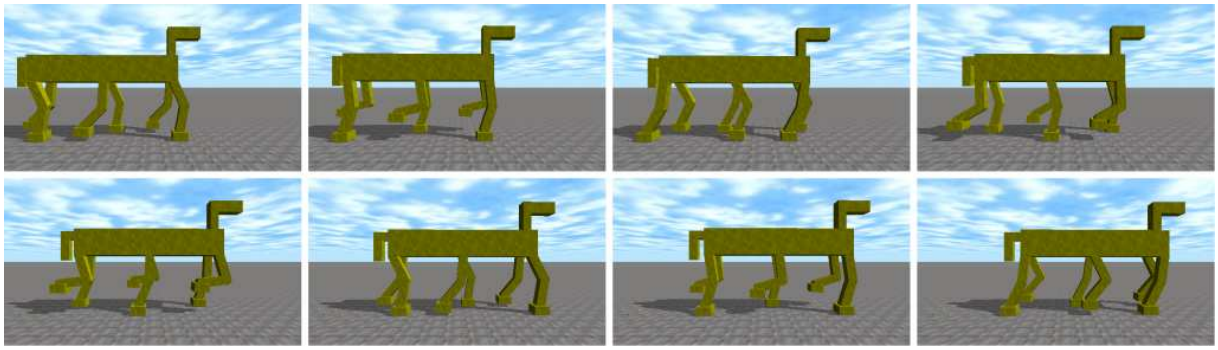


Figura 6.8: Caminhar realizado pelo robô HexaL3J (1fps)

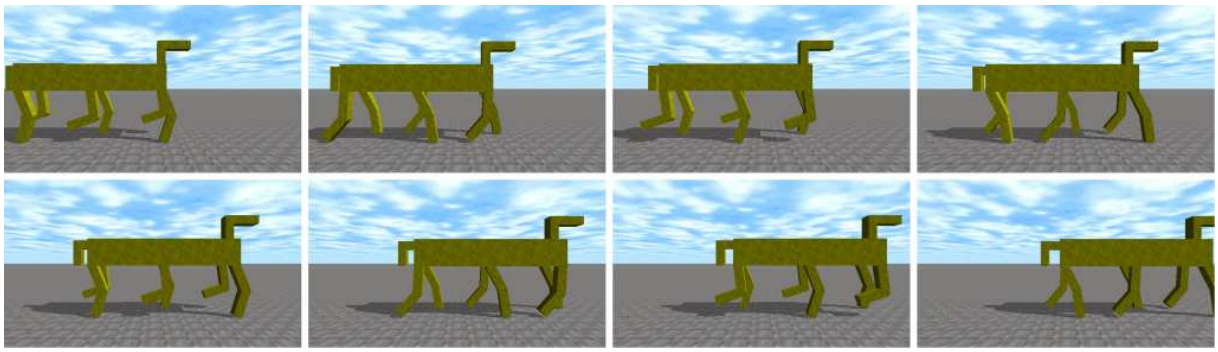


Figura 6.9: Caminhar realizado pelo robô HexaL2J (1fps)

A primeira coluna (**E**) representa o índice do experimento. As demais colunas representam, respectivamente, os resultados obtidos utilizando um autômato baseado em uma tabela de ângulos (TabAngulos), um autômato baseado em uma tabela de posições (TabLocus), funções cíclicas no formato de meia-elipse (MeiaElipse), e uma Rede Neural do tipo Elman (CtrlNeural). As sub-colunas representam, respectivamente, o *fitness* (F), calculado através da Fórmula 5.22, a distância percorrida pelo robô (D) em 30 segundos e a taxa de instabilidade (G), calculada através da Fórmula 5.20. As duas últimas linhas da tabela trazem a média (μ) e o desvio padrão (σ) de cada coluna.

Nos experimentos realizados utilizando a tabela de ângulos (TabAngulos) e a tabela de posições (TabLocus), foram utilizados 4 estados no autômato. No controle neural (CtrlNeural), foram utilizados 3 neurônios na camada oculta. Estes parâmetros foram determinados através da realização de diversos experimentos, estatisticamente válidos (vide Anexo A), nos quais estes valores se mostraram bastante adequados. A lista a seguir mostra os tempos médios de evolução (utilizando 350 indivíduos e 700 gerações) de cada uma das estratégias:

- TabAngulos: 318,91 minutos (5,32 horas);
- TabLocus: 311,45 minutos (5,19 horas);
- MeiaElipse: 1266,07 minutos (21,10 horas);
- CtrlNeural: 576,43 minutos (9,61 horas);

As diferenças nos tempos médios de evolução se devem as características de cada estratégia. Na meia elipse, por exemplo, o cálculo da cinemática inversa precisa ser realizado em tempo real (a cada 50 milisegundos de simulação), o que consome bastante tempo. Já utilizando a tabela de ângulos (TabAngulos), quase nenhum cálculo precisa ser realizado em tempo real, o que melhora bastante o desempenho. O tempo total despendido na realização dos experimentos

Tabela 6.4: Avaliação as estratégias de controle

	TabAngulos			**TabLocus**			**MeiaElipse**			**CtrlNeural**		
E	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*
1	14,04	32,2	0,128	12,75	23,8	0,086	16,94	33,2	0,095	16,27	29,2	0,079
2	14,28	32,4	0,126	11,79	30,7	0,160	17,20	30,7	0,078	16,63	28,3	0,069
3	13,18	30,3	0,129	13,35	27,3	0,104	15,47	28,5	0,084	16,99	27,8	0,063
4	15,87	26,8	0,069	13,05	27,3	0,107	16,04	30,4	0,089	16,68	27,9	0,067
5	16,64	36,6	0,120	12,37	30,2	0,142	16,51	33,3	0,101	16,16	28,2	0,074
6	16,48	27,7	0,068	14,47	37,1	0,152	17,90	37,9	0,111	15,97	31,1	0,093
7	14,88	31,7	0,112	11,60	27,3	0,135	17,08	37,5	0,119	17,33	29,6	0,070
8	13,77	29,0	0,110	13,57	23,0	0,069	16,75	33,6	0,100	16,65	29,0	0,074
9	15,33	34,4	0,124	13,51	31,4	0,130	17,50	29,9	0,070	16,29	30,2	0,085
10	15,80	37,0	0,134	14,15	24,6	0,073	15,95	31,6	0,097	16,23	29,8	0,083
μ	**15,03**	**31,8**	**0,112**	**13,06**	**28,3**	**0,116**	**16,16**	**31,1**	**0,092**	**16,52**	**29,1**	**0,076**
σ	**1,19**	**3,5**	**0,024**	**0,95**	**4,2**	**0,033**	**1,47**	**2,5**	**0,012**	**0,42**	**1,1**	**0,009**

da Tabela 6.4 foi de 412,15 horas (17,17 dias). A Figura 6.10 mostra os gráficos de *boxplot* e do intervalo de confiança (CI) a 95%, relativos ao *fitness* dos experimentos da Tabela 6.4.

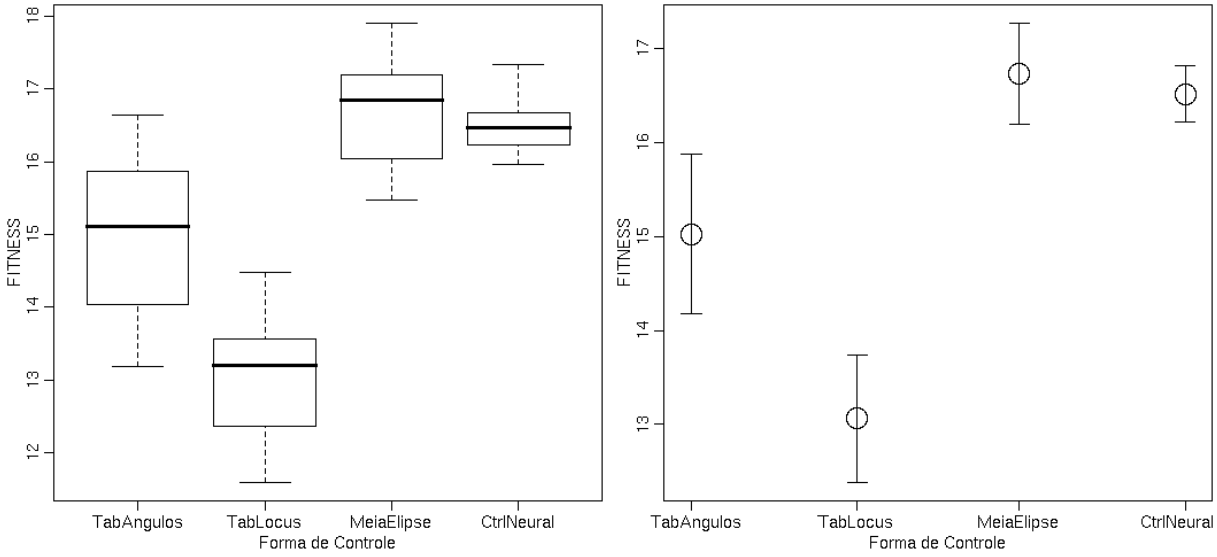
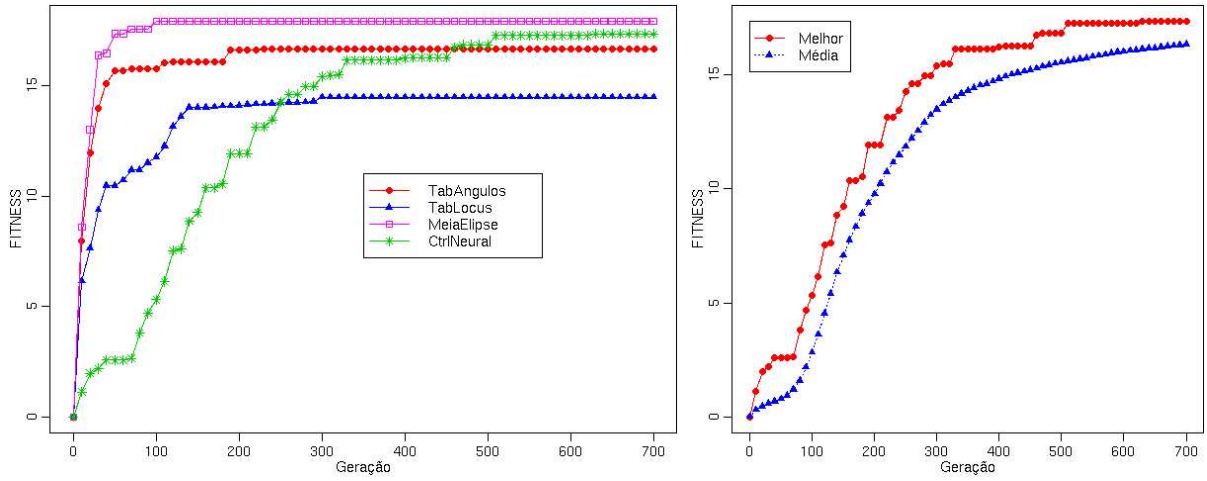


Figura 6.10: Gráfico de *boxplot* e CI dos experimentos da Tabela 6.4

Pela análise dos gráficos da Figura 6.10, pode-se afirmar com 95% de confiança que os resultados obtidos com as duas primeiras estratégias (TabAngulos e TabLocus) são inferiores aos demais resultados. Com relação as outras estratégias, não é possível afirmar que uma seja superior a outra, pois as diferenças não são significativas do ponto de vista estatístico (os intervalos de confiança se sobrepõe). A Figura 6.11(a) mostra um gráfico que compara a evolução das soluções (*fitness* do melhor indivíduo) obtidas com as quatro estratégias de controle. Os experimentos utilizados neste gráfico foram os melhores resultados de cada estratégia.

Percebe-se que a meia elipse atingiu seu melhor resultado ($F = 17,90$) em aproximadamente 95 épocas, enquanto que o controlador neural precisou de 625 épocas para chegar a um



(a) Progresso das diversas estratégias

(b) Progresso do controlador neural

Figura 6.11: Evolução das melhores soluções durante o aprendizado

resultado similar ($F = 17,33$). Assim, pode-se afirmar que o controlador neural precisa de muito mais épocas de evolução que as demais técnicas para atingir resultados satisfatórios. Isto se deve ao tamanho do espaço de estados, que é bem maior no controlador neural (a meia elipse só tem 6 parâmetros, enquanto a ANN tem 40 pesos sinápticos). A Figura 6.11(b) mostra o progresso do melhor indivíduo (círculos) e da média da população (triângulos) durante a evolução do melhor experimento do controlador neural.

Utilizando os resultados da Tabela 6.4 bem como os gráficos das Figuras 6.10 e 6.11 e a análise visual dos resultados (caminhar realizado pelos robôs), foi possível avaliar as características de cada uma das técnicas de controle. As próximas seções descrevem estas características, bem como as vantagens e desvantagens de cada uma das técnicas propostas. Cabe ressaltar que nenhuma destas estratégias pode ser considerada ideal para todas as situações, e a escolha da estratégia mais indicada deve levar em conta o ambiente em que o robô irá atuar (PFEIFER; SCHEIER, 1999).

