UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

DANIEL MONTEIRO BASSO

Arquitetura Neural para Controle da Locomoção de um Robô Quadrúpede Baseada em Referências Biológicas

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Paulo Martins Engel
Orientador

Porto Alegre, setembro de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Daniel Monteiro Basso,

Arquitetura Neural para Controle da Locomoção de um Robô Quadrúpede Baseada em Referências Biológicas /

Daniel Monteiro Basso. – Porto Alegre: PPGC da UFRGS, 2005.

55 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Paulo Martins Engel.

1. Locomoção. 2. Gerador central de padrões. 3. Redes neurais. I. Engel, Paulo Martins. II. Título.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CPG	Central Pattern Generator
GPL	General Public License
LTP	Long Term Potentiation
CTRNN	Continuous Time Recurrent Neural Network

LISTA DE FIGURAS

RESUMO

A natureza é uma fonte inesgotável de soluções elegantes para os mais diversos problemas, da aparente simplicidade do formato do ovo, que garante maior segurança ao organismo em desenvolvimento, à evidente complexidade do sistema nervoso humano, que é capaz de pensar sobre problemas e encontrar soluções para eles.

Neste trabalho foram aplicados modelos computacionais inspirados em estudos neurológicos para a realização da locomoção de um robô quadrúpede. O corpo do robô foi simulado, permitindo incorporar características mais semelhantes às de um animal do que seria possível com um robô de verdade, como grande quantidade de articulações e força dos motores. A arquitetura neural responsável pelo controle do robô teve como referência para seu desenvolvimento os Geradores Centrais de Padrões, que são circuitos neurais capazes de gerar a seqüência de ativações musculares envolvidas em diversos movimentos rítmicos, como respirar e caminhar, e estão em sua maioria localizados na medula espinhal. Para compor esta rede foi usado um modelo de neurônio com algumas propriedades encontradas em um neurônio real, como a capacidade de inibir a ativação de outros neurônios, e outras puramente matemáticas, tornando-o uma ferramenta versátil para incorporar a funcionalidade de outros modelos de neurônios e de redes neurais.

Trabalhos futuros farão uso dos resultados desta pesquisa como base para abordar temas mais complexos, como planejamento motor e tomada de decisão.

Palavras-chave: Locomoção, gerador central de padrões, redes neurais.

Neural Architecture for the Locomotion Control of a Quadruped Robot Based on Biological References

ABSTRACT

Nature is an endless source of elegant solutions to every kind of problem, from the seeming simplicity of the egg's shape, which assures a safer environment to the developing organism, to the evident complexity of the human nervous system, which is capable of thinking about problems and finding solutions to them.

In this work some computational models inspired on neurological research were applied to achieve the locomotion of a quadruped robot. The robot's body was simulated, allowing to have features more similar to their animal counterparts than would be possible using a real robot, such as the high amount of articulations and the motors' strength. The neural architecture responsible for the robot's control was developed using Central Pattern Generators as reference, which are neural circuits capable of generating the sequence of muscle activations involved in a variety of rhythmic movements, such as breathing and walking, and are mostly located in the spine. To assemble this network a neuron model was used which has some properties found on a real neuron, as the ability to inhibit other neurons' activation, and others purely mathematical, making it a versatile tool to incorporate the functionality of other neuron and neural networks models.

Future works will make use of these research's results as a base to approach more complex problems, such as motor planning and decision making.

Keywords: Locomotion, central pattern generators, neural networks.

1 INTRODUÇÃO

A computação pode ser definida como a manipulação de símbolos de acordo com regras fixas, ou, menos abstratamente, como o processo de busca de uma solução pelo uso de um algoritmo. Qualquer sistema físico, inclusive o próprio universo, pode ser considerado um computador, embora a semântica dada aos estados do sistema e às regras da dinâmica esteja a cargo do observador.

Estamos mais acostumados a sistemas computacionais baseados na funcionalidade de componentes eletrônicos, pois seu aprofundado estudo nos possibilitou criar os mais variados dispositivos, responsáveis por grande parte do conforto que a humanidade desfruta atualmente. Estes sistemas foram projetados para resolver problemas específicos, com pouca ou nenhuma adaptabilidade a situações imprevistas.

Em um sistema dinâmico complexo, no entanto, a manutenção da integridade dos elementos que o compõem é altamente dependente da capacidade de se adaptarem ao inesperado (MATURANA, 1998). A dinâmica de nosso planeta deu origem a diversos processos computacionais que proporcionam uma maior adaptabilidade aos sistemas vivos. Um desses processos, que atua no nível da espécie, é a seleção natural de (DARWIN, 1991). Graças a ela, estruturas progressivamente mais complexas puderam se desenvolver, até chegar ao que consideramos o ápice da adaptabilidade no nível de indivíduo: o sistema nervoso animal.