6.3.1 Controle baseado em uma tabela de ângulos

Nesta estratégia de controle (TabAngulos), um autômato de quatro estados foi utilizado para o controle das juntas do robô. As principais vantagens desta estratégia são:

- Poucos parâmetros a serem otimizados (apenas 11), o que facilita a convergência;
- Não são impostas restrições severas quanto a forma de caminhar;
- Simplicidade de operação, o que garante uma boa performance em tempo real.

Exemplos de robôs controlados através desta técnica foram mostrados anteriormente nas Figuras 6.8 (HexaL3J), 6.4 (TetraL3J), 6.9 (HexaL2J) e 6.7 (TetraL2J). As principais desvantagens desta estratégia de controle são:

- A movimentação dos *endpoints* é um tanto restrita: com quatro estados, a melhor trajetória possível é um triângulo. Movimentos mais sofisticados podem ser obtidos utilizando mais estados, mas isto aumenta o espaço de estados e dificulta a convergência;

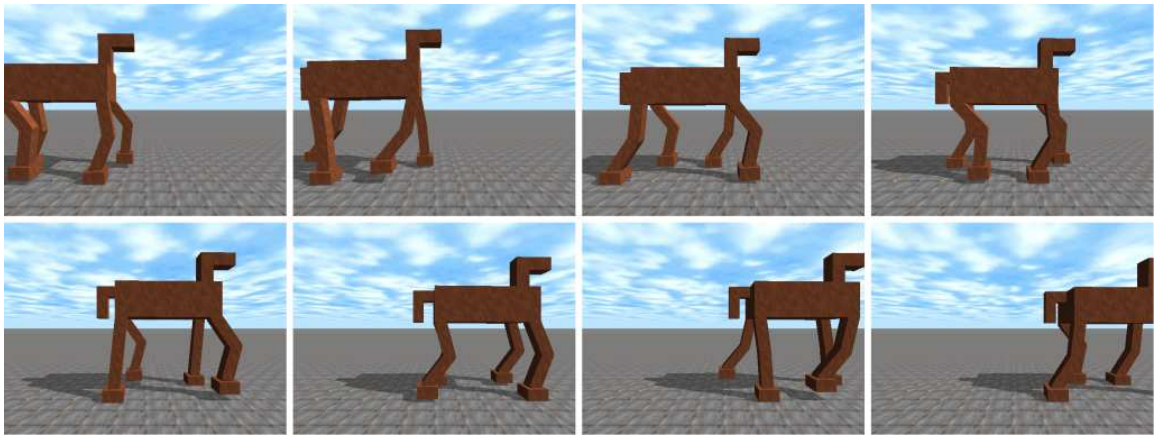


Figura 6.12: Robô controlado por uma tabela de posições (1fps)

- Falta de adaptabilidade: a movimentação das juntas ocorre de forma pré-programada;
- Para se utilizar o passo "trote", todas as pernas do robô precisam ser iguais, o que torna a movimentação do robô diferente de seus equivalentes biológicos;
- Os resultados obtidos são inferiores aos da MeiaElipse e do CtrlNeural.

6.3.2 Controle baseado em uma tabela de posições

Nesta estratégia de controle (TabLocus), o Algoritmo Genético evolui uma tabela descrevendo as posições dos *endpoints* desejadas para cada um dos quatro estados do autômato. Em seguida, o cálculo da cinemática inversa é realizado utilizando o método de Powell (BRENT, 1973; ACTON, 1970), e assim é gerada uma tabela de controle similar a utilizada na estratégia anterior. As principais vantagens desta estratégia são:

- Número reduzido de parâmetros a serem otimizados, o que facilita a convergência;
- Não são impostas restrições severas quanto a forma de caminhar;
- Bom desempenho em tempo real (a cinemática inversa é calculada a priori);
- O "trote" pode ser utilizado mesmo se as pernas forem diferentes entre si.

A Figura 6.12 mostra um exemplo de robô que se desloca utilizando esta estratégia de controle. As principais desvantagens desta estratégia são:

- A movimentação dos *endpoints* é um tanto restrita (a melhor é um triângulo);
- Necessidade de se calcular a cinemática inversa;
- Falta de adaptabilidade: as juntas se movimentam de forma pré-programada;
- Os resultados obtidos são inferiores aos das demais técnicas.

6.3.3 Controle baseado em funções cíclicas

Nesta estratégia de controle, as trajetórias dos *endpoints* são controladas através de uma função cíclica (MeiaElipse), e o GA é utilizado apenas para a otimização dos parâmetros desta elipse, e não para a definição do caminhar propriamente dito. As principais vantagens desta estratégia de controle são:

- Poucos parâmetros livres (apenas 6), o que garante uma rápida convergência;

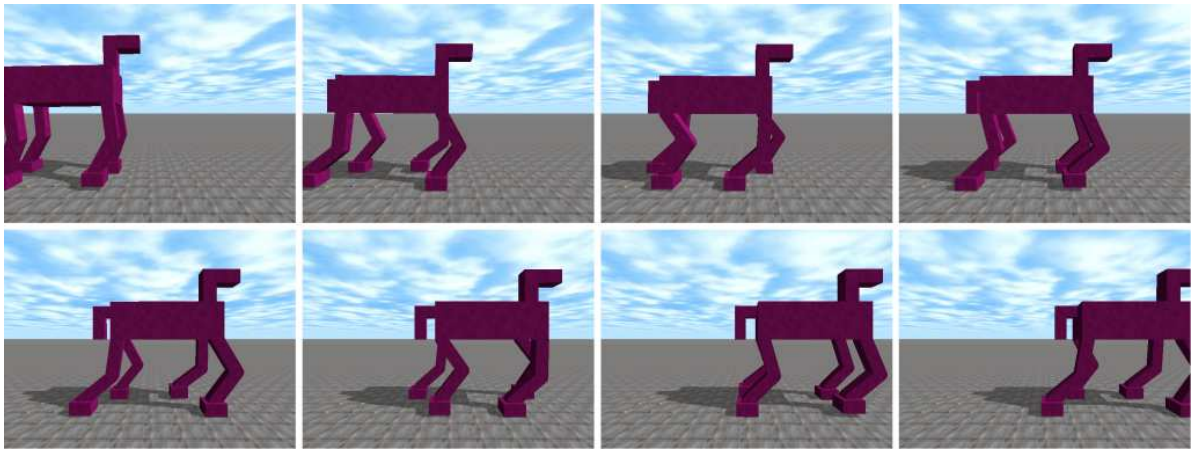


Figura 6.13: Robô controlado por uma meia-elipse (1fps)

- O caminhar obtido é bastante estável em superfícies planas: o corpo do robô se mantém praticamente na mesma altura durante o caminhar;
- O "trote" pode ser utilizado mesmo se as pernas forem diferentes;
- Os resultados obtidos são bastante promissores.

A Figura 6.13 mostra um exemplo de robô que se desloca utilizando esta estratégia de controle. As principais desvantagens desta estratégia são:

- Caminhar bastante restrito: a movimentação das juntas ocorre de forma extremamente rígida, o que acarreta os mesmos problemas que ocorriam nas máquinas caminhantes (Seção 4.1) controladas por engrenagens (RAIBERT, 1986);
- Ocorrência de pequenos impactos quando os *endpoints* tocam o chão. Nos seres vivos, as patas percorrem uma trajetória mais complexa para reduzir os impactos (BEKEY, 2005);
- Necessidade de se calcular a cinemática inversa de forma rápida e eficiente, o que pode prejudicar o desempenho em tempo real.

6.3.4 Controle neural

Nesta estratégia de controle (CtrlNeural), uma ANN é utilizada para o controle das juntas do robô. Esta ANN recebe como entrada os ângulos atuais das juntas do robô, e devolve na saída os ângulos desejados. As principais vantagens desta estratégia são:

- Plausibilidade biológica: os CPGs (Seção 2.5) presentes nos seres vivos são compostos de neurônios biológicos;
- Adequação ao princípio dos processos paralelos fracamente acoplados (Seção 2.7.3) e ao princípio da aprendizagem (Seção 2.7.5);
- Grande poder de representação, permitindo movimentos suaves e sofisticados;
- Não impõe restrições quanto a forma de caminhar;
- Generalização e robustez frente a situações novas e inesperadas (Figura 6.21);
- Resultados promissores e excelente desempenho em tempo real.

A Figura 6.14 mostra um exemplo de robô que se desloca utilizando esta estratégia de controle. As principais desvantagens desta estratégia são:

- Excesso de parâmetros livres (40 pesos sinápticos), o que dificulta o aprendizado;

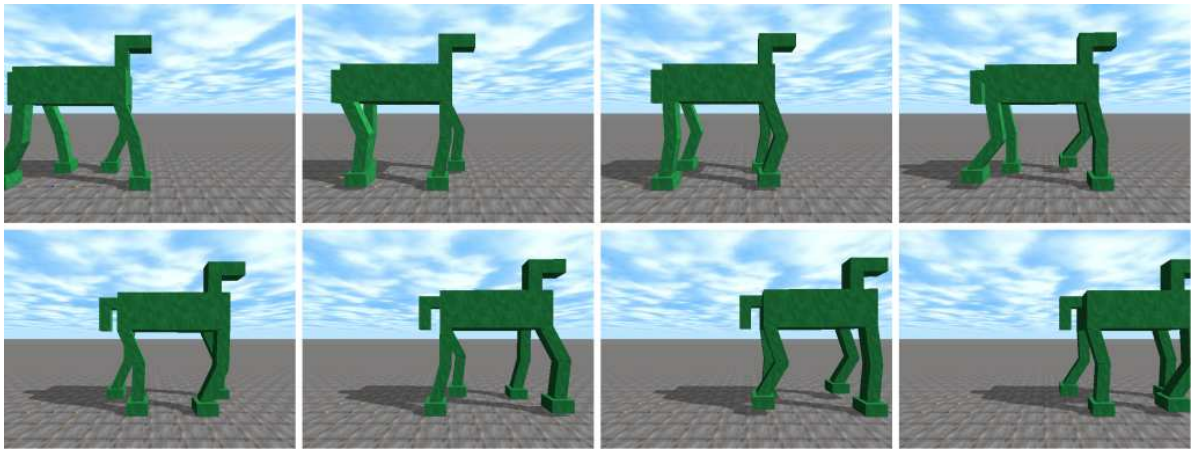


Figura 6.14: Robô controlado por uma Rede Neural do tipo Elman (1fps)

- Dificuldade de configuração do tamanho da camada oculta: com poucos neurônios, a ANN não terá poder de representação; com muitos neurônios, o espaço de busca se tornará muito grande, dificultando ainda mais a convergência.

Apesar das dificuldades de convergência, o controle neural se mostra uma boa alternativa para o problema em questão, pois garante um maior grau de robustez e adaptabilidade.

6.4 Co-evolução da morfologia e controle

Esta seção descreve os experimentos realizados nos quais a morfologia do robô foi evoluída em conjunto com os parâmetros de controle. Nestes experimentos, foram utilizados os mesmos parâmetros da Tabela 6.2 no GA, com exceção do tamanho da população, que foi aumentado para 350, e do número de gerações, que foi aumentado para 700. A Tabela 6.5 mostra os resultados obtidos nestes experimentos.

Tabela 6.5: Experimentos realizados com a evolução da morfologia e do controle

	TabAngulos			**TabLocus**			**MeiaElipse**			**CtrlNeural**		
E	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*
1	24,73	35,9	0,045	19,03	45,7	0,140	17,02	39,5	0,132	18,80	38,0	0,101
2	23,63	63,2	0,167	17,48	42,9	0,145	20,85	41,6	0,099	17,90	33,0	0,075
3	20,37	54,0	0,164	17,33	48,5	0,179	17,41	32,4	0,086	19,84	39,5	0,099
4	17,86	49,8	0,179	16,16	51,3	0,216	15,69	43,6	0,177	17,80	37,9	0,113
5	20,86	45,1	0,116	18,63	46,1	0,147	19,30	44,8	0,129	20,09	27,4	0,031
6	18,25	50,7	0,178	17,57	37,7	0,115	18,34	33,8	0,084	15,90	32,8	0,105
7	19,97	53,4	0,166	17,41	51,0	0,192	15,35	30,6	0,098	18,87	41,1	0,117
8	21,01	36,0	0,071	15,45	36,1	0,134	21,14	44,3	0,109	18,50	36,2	0,095
9	24,57	59,9	0,143	15,70	47,6	0,202	18,14	40,0	0,120	19,08	39,2	0,105
10	20,14	36,8	0,083	13,76	51,7	0,273	16,00	36,8	0,129	15,57	37,4	0,140
μ	**21,14**	**48,5**	**0,131**	**16,85**	**45,8**	**0,174**	**17,92**	**38,7**	**0,116**	**18,24**	**36,2**	**0,098**
σ	**2,43**	**9,8**	**0,049**	**1,59**	**5,5**	**0,048**	**2,04**	**5,1**	**0,028**	**1,50**	**4,1**	**0,029**

A primeira coluna (**E**) representa o índice do experimento. As demais colunas representam, respectivamente, os resultados obtidos utilizando as quatro estratégias de controle. As sub-colunas representam, respectivamente, o *fitness* (F), a distância percorrida (D) e a taxa de instabilidade (G). As duas últimas linhas da tabela trazem a média (μ) e o desvio padrão (σ) dos resultados de cada coluna.

A Figura 6.15 mostra os gráficos de *boxplot* e do intervalo de confiança. Estes gráficos relacionam os experimentos da Tabela 6.4 com os experimentos da Tabela 6.5. Percebe-se cla-

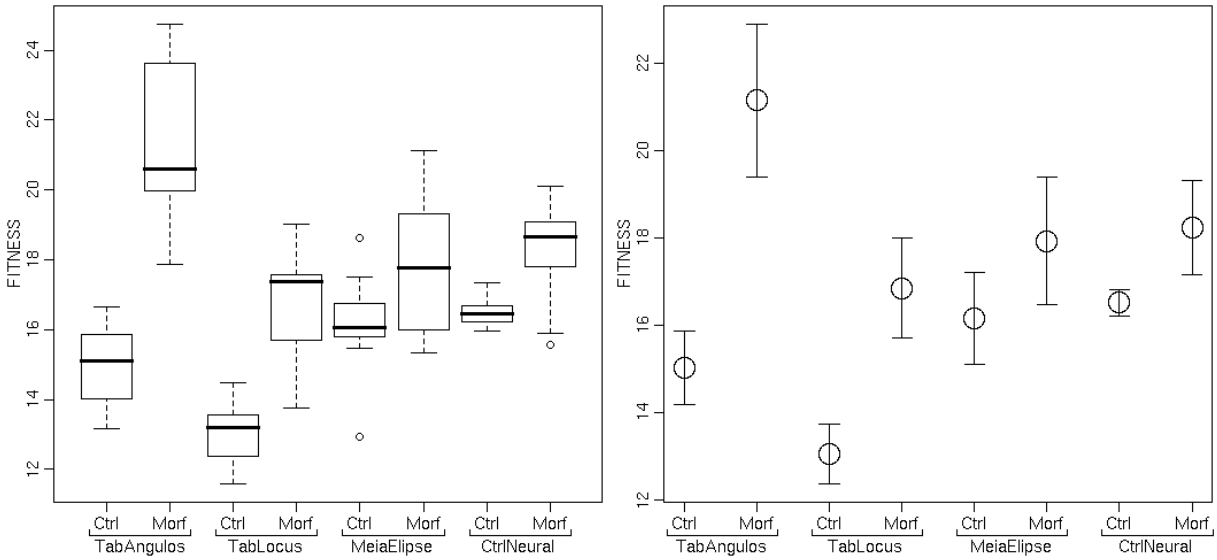


Figura 6.15: Gráfico de *boxplot* e CI dos experimentos da Tabela 6.5

ramente que, em todas as estratégias, a evolução da morfologia em conjunto com os parâmetros de controle produziu melhores resultados do que a evolução dos parâmetros de controle de forma isolada. Nos experimentos da Tabela 6.5, o controle baseado em uma tabela de ângulos (TabAngulos) foi o que garantiu os melhores resultados. Isto ocorre porque esta estratégia possui um poder de representação razoável utilizando poucos parâmetros livres (apenas 11). Já o controle neural (CtrlNeural) foi um pouco prejudicado pelo aumento do espaço de estados, mas mesmo assim houve uma melhoria razoável nos resultados.

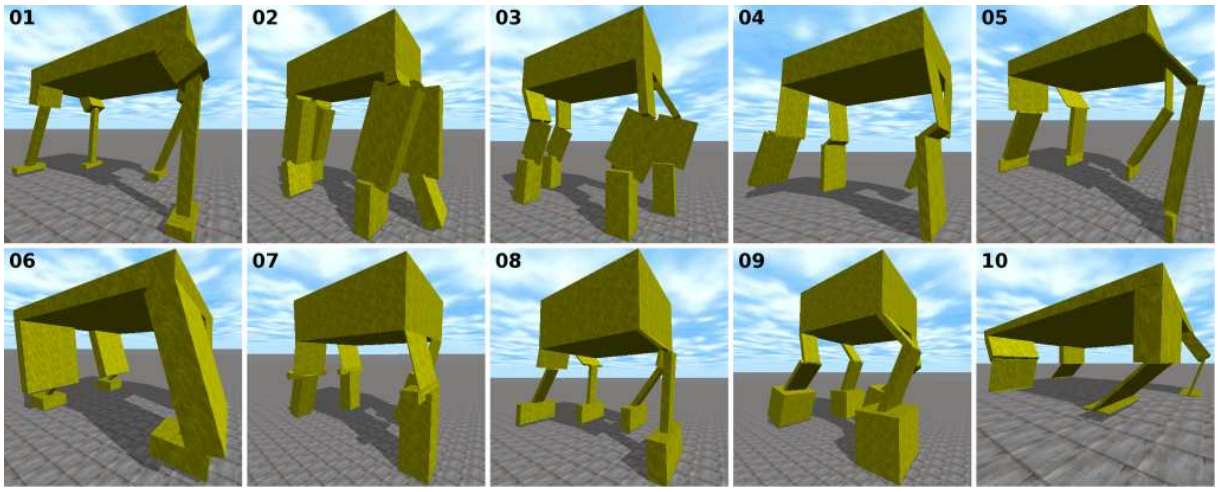


Figura 6.16: Morfologia final obtida em cada experimento

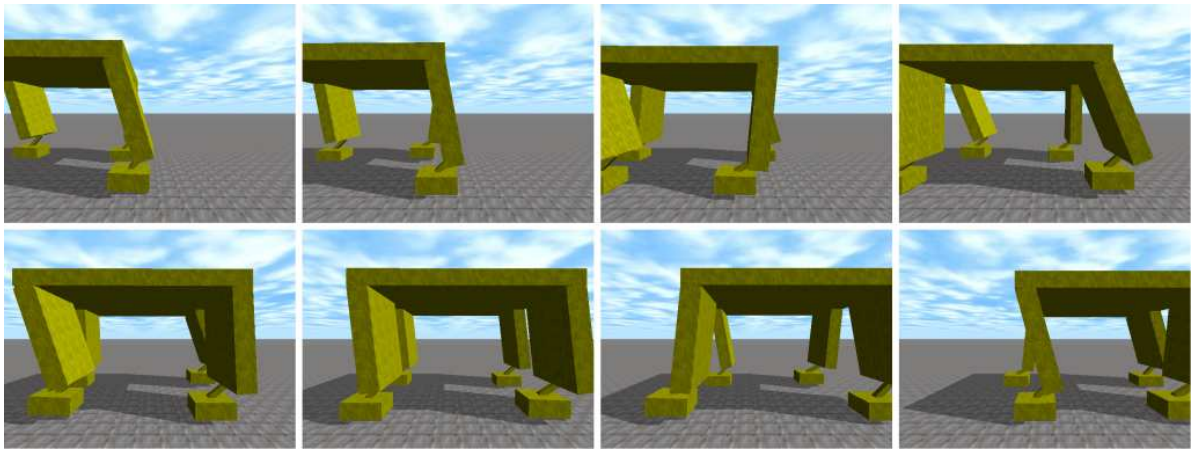


Figura 6.17: Robô evoluído no experimento 06 (1fps)

A Figura 6.16 mostra as morfologias que evoluíram ao final das 700 gerações, para cada experimento controlado pela tabela de ângulos. Os números no canto superior esquerdo se referem ao experimento no qual a morfologia do robô foi evoluída. A Figura 6.17 mostra o caminhar de um dos modelos evoluídos.

A título de demonstração, a Figura 6.18 mostra as alterações da morfologia de um robô durante a evolução. Os números no canto superior esquerdo se referem a geração na qual cada modelo de robô se tornou dominante. Percebe-se que a morfologia vai aos poucos sendo melhorada, até que se chegue a um modelo de robô parecido com o da Figura 5.8(b). Cabe ressaltar que o espaço de estados permite que apareçam soluções diferentes entre si, mas igualmente eficientes, a exemplo do que ocorre nos seres vivos.

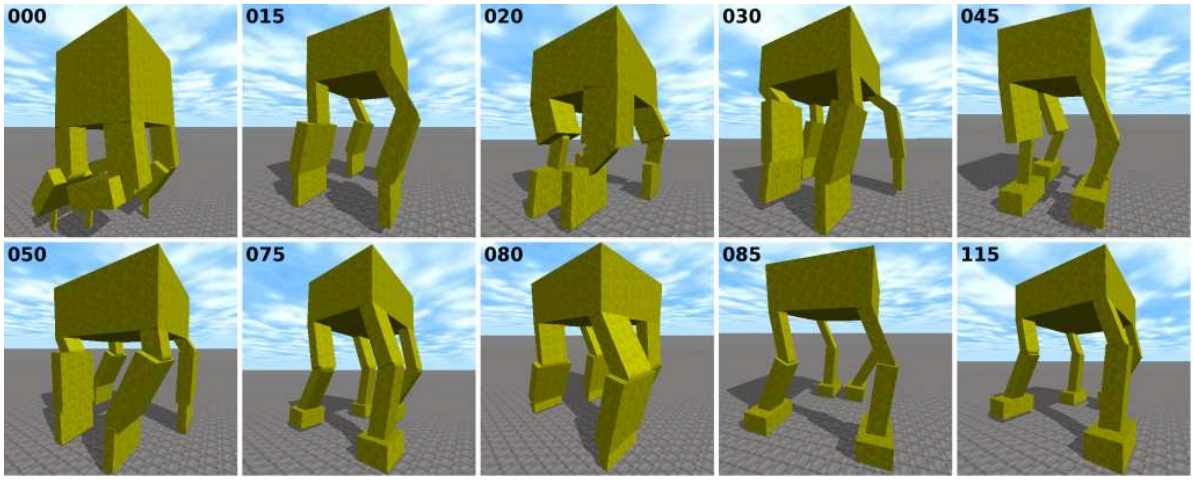


Figura 6.18: Progresso da evolução da morfologia

Outra área na qual a evolução da morfologia é especialmente útil é no projeto de robôs que precisam atuar em ambientes irregulares. Segundo Pfeifer e Scheier (1999), o ambiente em que um robô deve atuar deve fazer parte do projeto do mesmo, pois na natureza não existem seres universais. De fato, todos os animais atuam em determinado nicho ecológico: os cavalos, que se deslocam com bastante eficiência nos campos, tem dificuldades em terrenos pedregosos; os cachorros não são capazes de subir em árvores ou até mesmo em escadas; já os gatos conseguem se deslocar facilmente nestes tipos de terrenos, apesar de seu tamanho reduzido.

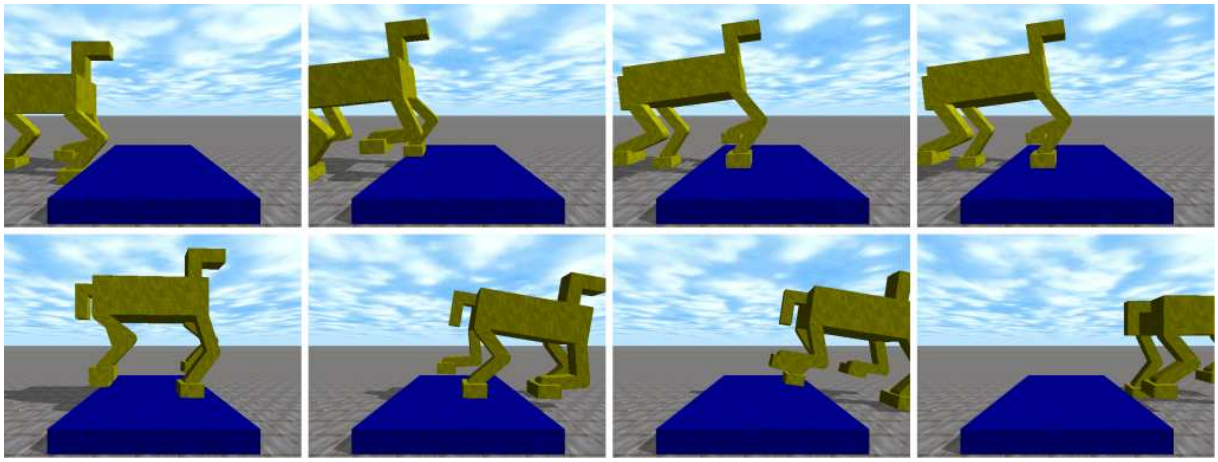


Figura 6.19: Robô TetraL3J especialmente treinado para transpor desníveis

Assim, embora um robô como o TetraL3J consiga transpor desníveis sem maiores dificuldades (Figura 6.19), ele não foi projetado para subir escadas (vide Anexo D). Mas utilizando a evolução da morfologia em conjunto com os parâmetros de controle, é possível o surgimento de robôs especializados nesta tarefa, como mostra a Figura 6.20. Este robô foi controlado por uma tabela de ângulos durante o caminhar.

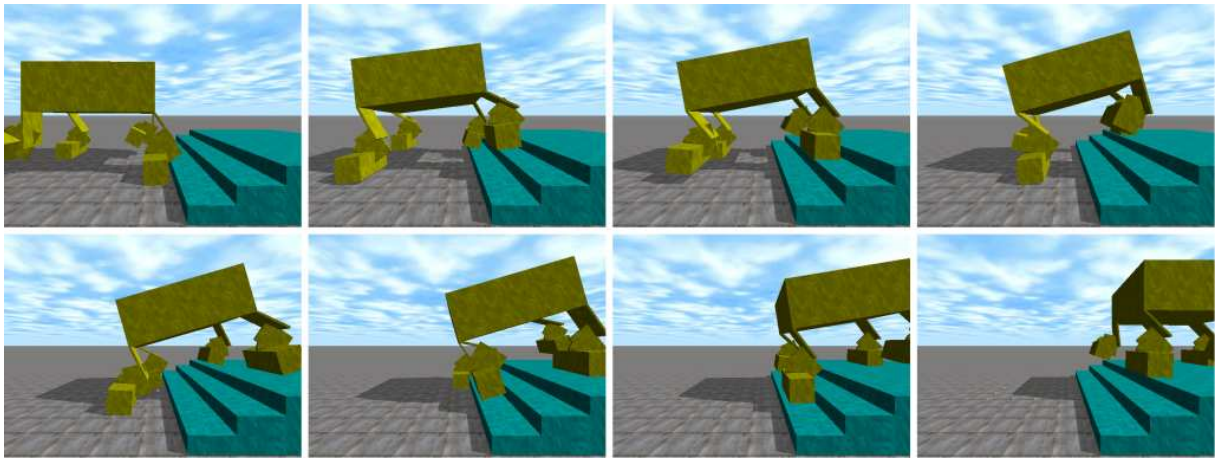


Figura 6.20: Robô evoluído para subir escadas

É importante ressaltar que o deslocamento em superfícies irregulares está fora do escopo deste trabalho, de forma que as Figuras 6.19 e 6.20 servem apenas para ilustrar as vantagens da evolução da morfologia em conjunto com os parâmetros de controle. Para que um robô possa atuar em ambientes irregulares de forma autônoma, ele precisa de informações sensoriais mais sofisticadas (visão, equilíbrio e tato) para perceber o ambiente, e assim poder planejar melhor as suas ações (HEINEN, 2002).

Outro ponto importante a destacar é que embora o TetraL3J não seja adequado para subir escadas, o controlador neural é robusto o suficiente para permitir que ele consiga executar esta tarefa de modo razoável, como mostra a Figura 6.21. O controlador neural utilizado neste exemplo foi evoluído em apenas 150 épocas e com populações de 75 indivíduos, o que demonstra que as Redes Neurais do tipo Elman são bastante indicadas para a tarefa em questão.

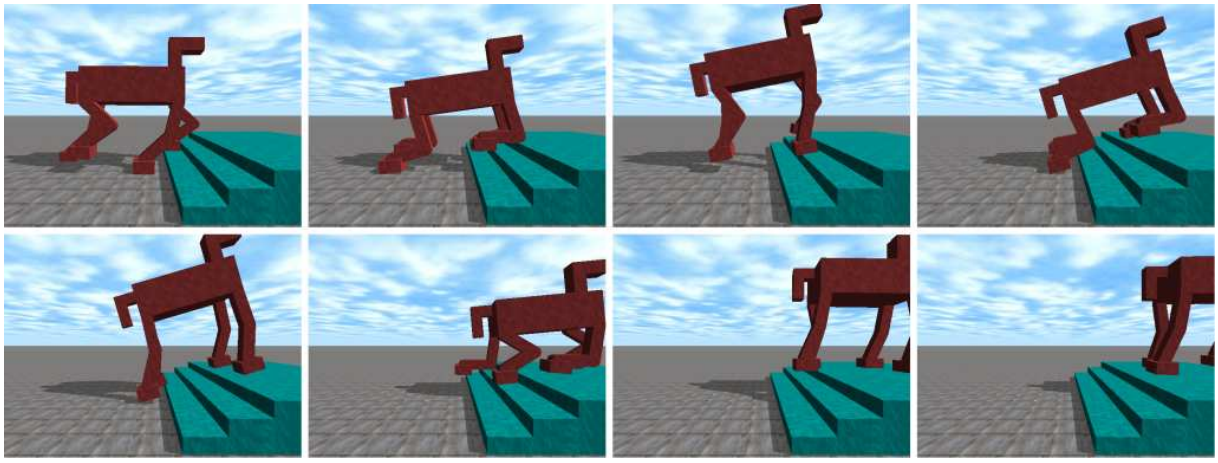


Figura 6.21: Robô TetraL3J controlado uma ANN subindo uma escada

A Figura 6.22 mostra outro modelo de robô evoluído em um ambiente irregular. Percebe-se que o GA precisou aumentar o tamanho do robô de forma que ele pudesse transpor os obstáculos, de forma similar ao que acontece na natureza em animais como os elefantes e os extintos mamutes, que graças ao seu tamanho conseguem transpor obstáculos de tamanho moderado.

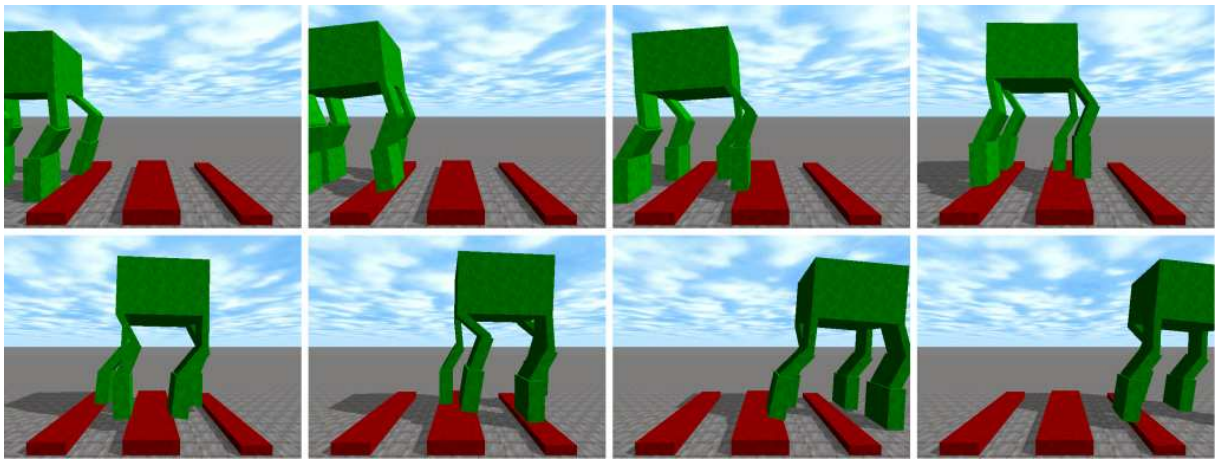


Figura 6.22: Robô especializado em terrenos irregulares

Assim, a evolução da morfologia pode ser considerada uma ferramenta bastante útil no projeto de robôs móveis autônomos, permitindo que sejam desenvolvidas soluções especialmente adaptadas para os ambientes nos quais os robôs irão atuar (PFEIFER; SCHEIER, 1999).

7 Conclusões e perspectivas

O objetivo desta dissertação foi propor, testar e avaliar o uso de técnicas de aprendizado de máquina (ML) no controle do caminhar de robôs com pernas. Para que este objetivo fosse atingido, foram pesquisadas diversas técnicas de ML, como as Redes Neurais Artificiais (ANN), os Algoritmos Genéticos (GA) e o método de Powell (*Powell's direction set*). Além disto, uma extensa pesquisa de técnicas do estado da arte foi realizada, na qual foram investigados diversos trabalhos envolvendo robôs reais e simulados de duas, quatro, seis ou mais pernas, além de alguns trabalhos na área de vida artificial.

Esta pesquisa permitiu a elaboração de quatro modelos de controle, que foram implementados através de um protótipo que simula o caminhar de robôs com pernas em um ambiente virtual, implementado com o uso da biblioteca ODE. Todos estes modelos tiveram seus parâmetros configurados de forma automática através de algoritmos genéticos, implementados utilizando a biblioteca GAlib. Os experimentos realizados mostraram que as técnicas de controle propostas são bastante eficientes, onde foram descritas de forma clara e objetiva as vantagens e desvantagens de cada uma delas.

Além dos modelos de controle, a partir da função de *fitness* usual que media apenas a distância percorrida, foram propostas e validadas outras três funções de *fitness*, que utilizam informações sensoriais (giroscópios e *bumpers*) para acelerar a evolução e permitir que os robôs se desloquem de forma mais estável. O modelo proposto também permitiu a experimentação utilizando modelos de robôs com quatro e seis pernas, com duas ou três juntas em cada perna, de forma a verificar qual modelo seria mais viável de ser construído futuramente. Além disso, em alguns experimentos foi realizada a evolução da morfologia em conjunto com os parâmetros de controle, de forma a permitir que fossem descobertos modelos de robôs mais eficientes do que os originalmente propostos.

Assim, as principais contribuições deste trabalho que merecem ser destacadas são:

- A realização de uma extensa pesquisa bibliográfica de técnicas do estado da arte no controle inteligente do caminhar de robôs com pernas;
- A proposta de um *framework* para o estudo e simulação de técnicas de controle autônomo de robôs articulados com pernas;
- A implementação de um protótipo capaz de simular robôs articulados em um ambiente virtual realístico 3D (baseado em simulação física);
- A elaboração e a validação de funções de *fitness* que se mostraram superiores as utilizadas na maiorias dos trabalhos pesquisados;
- A elaboração de quatro estratégias de controle, inspiradas em técnicas do estado da arte, e validação das mesmas através de diversos experimentos;
- A realização de um estudo comparativo entre quatro modelos de robôs distintos, permitindo uma avaliação de sua estabilidade e complexidade (relacionada diretamente ao

custo de sua implementação em hardware);
- A realização de experimentos avaliando a evolução da morfologia do robô juntamente com a evolução do mecanismo de controle.

Como perspectivas futuras, pode-se destacar as seguintes possibilidades:

- A construção de robôs reais, de forma a validar os modelos de controle de forma realista;
- A utilização de robôs bípedes, que representa um desafio muito maior devido as dificuldades de se manter a estabilidade dinâmica durante o caminhar (HEINEN, 2006; HEINEN; OSÓRIO, 2006g);
- O deslocamento em terrenos irregulares: embora em alguns experimentos tenham sido utilizados ambientes com degraus e escadas, este não foi o foco deste trabalho. Para que um robô possa realmente se deslocar de forma segura nestes tipos de ambientes, são necessárias informações sensoriais (câmeras de vídeo, sonar, infravermelho, inclinômetros, etc) para se planejar os movimentos em tempo real, bem como *endpoins* mais elaborados que possam se moldar as irregularidades do terreno;
- Utilização de uma arquitetura de controle híbrida, similar a descrita por Heinen (2002), que permita a navegação autônoma de robôs com pernas em ambientes parcialmente desconhecidos.

É importante destacar que os dois últimos ítens ainda são foco de extensa pesquisa, de forma que ainda serão necessários muitos estudos até que se consiga desenvolver um robô que consiga se deslocar de forma similar aos seres vivos em terrenos irregulares e desconhecidos, e que se adapte às mudanças do ambiente em tempo real.

ANEXO A – Experimentos exploratórios

Tabela A.1: Experimentos realizados para a definição do número de estados do autômato

	4 estados			**6 estados**			**8 estados**			**10 estados**		
E	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*
1	12,78	25,8	0,101	8,12	13,4	0,065	7,92	14,4	0,080	0,76	1,6	0,016
2	12,41	18,5	0,048	8,07	17,7	0,118	0,88	1,9	0,019	1,00	2,2	0,029
3	12,03	21,1	0,073	6,06	18,7	0,207	0,94	2,1	0,030	0,72	1,5	0,012
4	13,18	29,5	0,123	7,87	14,3	0,080	0,78	1,6	0,009	1,08	2,4	0,032
5	13,73	25,6	0,085	12,02	18,0	0,048	1,01	2,2	0,027	5,00	9,9	0,096
6	13,08	23,2	0,076	11,70	18,8	0,060	0,98	2,1	0,025	1,07	2,4	0,036
7	13,83	23,7	0,071	9,11	23,6	0,159	1,43	3,2	0,047	0,78	1,6	0,011
8	13,22	23,4	0,076	0,93	2,1	0,030	8,67	16,4	0,089	0,99	2,1	0,016
9	8,23	19,1	0,132	2,63	5,6	0,111	3,89	7,2	0,068	0,92	1,9	0,015
10	12,50	21,0	0,067	12,70	19,3	0,052	0,88	1,9	0,028	6,61	16,2	0,137
μ	**12,50**	**23,1**	**0,085**	**7,92**	**15,1**	**0,093**	**2,74**	**5,3**	**0,042**	**1,89**	**4,2**	**0,040**
σ	**1,60**	**3,3**	**0,026**	**3,88**	**6,6**	**0,056**	**3,08**	**5,6**	**0,027**	**2,10**	**4,9**	**0,042**

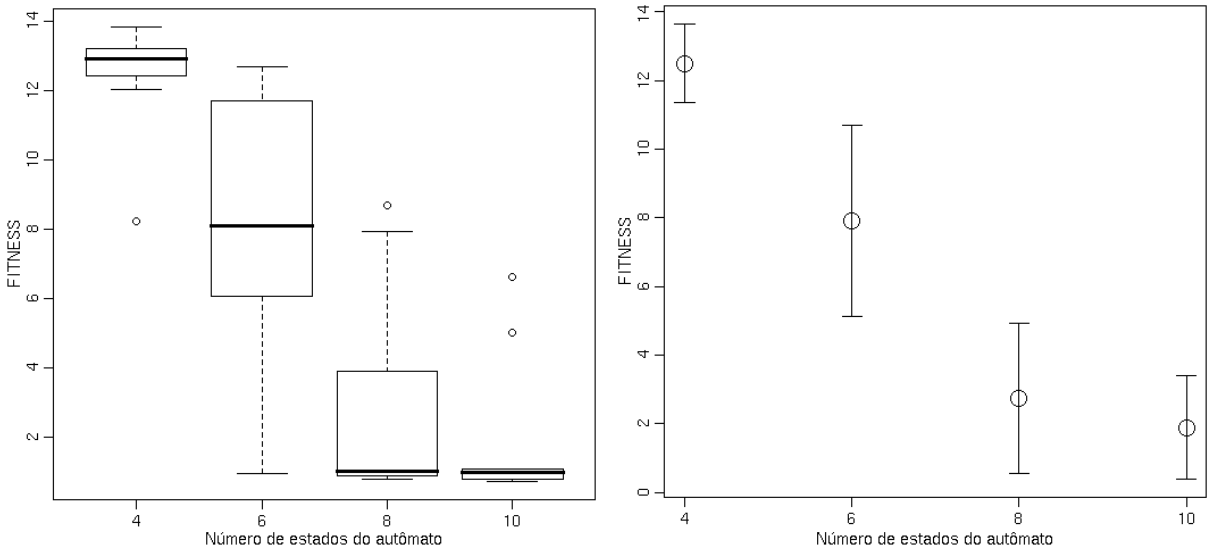


Figura A.1: Gráfico de *boxplot* e CI dos experimentos da Tabela A.1

Tabela A.2: Experimentos realizados para a definição do número de neurônios ocultos

	2 ocultos			**3 ocultos**			**4 ocultos**			**5 ocultos**		
E	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*	*F*	*D*	*G*
1	3,44	6,6	0,075	5,51	16,9	0,206	3,68	7,8	0,109	3,41	9,0	0,164
2	6,08	10,8	0,078	6,91	18,1	0,160	4,85	15,6	0,218	2,03	3,2	0,036
3	0,98	1,4	0,040	5,97	18,9	0,215	2,23	6,7	0,198	2,00	3,6	0,062
4	2,56	4,3	0,045	2,40	4,0	0,044	3,67	10,3	0,177	2,58	6,8	0,160
5	8,90	18,7	0,110	5,24	16,3	0,211	6,88	16,4	0,138	3,91	10,3	0,157
6	1,83	3,0	0,041	3,99	13,9	0,236	2,82	7,3	0,158	4,71	15,5	0,228
7	2,64	5,0	0,070	7,64	15,4	0,101	5,90	18,0	0,204	2,58	3,9	0,038
8	5,88	12,4	0,110	7,17	16,6	0,130	2,76	4,4	0,039	4,07	9,3	0,127
9	3,07	9,7	0,214	5,64	15,0	0,165	4,02	18,3	0,351	3,29	13,7	0,312
10	1,04	1,4	0,037	2,75	4,3	0,036	2,58	5,1	0,081	2,76	4,1	0,040
μ	**3,64**	**7,3**	**0,082**	**5,32**	**13,9**	**0,151**	**3,94**	**11,0**	**0,167**	**3,13**	**7,9**	**0,132**
σ	**2,54**	**5,5**	**0,054**	**1,79**	**5,4**	**0,071**	**1,52**	**5,5**	**0,086**	**0,90**	**4,4**	**0,092**

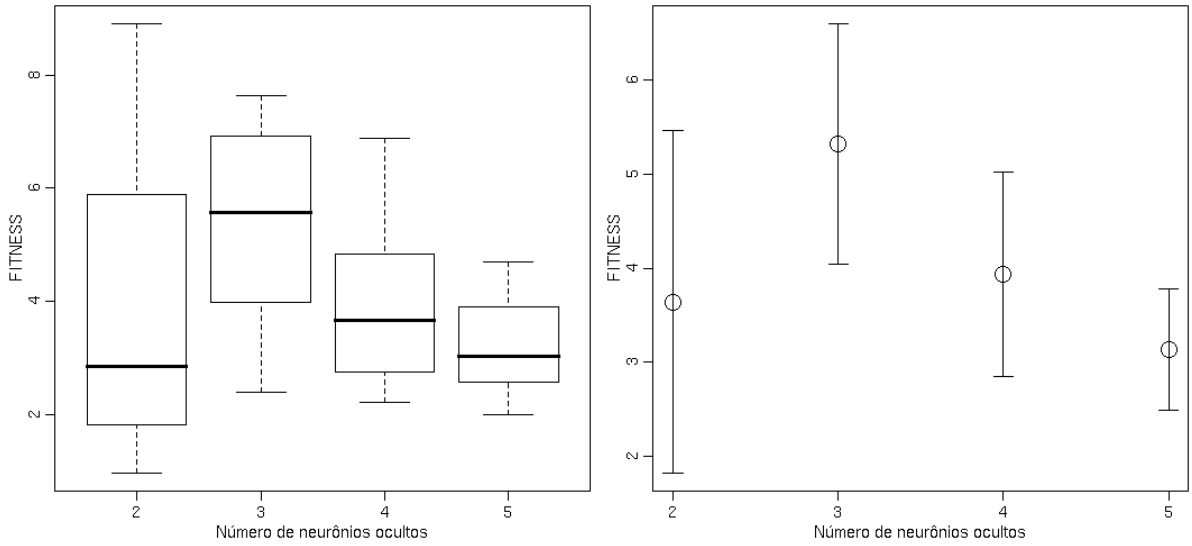


Figura A.2: Gráfico de *boxplot* e CI dos experimentos da Tabela A.2

Observação: os parâmetros do Algoritmo Genético utilizados nos experimentos exploratórios são os mesmos mostrados na Tabela6.2, com exceção do tamanho da população e do número de gerações, que foram reduzidos respectivamente para 75 e 150.

ANEXO B – Arquivos do LegGen

B.1 Exemplo de arquivo de parâmetros

```
# Arquivo de parâmetros do simulador LegGen
population_size 75
number_of_generations 150
mutation_probability 0.08
crossover_probability 0.8
replacement_percentage 1.0
random_seed 0
score_frequency 5
time_simulation 30.0
number_states 4
weights_init 0.1
weights_interval 1.0
hidden_neurons 3
time_simulation 30.0
```

B.2 Exemplo de arquivo de definição do robô

```
# Arquivo de definição do robô TetraL3J
 4 3 T S              # Numero de pernas e de partes por perna
 4.50 2.50 1.50       #  Dimensoes do corpo do robô (x, y e z)
 0.00 0.00 4.25       #   Posicoes do corpo do robô (x, y e z)
 0.85 0.90 0.50 0.50 0.50 1.50 0.50 0.50 1.50   # Tam. perna 1
 0.85 0.90 0.50 0.50 0.50 1.50 0.50 0.50 1.50   # Tam. perna 2
 0.85 0.90 0.50 0.50 0.50 1.50 0.50 0.50 1.50   # Tam. perna 3
 0.85 0.90 0.50 0.50 0.50 1.50 0.50 0.50 1.50   # Tam. perna 4
-1.5708 0.7854 0.0 2.0944 -1.0472 0.2618  # Restricoes perna 1
-1.5708 0.7854 0.0 2.0944 -1.0472 0.2618  # Restricoes perna 2
-1.5708 0.7854 0.0 2.0944 -1.0472 0.2618  # Restricoes perna 3
-1.5708 0.7854 0.0 2.0944 -1.0472 0.2618  # Restricoes perna 4
```

B.3 Exemplos de arquivos de controle evoluídos

B.3.1 Tabela posições de um autômato

```
 4 3 4
 0.830000   1.560000   1.600000   1.560000   1.600000
-0.916298  -0.305433  -0.436332  -0.654498
 1.090831   1.178097   1.003564   1.047198
-0.174533  -0.872665  -0.567232  -0.392699
-0.436332  -0.654498  -0.916298  -0.305433
 1.003564   1.047198   1.090831   1.178097
-0.567232  -0.392699  -0.174533  -0.872665
-0.436332  -0.654498  -0.916298  -0.305433
 1.003564   1.047198   1.090831   1.178097
-0.567232  -0.392699  -0.174533  -0.872665
-0.916298  -0.305433  -0.436332  -0.654498
 1.090831   1.178097   1.003564   1.047198
-0.174533  -0.872665  -0.567232  -0.392699
```

B.3.2 Parâmetros da meia-elipse

```
 0.65250379
 0.58398241
-2.94717908
 0.25622186
 1.71372545
 3.90693760
```

B.3.3 Pesos sináticos da rede neural

```
4 3 4 40
-0.0310  0.0664 -0.5168 -0.0031 -0.0627 -0.0740  0.0158  0.0320
-0.0167  0.0086  0.0999  0.0455 -0.0439 -1.0000  0.0185  0.0306
-0.0461  0.0869 -0.0553 -0.0749  0.0527  0.0018  0.0970  0.0224
 0.0024  1.0000  0.8476 -0.0655
 0.0392  0.0736 -0.0591  0.0890
 0.0324  0.0653  0.0763  0.0256
-0.0536 -0.0745 -0.0054 -0.0806
```

ANEXO C – Relório de aprendizado

```
Iniciando evolução artificial...
FIT:  0.387032  DESL:  0.947709  BUMP: 0.936906  INST: 0.051175
FIT:  0.418318  DESL:  2.146380  BUMP: 0.893005  INST: 0.323797
FIT:  3.519143  DESL:  8.333686  BUMP: 0.004222  INST: 0.136388
FIT:  4.124235  DESL: 12.418003  BUMP: 0.022625  INST: 0.198836
GEN:   10  MAX:  4.124235  MED: 2.367349  DEV: 0.65417  0:00:48
FIT:  4.208877  DESL: 12.427189  BUMP: 0.014300  INST: 0.193831
FIT:  5.798768  DESL: 14.807562  BUMP: 0.017892  INST: 0.153568
FIT:  6.622537  DESL: 11.108475  BUMP: 0.006564  INST: 0.067081
FIT:  6.923456  DESL: 14.673038  BUMP: 0.012872  INST: 0.110645
FIT:  7.101171  DESL: 11.452846  BUMP: 0.007028  INST: 0.060578
FIT:  8.795404  DESL: 17.174117  BUMP: 0.003236  INST: 0.094939
GEN:  20  MAX:  8.795403  MED:  6.009120  DEV: 0.64389  0:02:29
FIT:  9.074249  DESL: 17.087669  BUMP: 0.003025  INST: 0.088007
FIT:  9.103486  DESL: 18.374318  BUMP: 0.003122  INST: 0.101526
FIT:  9.183908  DESL: 17.111019  BUMP: 0.002650  INST: 0.086050
FIT:  9.257615  DESL: 17.200848  BUMP: 0.003353  INST: 0.085467
FIT:  9.730050  DESL: 18.527956  BUMP: 0.001206  INST: 0.090299
GEN:  30  MAX:  9.730050  MED:  8.899470  DEV: 0.23374  0:04:10
FIT:  9.777045  DESL: 18.536171  BUMP: 0.001281  INST: 0.089461
GEN:  40  MAX:  9.777045  MED:  9.433713  DEV: 0.22214  0:05:49
FIT:  9.848270  DESL: 18.543285  BUMP: 0.001469  INST: 0.088143
FIT:  9.951559  DESL: 20.166532  BUMP: 0.005172  INST: 0.102130
FIT:  9.960310  DESL: 19.110909  BUMP: 0.000906  INST: 0.091780
FIT: 10.485287  DESL: 22.413254  BUMP: 0.004406  INST: 0.113319
GEN:  50  MAX: 10.485287  MED:  9.794024  DEV: 0.10488  0:07:28
FIT: 10.527201  DESL: 22.423369  BUMP: 0.004328  INST: 0.112571
FIT: 10.924627  DESL: 23.361281  BUMP: 0.005583  INST: 0.113282
GEN:  60  MAX: 10.924627  MED: 10.277435  DEV: 0.27484  0:08:50
FIT: 11.102284  DESL: 23.397579  BUMP: 0.004736  INST: 0.110272
GEN:  70  MAX: 11.102284  MED: 10.840046  DEV: 0.16645  0:09:47
GEN:  80  MAX: 11.102284  MED: 11.006982  DEV: 0.07154  0:10:38
FIT: 11.338353  DESL: 23.503145  BUMP: 0.004786  INST: 0.106810
FIT: 11.412980  DESL: 23.498551  BUMP: 0.003128  INST: 0.105580
GEN:  90  MAX: 11.412980  MED: 11.129711  DEV: 0.09882  0:11:26
GEN: 100  MAX: 11.412980  MED: 11.329378  DEV: 0.12732  0:12:08
Tempo total de aprendizado: 00:12:24
```

ANEXO D – *"Bloopers"*

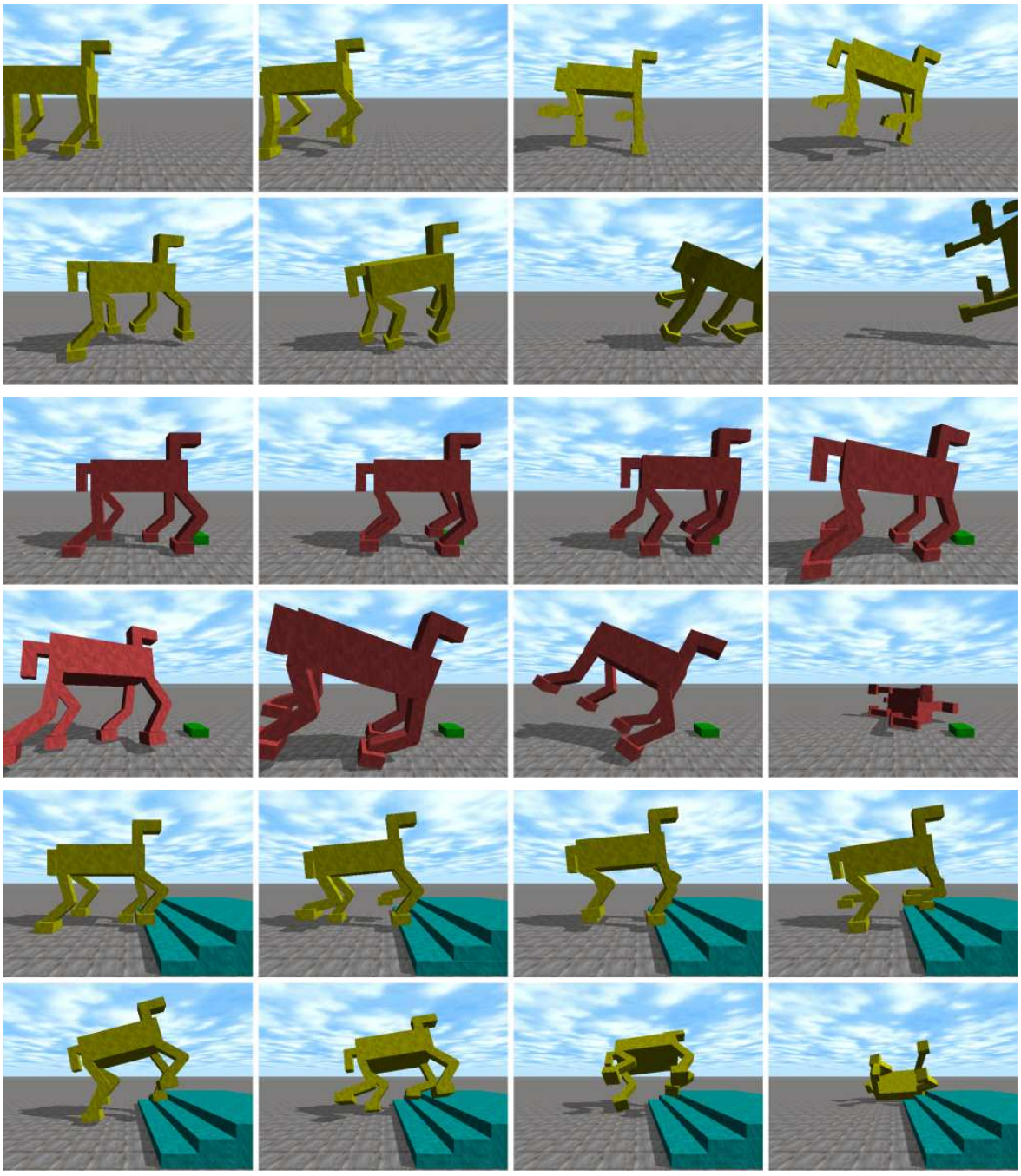


Figura D.1: Imagens que demonstram o realismo físico das simulações

Referências Bibliográficas

ACTON, Forman S. *Numerical Methods that Work*. New York, USA: Harper and Row, 1970. 235 p.

ASADA, Minoru; KATOH, Yutaka; OGINO, Masaki; HOSODA, Koh. A humanoid approaches to the goal - reinforcement learning based on rhythmic walking parameters. In: POLANI, Daniel; BROWNING, Brett; BONARINI, Andrea; YOSHIDA, Kazuo (Ed.). *Proceedings of the 7th International RoboCup Symposium*. Padua, Italy: Springer-Verlag, 2003. (Lecture Notes in Computer Science, v. 3020), p. 344–354.

BACK, Thomas. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, USA: Oxford University Press, 1996.

BARAFF, David; WITKIN, Andrew. *Physically Based Modeling: Principles and Practice*. Pittsburgh, CA, USA, 1997.

BATAVIA, Parag; POMERLEAU, Dean; THORPE, Charles. *Applying Advanced Learning Algorithms to ALVINN*. Pittsburgh, PA, USA, 1996.

BEER, Randall D. On the dynamics of small continuous-time recurrent neural networks. *Adapt. Behav.*, v. 3, n. 4, p. 469–509, 1995.

BEKEY, George A. *Autonomous Robots: From Biological Inspiration to Implementation and Control*. Cambridge, MA, USA: The MIT Press, 2005. 577 p.

BOEING, Adrian; HANHAM, Stephen; BRäUNL, Thomas. Evolving autonomous biped control from simulation to reality. In: MASSEY UNIVERSITY. *Proceedings of the 2nd International Conference on Autonomous Robots and Agents (ICARA)*. Palmerston North, New Zealand: IEEE Press, 2004. p. 440–445.

BONGARD, Josh C; PFEIFER, Rolf. A method for isolating morphological effects on evolved behaviour. In: *Proceedings of the 7th International Conference on Simulation of Adaptive Behaviour (SAB)*. Edinburgh, UK: The MIT Press, 2002. p. 305–311.

BONNASSE-GAHOT, Laurent. *Using Genetic Algorithms to Evolve Locomotion in Artificial Creatures*. Paris, France, jul. 2005.

BORENSTEIN, Johann; EVERETT, H R; FENG, Liqiang; WEHE, David K. Mobile robot positioning: Sensors and techniques. *Journal of Robotic Systems*, v. 14, n. 4, p. 231–249, 1997.

BRAGA, Antônio de Pádua; LUDERMIR, Teresa Bernarda; CARVALHO, André Carlos Ponde de Leon Ferreira. *Redes Neurais Artificiais - Teoria e Aplicações*. Rio de Janeiro, Brazil: LTC Editora, 2000.

BRENT, Richard P. *Algorithms for Minimization without Derivatives*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1973.

BROOKS, Rodney A. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, IEEE Press, v. 2, n. 1, p. 14–23, mar. 1986.

BUCHLI, Jonas; IJSPEERT, Auke Jan. Distributed central pattern generator model for robotics application based on phase sensitivity analysis. In: IJSPEERT, Auke Jan; MURATA, Masayuki; WAKAMIYA, Naoki (Ed.). *Biologically Inspired Approaches to Advanced Information Technology: First International Workshop, BioADIT 2004*. Lausanne, Switzerland: Springer-Verlag Berlin Heidelberg, 2004. (Lecture Notes in Computer Science, v. 3141), p. 333–349.

BUSCH, Jens; ZIEGLER, Jens; AUE, Christian; ROSS, Andree; SAWITZKI, Daniel; BANZHAF, Wolfgang. Automatic generation of control programs for walking robots using genetic programming. In: LUTTON, Evelyne; FOSTER, James A.; MILLER, Julian; RYAN, Conor; TETTAMANZI, Andrea G. B. (Ed.). *Proceedings of the 5th European Conference on Genetic Programming (EuroGP)*. Kinsale, Ireland: Springer-Verlag, 2002. (Lecture Notes in Computer Science, v. 2278), p. 258–267.

CHERNOVA, Sonia; VELOSO, Manuela. An evolutionary approach to gait learning for four-legged robots. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sendai, Japan: IEEE Press, 2004.

CRAIG, John J. *Introduction to Robot Mechanics and Control*. 2. ed. Reading, MA, USA: Addison-Wesley, 1989.

DARWIN, Charles. *Origin of Species*. London, UK: John Murray, 1859.

DE JONG, Kenneth A. *An Analysis of the Bahavior of a Class of Genetic Adaptative Systems*. Tese (Doctoral Thesis) — University of Michigan, Ann Arbor, MI, USA, 1975.

DUDEK, Gregory; JENKIN, Michael. *Computational Principles of Mobile Robotics*. Cambridge, UK: Cambridge University Press, 2000.

EDELMAN, Gerald M. *Neural Darwinism: The Theory of Neuronal Group Selection*. 1. ed. New York, USA: Basic Books, 1987. 371 p.

EGGENBERGER, Peter. Cell interactions as a control tool of developmental processes for evolutionary robotics. In: MAES, P; MATARIC, M; MEYER, J A; POLLACK, J; WILSON, S W (Ed.). *From Animal to Animats: Proceedings of the 4th International Conference on Simulation of Adaptative Behaviour*. Cambridge, MA, USA: The MIT Press, 1996. p. 440–448.

EGGENBERGER, Peter. Evolving morphologies of simulated 3d organisms based on differential gene expression. In: HUSBANDS, P; HARVEY, I (Ed.). *Proceedings of the Fourth European Conference on Artificial Life (ECAL'97)*. Cambridge, MA, USA: The MIT Press, 1997. p. 205–213.

EKEBERG, Orjan. A combined neuronal and mechanical model of fish swimming. *Biological Cybernetics*, v. 69, p. 363–274, 1993.

ELMAN, Jeffrey L. Finding structure in time. *Cognitive Science*, v. 14, p. 179–211, 1990.

ENDO, Ken; MAENO, Takashi; KITANO, Hiroaki. Co-evolution of morphology and walking pattern of biped humanoid robot using evolutionary computation: Evolutionary designing method and its evaluation. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE Press, 2003. p. 340–345.

FISHER, Ronald Aylmer. *The Genetic Theory of Natural Selection*. Cambridge, MA, USA: Cambridge Univ. Press, 1958.

FLORIAN, Razvan V. Autonomous artificial intelligent agents. *Cluj-Napoca*, Romania, fev. 2003.

FUJITA, M. Aibo: Toward the era of digital creatures. *International Journal of Robotics Research*, v. 20, n. 10, p. 781–794, out. 2001.

GOLDBERG, David E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989.

GOLUBOVIC, Dragos; HU, Huosheng. A hybrid evolutionary algorithm for gait generation of sony legged robots. In: *Proceedings of the 28th Annual Conference of the IEEE Industrial Electronics Society (IECON)*. Sevilla, Spain: IEEE Press, 2002.

GOLUBOVIC, Dragos; HU, Huosheng. Ga-based gait generation of sony quadruped robots. In: INTERNATIONAL ASSOCIATION OF SCIENCE AND TECHNOLOGY FOR DEVELOPMENT (IASTED). *Proceedings of the 3th IASTED International Conference on Artificial Intelligence and Applications (AIA)*. Benalmadena, Spain, 2003.

GRILLNER, Stein. Control of locomotion in bipeds, tetrapods and fish. In: BROOKS, V B (Ed.). *Handbook of Physiology, The Nervous System, 2, Motor Control*. 3. ed. Bethesda, MD, USA: American Physiology Society, 1981. p. 1179–1236.

GRILLNER, Stein. Neurobiological bases of rhythmic motor acts in vertebrates. *Science*, v. 228, n. 4696, p. 143–149, 1985.

HAYKIN, Simon. *Redes Neurais: Princípios e Prática*. 2. ed. Porto Alegre, RS, Brazil: Bookman, 2001.

HEINEN, Farlei José. *Robótica Autônoma: Integração entre Planificação e Comportamento Reativo*. São Leopoldo, RS, Brazil: UNISINOS Editora, 1999.

HEINEN, Farlei José. *Sistema de Controle Híbrido para Robôs Moveis Autônomos*. Dissertação (Master's Thesis - Applied Computing) — Universidade do Vale do Rio dos Sinos (UNISINOS), São Leopoldo, RS, Brazil, 2002.

HEINEN, Farlei José; CECHIN, Adelmo; WAGNER, Adiléia; SOUTO, Antônio. Simulação de manipuladores e modelagem através de redes neurais. In: UNIVERSIDADE REGIONAL DE BLUMENAU (FURB). *Anais do X Seminco*. Blumenau, SC, Brazil: FURB Editora, 2001. p. 11–24.

HEINEN, Farlei José; OSÓRIO, Fernando Santos. HyCAR - a robust hybrid control architecture for autonomous robots. In: SOFT COMPUTING SYSTEMS. *Proceedings of the Hybrid Intelligent Systems (HIS)*. Santiago, Chile: IOS Press, 2002. v. 87, p. 830–840.

HEINEN, Milton Roberto. *Controle Inteligente de Robôs Móveis Simulados*. 89 p. Dissertação (Master's Thesis Proposal) — Universidade do Vale do Rio dos Sinos (UNISINOS), São Leopoldo, RS, Brazil, 2006.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Autenticação de assinaturas utilizando algoritmos de aprendizado de máquina. In: CONGRESSO DA SBC. *Anais do V ENIA*. São Leopoldo, RS, Brazil, 2005.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Algoritmos genéticos aplicados ao problema de roteamento de veículos. *Hífen*, Uruguaiana, RS, Brazil, v. 30, n. 58, p. 89–96, 2006.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Applying genetic algorithms to control gait of physically based simulated robots. In: IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE (WCCI). *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. Vancouver, Canada: IEEE Press, 2006b.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Evolução do caminhar de robôs móveis simulados utilizando algoritmos genéticos. In: GRAHL, E A; HUBNER, J F (Ed.). *Anais do XV Seminário de Computação (Seminco)*. Blumenau, SC, Brazil: FURB Editora, 2006c. p. 131–142.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Gait control generation for physically based simulated robots using genetic algorithms. In: SCHIMAN, Jaime; COELHO, Helder; REZENDE, Solange Oliveira (Ed.). *Proceedings of the International Joint Conference 2006, 10th Ibero-American Conference on AI (IBERAMIA), 18th Brazilian Symposium on AI (SBIA)*. Ribeirão Preto - SP, Brazil: Springer-Verlag, 2006d. (Lecture Notes in Computer Science).

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Handwritten signatures authentication using artificial neural networks. In: IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE (WCCI). *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*. Vancouver, Canada: IEEE Press, 2006e.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Neural networks applied to gait control of physically based simulated robots. In: CANUTO, Anne M. P.; SOUTO, Marcilio C. P. de; SILVA, Antonio C. Roque da (Ed.). *Proceedings of the International Joint Conference 2006, 9th Brazilian Neural Networks Symposium (SBRN)*. Ribeirão Preto, SP, Brazil: IEEE Press, 2006f.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Robôs bípedes: O estado da arte. In: ULBRA. *Anais do V Seminário de Informática – SEMINFO RS'2006*. Torres, RS, Brazil: ULBRA Editora, 2006g.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Um estudo comparativo de heurísticas aplicadas ao problema de roteamento de veículos. In: UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES (URI). *Anais do IX Fórum de Tecnologia – XVI Seminário Regional de Informática (SRI)*. Santo Ângelo, RS, Brazil: SBC Editora, 2006h.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Uso de algoritmos genéticos para a configuração automática do caminhar em robôs móveis. In: UFMS, UCDB. *Anais do Encontro de Robótica Inteligente (EnRI)*. Campo grande, MS, Brazil, 2006i.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Uso de realidade virtual para a simulação do caminhar em robôs móveis. In: COSTA, Rosa Maria E M; TORI, Romero; MEIGUINS, Bianchi Serique (Ed.). *Proceedings of the VIII Symposium on Virtual Reality*. Belém, PA, Brazil: Editora CESUPA, 2006j.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Co-evolução da morfologia e controle de robôs móveis simulados utilizando realidade virtual. In: *Proceedings of the IX Symposium on Virtual and Augmented Reality – to appear*. Petrópolis, RJ, Brazil: ACM Press, 2007a.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. Evolving gait control of physically based simulated robots. *Revista de Informática Teórica e Aplicada (RITA) – to appear*, UFRGS Editora, Porto Alegre, RS, Brazil, 2007.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos; HEINEN, Farlei José. Estacionamento de um veículo de forma autônoma simulado em um ambiente tridimensional realístico. In: UNIVERSIDADE REGIONAL DE BLUMENAU (FURB). *Anais do XIV Seminário de Computação (Seminco)*. Blumenau, SC, Brazil: FURB Editora, 2005. p. 56–65.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos; HEINEN, Farlei José; KELBER, Christian. Autonomous vehicle parking and pull out using artificial neural networks. In: REZENDE, Solange Oliveira; FILHO, Antonio Carlos Roque da Silva (Ed.). *Proceedings of International Joint Conference, 10th Ibero-American Artificial Intelligence Conference, 18th Brazilian Artificial Intelligence Symposium, 9th Brazilian Neural Networks Symposium, IBERAMIA-SBIA-SBRN*. Ribeirão Preto, SP, Brazil: ICMC-USP, 2006a.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos; HEINEN, Farlei José; KELBER, Christian. Estacionamento de um veículo de forma autônoma utilizando redes neurais artificiais. In: UFMS, UCDB. *Anais do Encontro de Robótica Inteligente (EnRI)*. Campo grande, MS, Brazil, 2006b.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos; HEINEN, Farlei José; KELBER, Christian. SEVA3D: Using artificial neural networks to autonomous vehicle parking control. In: IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE (WCCI). *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*. Vancouver, Canada: IEEE Press, 2006c.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos; HEINEN, Farlei José; KELBER, Christian. Uso de realidade virtual no desenvolvimento de um sistema de controle do estacionamento de veículos autônomos. In: COSTA, Rosa Maria E M; TORI, Romero; MEIGUINS, Bianchi Serique (Ed.). *Proceedings of VIII Symposium on Virtual Reality*. Belém, PA, Brazil: Editora CESUPA, 2006d. p. 245–256.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos; HEINEN, Farlei José; KELBER, Christian. SEVA3D: Autonomous vehicles parking simulator in a three-dimensional environment. *Infocomp: Journal of Computer Science – to appear*, Lavras, MG, Brazil, 2007.

HITOMI, Kentarou; SHIBATA, Tomohiro; NAKAMURA, Yataka; ISHII, Shin. On-line learning of a feedback controller for quasi-passive-dynamic walking by a stochastic policy gradient method. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Alberta, Canada: IEEE Press, 2005. p. 1923–1928.

HOLLAND, John H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press, 1975.

HOLLAND, Owen E; SNAITH, Martin A. Neural control of locomotion in a quadrupedal robot. *IEE Proceedings Part F: Radar and Signal Processing*, v. 139, n. 6, p. 431–436, dez. 1992.

HOOPER, Scott L. Central pattern generators. *Embryonic ELS*, Macmillan Publishers Ltd, Hampshire, England, n. 32, p. 1–12, jun. 2001.

JACOB, David; POLANI, Daniel; NEHANIV, Chrystopher L. Legs than can walk: Embodiment-based modular reinforcement learning applied. In: *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*. Espoo, Finland: IEEE Press, 2005. p. 365–372.

JORDAN, Michael I. Attractor dynamics and parallelism in a connectionist sequential machine. In: COGNITIVE SCIENCE SOCIETY (CSS). *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Englewood Cliffs, NJ, USA: Erlbaum Associates, 1986. p. 531–546.

KELBER, C; JUNG, C R; OSÓRIO, Fernando Santos; HEINEN, Farlei Jose. Electrical drives in intelligent vehicles: Basis for active driver assistance systems. In: *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE)*. Dubrovnik, Croatia: IEEE Press, 2005. v. 4, p. 1623–1628.

KNIGHT, Rob; NEHMZOW, Ulrich. *Walking Robots - A Survey and a Research Proposal*. Essex, UK, 2002.

KOHL, Nate; STONE, Peter. Machine learning for fast quadrupedal locomotion. In: AMERICAN ASSOCIATION FOR ARTIFICIAL INTELLIGENCE (AAAI). *Proceedings of the 19th National Conference on Artificial Intelligence, 16th Conference on Innovative Applications of Artificial Intelligence*. San Jose, CA, USA: AAAI Press / The MIT Press, 2004a. p. 611–616.

KOHL, Nate; STONE, Peter. Policy gradient reinforcement learning for fast quadrupedal locomotion. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. New Orleans, LA, USA: IEEE Press, 2004b. p. 2619–2624.

KOHONEN, Teuvo. *Self-Organizing Maps*. 2. ed. Berlin, Germany: Springer-Verlag, 1987.

KOLODNER, Janet. Case-based reasoning. *Morgan Kaufmann Series in Representation and Reasoning*, San Mateo, CA, USA, 1993.

LANGTON, Chris. *Arificial Life: An Overview*. Cambridge, MA, USA: The MIT Press, 1995.

LAUGIER, Christian; FRAICHARD, Thierry; PAROMTCHIK, I E; GARNIER, Philippe. Sensor based control architecture for a car-like vehicle. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Victoria, Canada: IEEE Press, 1998. p. 165–185.

LAW, Averill M; KELTON, David W. *Simulation Modeling and Analysis*. New York, USA: McGraw-Hill, 2000. 893 p.

LEMKE, Ney; CECHIN, Adelmo; OSÓRIO, Fernando Santos; WAGNER, Adiléia; WEHRLI, Adriano V; HEINEN, Farlei José. Comparação entre algoritmos geneéticos e redes neurais aplicados ao problema da cinemática inversa. In: CONFERÊNCIA LATINO-AMERICANA DE INFORMÁTICA (CLEI). *Programas y Resumenes de INFOUYCLEI*. Montevideo, Uruguai, 2002.

LEMONICK, Michel. Dante tours the inferno. *Time Magazine - Time Domestic/Science*, v. 144, n. 7, 1994.

LEVY, Steven. *Arificial Life: A Report From the Frontier Where Computers Meet Biology*. New York, USA: Vintage books, 1992.

LINGYUM, Hu; ZENGQI, Sun. Reinforcement learning method-based stable gait synthesis for biped robot. In: *Proceedings of the 8th IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV)*. Kunming, China: IEEE Press, 2004. p. 1017–1022.

MAN, Kim-Fung; TANG, Kit-Sang; KWONG, Sam. *Genetic Algorithms: Concepts and Designs*. London, UK: Springer-Verlag, 1999. 344 p.

MARDER, Eve; CALABRESE, Ronald L. Principles of rhythmic motor pattern production. *Physiological Reviews*, v. 76, p. 687–717, 1996.

MATSUOKA, Kiyotoshi. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological Cybernetics*, v. 56, p. 345–353, 1987.

MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 115–133, 1943.

MCGEER, Tad. Passive walking with knees. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Cincinnati, OH, USA: IEEE Press, 1990. v. 2, p. 1640–1645.

MCGHEE, Robert B. Some finite state aspects of legged locomotion. *Math. Biosci.*, v. 2, p. 67–84, 1968.

MCGHEE, Robert B. Robot locomotion. *Neural Control of Locomotion*, Plenum Press, New York, USA, p. 237–264, 1976.

MEDEIROS, Adelardo. Introdução à robótica. In: 50A REUNIÃO ANUAL DA SBPC. *Anais do XVII Encontro Nacional de Automática*. Natal, RN, Brazil, 1998. v. 1, p. 56–65.

MICHEL, Olivier. Webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, v. 1, n. 1, p. 39–42, 2004.

MITCHELL, Melanie. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: The MIT Press, 1996.

MITCHELL, Tom. *Machine Learning*. New York, USA: McGrall-Hill, 1997.

MURAO, Hajime; TAMAKI, Hisashi; KITAMURA, Shinzo. Walking pattern acquisition for quadruped robot by using modular reinforcement learning. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC)*. Tucson, AZ, USA: IEEE Press, 2001. v. 3, p. 1402–1405.

MUYBRIDGE, Eadweard. *Animals in Motion*. New York, USA: Dover Publications, Inc., 1957.

NELDER, John A; MEAD, Roger. A simplex method for function minimization. *Computer Journal*, v. 7, p. 308–313, 1965.

NIKOLOPOULOS, Chris. *Expert Systems - Introduction to First and Second Generation and Hybrid Knowledge Based Systems*. New York, USA: Marcel Dekker Inc. Press, 1997.

NIKOVSKI, Daniel. Evolving legged locomotion in virtual creatures. In: *Proceedings of the Midwest Artificial Intelligence and Cognitive Science Society Conference (MAICS)*. Carbondale, IL, USA: [s.n.], 1995.

NILSSON, Nils J. *Artificial Intelligence: A New Synthesis*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1998. 536 p.

NOLFI, Stefano. Power and the limits of reactive agents. *Neurocomputing*, v. 42, n. 1-4, p. 119–145, jan. 2002.

OGINO, Masaki; KATOH, Yutaka; AONO, Masahiro; ASADA, Minoru; HOSODA, Koh. Reinforcement learning of humanoid rhythmic walking parameters based on visual information. *Advanced Robotics*, v. 18, n. 7, p. 677–697, 2004.

OSÓRIO, Fernando Santos; BITTENCOURT, João Ricardo. Sistemas inteligentes baseados em redes neurais artificiais aplicados ao processamento de imagens. In: UNIVERSIDADE DE SANTA CRUZ DO SUL (UNISC). *Anais do I Workshop de Inteligência Artificial*. Santa Cruz do Sul, RS, Brazil, 2000.

OSÓRIO, Fernando Santos; MUSSE, Soraia Raupp; VIEIRA, Renata; HEINEN, Milton Roberto; PAIVA, Daniel C. Increasing reality in virtual reality applications through physical and behavioural simulation. In: ____. *Research in Interactive Design – Proceedings of the Virtual Concept Conference 2006*. Berlin, Germany: Springer-Verlag, 2006. v. 2, p. 1–45.

PALAZZO, Luiz A M. *Algoritmos para Computação Evolutiva – Relatório Técnico (UCPel)*. Pelotas, RS, Brazil, 1997. 60-72 p.

PARKER, Gary B; BRAUN, David W; CYLIAX, Ingo. Evolving hexapod gaits using a cyclic genetic algorithm. In: HAMZA, M H (Ed.). *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC)*. Banff, Canada, 1997. p. 141–144.

PARKER, Gary B; MILLS, Jonathan W. Metachronal wave gait generation for hexapod robots. In: *Proceedings of the 3th Biannual World Automation Congress (WAC)*. Anchorange, Alaska: TSI Press, 1998.

PARKER, Gary B; RAWLINS, Gregory J E. Cyclic genetic algorithms for the locomotion of hexapod robots. In: *Proceedings of the 2st World Automation Congress (WAC)*. Montpellier, France: TSI Press, 1996.

PAROMTCHIK, Igor E; LAUGIER, Christian. Autonomous parallel parking of a nonholonomic vehicle. In: *Proceedings of the IEEE International Symposium on Intelligent Vehicles (IV)*. Tokyo, Japan: IEEE Press, 1996. p. 13–18.

PFEIFER, Rolf; IIDA, Fumiya; BONGARD, Josh. New robotics: Design principles for intelligent systems. *Artificial Life*, v. 11, n. 1-2, p. 99–120, jan. 2005.

PFEIFER, Rolf; SCHEIER, Christian. From perception to action: The right direction? In: GAUSSIER, P; NICOUD, J D (Ed.). *Proceedings of the From Perception to Action Conference*. Los Alamitos, CA, USA: IEEE Press, 1994. p. 1–11.

PFEIFER, Rolf; SCHEIER, Christian. Sensory-motor coordination: The metaphor and beyond. *Robotics and Autonomous Systems*, v. 20, n. 2-4, p. 157–178, jun. 1997.

PFEIFER, Rolf; SCHEIER, Christian. *Understanding Intelligence*. Cambridge, MA, USA: The MIT Press, 1999. 697 p.

POMERLEAU, Dean A. Neural network based autonomous navigation. *Vision and Navigation - The CMU Navlab*, Kluwer Academic Publihers, 1990.

PORTA, Josep M. *rho-Learning: A Robotics Oriented Reinforcement Learning Algorithm*. Barcelona, Spain, mar. 2000.

POWELL, Michael J D. An efficient method for finding the minimum for a function of several variables without calculating derivatives. *The Computer Journal*, v. 7, p. 155–162, 1964.

PRESS, William H; TEUKOLSKY, Saul A; VETTERLING, William T; FLANNERY, Brian P. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge, MA, USA: Cambridge University Press, 1992.

QUINLAN, John Ross. *C4.5- Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kauffman Publishers, 1993.

RAIBERT, Marc H. *Legged Robots That Balance*. Cambridge, MA, USA: The MIT Press, 1986.

RAIBERT, Marc H; BROWN, H Benjamin; CHEPPONIS, Michael A. Experiments in balance with a 3d one-legged hopping machine. *International Journal Robotics Research*, v. 3, p. 75–92, 1984.

REEVE, Richard; HALLAM, John. An analysis of neural models for walking control. *IEEE Transactions on Neural Networks*, v. 16, n. 3, p. 733–742, maio 2005.

REEVE, Richard E. *Generating Walking Behaviours in Legged Robots*. Tese (Ph.D. Thesis) — University of Edinburgh, Edinburgh, Scotland, dez. 1999.

REZENDE, Solange Oliveira. *Sistemas Inteligentes: Fundamentos e Aplicações*. Barueri, SP, Brazil: Manole Editora, 2003. 525 p.

ROBERTS, Jonathan Damien; KEE, Damien; WYETH, Gordon. Improved joint control using a genetic algorithm for a humanoid robot. In: AUSTRALIAN ROBOTICS AND AUTOMATION ASSOCIATION (ARAA). *Proceedings of the Astralasian Conference on Robotics and Automation (ACRA)*. Brisbane, Australia, 2003.

ROFER, Thomas. Evolutionary gait-optimization using a fitness function based on proprioception. In: ROBOT SOCCER WORLD CUP (ROBOCUP). *Proceedings of the 8th International Workshop on RoboCup 2004*. Erscheinen, Germany: Springer-Verlag, 2005. (Lecture Notes in Computer Science, v. 3276), p. 310–322.

ROSENBLATT, Frank. *Principles of Neurodynamics*. New York, USA: Spartan Books, 1959.

RUMELHART, David E; HINTON, Geoffrey E; WILLIAMS, Ronald J. *Learning Internal Representations by Error Propagation*. Cambridge, MA, USA: The MIT Press, 1986.

SCHEIER, Christian; LAMBRINOS, Dimitrios. Categorization in a real-world agent using haptic exploration and active perception. In: POLLACK, J. et al. (Ed.). *Proceedings of the 4th International Conference on Simulation of Adaptive Behavior on From Animals to Animats*. Cambridge, MA, USA: The MIT Press, 1996. v. 4, p. 65–74.

SCHEIER, Christian; PFEIFER, Rolf. Classification as sensory-motor coordination: A case study on autonomous agents. In: MORÁN, F. et al. (Ed.). *Proceedings of the Third European Conference on Advances in Artificial Life*. London, UK: Springer-Verlag, 1995. (Lecture Notes In Computer Science, v. 929), p. 657–667.

SCIAVICCO, Lorenzo; SICILIANO, Bruno. *Modeling and Control of Robot Manipulator*. New York, USA: McGraw-Hill, 1996.

SHAPIRO, Jonathan. Genetic algorithms in machine learning. *Advanced Summer School on Machine Learning and Applications*, 1999.

SHIMADA, Shigenobu; EGAMI, Tadashi; ISHIMURA, Kosei; WADA, Mitsuo. Neural control of quadruped robot for autonomous walking on soft terrain. In: AL, H Asama et (Ed.). *Proceeding of the 6th International Symposium on Distributed Autonomous Robotic Systems (DARS)*. Fukuoka, Japan: Springer-Verlag, 2002. (Distributed autonomous robotic systems, v. 5), p. 415–423.

SHIN, Ching Long. Analysis of the dynamics of a biped robot with seven degrees of freedom. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Berlin, Germany: IEEE Press, 1996. p. 3008–3013.

SIMS, Karl. Evolving 3d morphology and behavior by competition. In: BROOKS, Rodney A; MAES, P (Ed.). *Artificial Life IV Proceedings*. Cambridge, MA, USA: The MIT Press, 1994a. p. 28–39.

SIMS, Karl. Evolving virtual creatures. *Computer Graphics*, Association for Computing Machinery, New York, USA, v. 28, p. 15–24, jul. 1994.

SMITH, Russel. Open dynamics engine v0.5 user guide. Last Visited: 23/02/2006. fev. 2006. Disponível em: <http://www.ode.org>.

STEIN, Paul S G; GRILLNER, Sten; SELVERSTON, Allen I; STUART, Douglas G. *Neurons, Networks, and Behavior*. Cambridge, MA, USA: The MIT Press, 1997.

STONE, Henry W. Mars pathfinder microrover - a small, low-cost, low-power spacecraft. In: AMERICAN INSTITUTE OF AERONAUTICS AND ASTRONAUTICS (AIAA). *Proceedings of the AIAA Forum on Advanced Developments in Space Robotics*. Madison, WI, USA, 1996.

SUTTON, Richard S; BARTO, Andrew G. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: The MIT Press, 1998.

TAGA, Gentaro. A model of the neuro-musculo-skeletal system for human locomotion. *Biological Cybernetics*, v. 73, p. 97–111, 1995.

TE BOEKHORST, René J A; LUNGARELLA, Max; PFEIFER, Rolf. Dimensionality reduction through sensory-motor coordination. In: KAYNAK, Okyay; ALPAYDIN them; OJA, Erkki; XU, Lei (Ed.). *Proceedings of the 10th Joint International Conference on Artificial Neural Networks and Neural Information Processing (ICONIP'03)*. Istanbul, Turkey: Springer-Verlag, 2003. (Lecture Notes in Computer Science, v. 2714), p. 496–503.

TEDRAKE, Russ; ZHANG, Teresa Weirui; SEUNG, H Sebastian. Stochastic policy gradient reinforcement learning on a simple 3d biped. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Senday, Japan: IEEE Press, 2004. p. 2849–2854.

TOTH, David; PARKER, Gary. Evolving gaits for the lynxmotion hexapod ii robot. In: *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI)*. Orlando, FL, USA: [s.n.], 2003.

TSAI, Jinn-Tsong; CHOU, Jyh-Horng; LIU, Tung-Kuan. Tuning the structure and parameters of a neural network by using hybrid taguchi-genetic algorithm. *IEEE Transactions on Neural Networks*, v. 17, n. 1, p. 69–80, jan. 2006.

VAPNIK, Vladimir Naumovich. *The Nature of Statistical Learning Theory*. New York, USA: Springer-Verlag, 1995.

VAPNIK, Vladimir Naumovich. *Statistical Learning Theory*. New York, USA: Wiley, 1998.

VAUGHAN, Eric D. *Evolution of 3-Dimensional Bipedal Walking with Hips and Ankles*. Dissertação (Master's Thesis) — Sussex University, Sussex, UK, 2003.

VAUGHAN, Eric D. *Flight of the Bipped: Simulation of Adaptative Behavior*. Sussex, UK, 2003.

VAUGHAN, Eric D; DI PAOLO, Ezequiel A; HARVEY, Inman R. The evolution of control and adaptation in a 3d powered passive dynamic walker. In: POLLACK, J; BEDAU, M; HUSBANDS, P; IKEGAMI, T; WATSON, R (Ed.). *Artificial Life IX: Proceedings of the Ninth Interational Conference on the Simulation and Synthesis of Life*. Cambridge, MA, USA: The MIT Press, 2004. p. 139–145.

WAGNER, Adiléia. *Extração de conhecimento a partir de Redes Neurais treinadas aplicada ao Problema da Cinemática Inversa na Robótica*. Dissertação (Master's Thesis) — Universidade do Vale do Rio dos Sinos (UNISINOS), São Leopoldo, RS, Brazil, 2003.

WAIBEL, Alex. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, n. 1, p. 39–46, 1989.

WAN, Eric A. Temporal backpropagation for FIR neural networks. In: *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*. San Diego, CA, USA: IEEE Press, 1990. v. 1, p. 575–580.

WEINGARTEN, Joel D; LOPES, Gabriel A D; BUEHLER, Martin; GROFF, Richard E; KODITSCHEK, Daniel E. Automated gait adaption for legged robots. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. New Orleans, LA, USA: IEEE Press, 2004.

WERBOS, Paul. Backpropagation trough time: What it does and to do it. *Proceedings of the IEEE*, IEEE Press, v. 78, p. 1550–1560, 1990.

WILLIAMS, Ronald J; ZIPSER, David. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, n. 1, p. 270–280, 1989.

WOLFF, Krister; NORDIN, Peter. Evolution of efficient gait with an autonomous biped robot using visual feedback. In: UNIVERSITY OF TWENTE. *Proceedings of the Mechatronics Conference*. Enschede, The Netherlands, 2002.

WOLFF, Krister; NORDIN, Peter. Evolutionary learning from first principles of biped walking on a simulated humanoid robot. In: ADES, M; DESCHAINE, L M (Ed.). *Proceedings of the Business and Industry Symposium of the Advanced Simulation Technologies Conference (ASTC'03)*. Orlando, FL, USA: SCS, 2003a. p. 31–36.

WOLFF, Krister; NORDIN, Peter. Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming. In: KEIJZER, Maarten (Ed.). *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. Chicago, IL, USA: Springer-Verlag, 2003b. (Lecture Notes in Computer Science, v. 2723).

WYETH, Gordon; KEE, Damien; YIK, Tak Fai. Evolving a locus based gait for a humanoid robot. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE Press, 2003. v. 2, p. 1638–1643.

ZYKOV, Viktor; BONGARD, Josh; LIPSON, Hod. Evolving dynamic gaits on a physical robot. In: KEIJZER, Maarten (Ed.). *Late Breaking Papers at the Genetic and Evolutionary Computation Conference (GECCO)*. Seattle, WA, USA: Springer-Verlag, 2004. (Lecture Notes in Computer Science, v. 3102).