Fruto de bilhões de anos de evolução, o sistema nervoso animal é munido de inúmeras técnicas de sobrevivência, cada uma resultado de diversos sistemas computacionais atuando conjuntamente. O estudo desses sistemas já proporcionou diversos avanços na forma como processamos dados de diversos tipos, e modelos aplicados em dispositivos eletrônicos atualmente são capazes de, por exemplo, identificar peças com defeitos de fabricação em uma linha de produção.

Podemos dividir as abordagens de pesquisa do sistema nervoso em duas grandes categorias: as *top-down*, em que interessa apenas o comportamento final do sistema, e modelos criados não precisam necessariamente funcionar sob os mesmos princípios do sistema analisado, bastando que exibam o mesmo comportamento

(modelos *black-box*); e as *bottom-up*, em que se espera atingir o comportamento global do sistema mimetizando-se o comportamento individual de seus componentes.

Como exemplos de disciplinas que usam a abordagem *top-down* podemos citar grande parte da Psicologia e alguns ramos da Inteligência Artificial, como Sistemas Multiagentes, Sistemas Especialistas e Processamento de Linguagem Natural. Usando métodos com pouco ou nenhum referencial biológico já foi possível criar sistemas capazes de realizar inferências lógicas e mesmo provar teoremas matemáticos complexos, algo que até há pouco tempo muitos acreditavam ser uma tarefa impossível para uma máquina. Mas o que geralmente se observa nesses sistemas é que a adição de novas capacidades é difícil e, quando possível, normalmente é uma solução deselegante. Esse é um problema típico de especificações em Engenharia de Software, para que um sistema seja coeso é necessário que todos os fatores tenham sido levados em consideração antes de que se inicie a implementação. Uma vez tendo uma estrutura pronta, alterá-la é custoso.

Os ramos da Medicina, Biologia e Física que estudam o cérebro, no entanto, partem do extremo oposto, analisando e criando modelos dos níveis mais básicos e suas inter-relações. A maior parte da compreensão que se tem do sistema nervoso se concentra nas áreas sensoras e motoras, devido à facilidade de acesso e experimentação. No entanto o conhecimento das regiões associativas, ou integradoras, vem aumentando consideravelmente com o auxílio de novas técnicas de imagem por ressonância magnética funcional.

Este trabalho faz parte de um projeto de longo prazo para a simulação de partes do sistema nervoso, usando a abordagem *bottom-up* inspirada pela evolução, isto é, seguindo intuitivamente a ordem em que os diversos subsistemas que compõem o sistema nervoso humano foram surgindo. O objetivo deste projeto é criar um robô que exiba o comportamento que se esperaria de animais que tenham os subsistemas que estejam sendo simulados. Gradualmente espera-se chegar ao nível em que o raciocínio abstrato e a linguagem possam se realizados. Até chegarmos nesse ponto, provavelmente muito tempo haverá decorrido, a performance dos processadores terá melhorado muito e mais dados sobre o funcionamento do cérebro estarão disponíveis.

Como meta, para este trabalho em específico, estabelecemos a simulação do comportamento dos circuitos neurais responsáveis por uma das atividades mais essenciais aos animais: a locomoção, a habilidade de se mover de um lugar a outro. Várias formas de se locomover evoluíram, como nadar, rastejar, voar e caminhar. Apesar de serem métodos que se realizam de maneiras fisicamente distintas, todos são movimentos rítmicos, e os circuitos neurais que os realizam são semelhantes. Escolhemos focar nosso estudo sobre a locomoção quadrúpede.

Mas para avaliar o comportamento dos sistemas simulados é necessário que haja

um corpo que interaja com um meio ambiente, e é desejável que tenha características biofísicas semelhantes às de um animal real. Os atuadores deste corpo, sejam eles motores ou músculos sintéticos, devem ter um tempo de resposta e força semelhantes ao de músculos naturais. No caso deste trabalho, o número de articulações dos membros, pelo menos, deve ser realístico. E tem de ser facilmente expansível para trabalhos futuros. Um robô com essas características é, atualmente, inviável de ser feito, e portanto foi necessário que fizéssemos uso de simulações de robôs. Há alguns pontos positivos em se usar robôs simulados: não é necessário manutenção mecânica ou elétrica; não há acoplamento entre tempo real e o tempo simulado, permitindo que experimentos sejam realizados mais rapidamente; o controle sobre os parâmetros é total, e experimentos podem ser repetidos obtendo-se exatamente os mesmos resultados, o que permite uma melhor análise dos dados.

Diversos ambientes de simulação de robôs foram investigados, no entanto nenhum pareceu apropriado, o que nos levou a criar a primeira contribuição deste trabalho, o conjunto de ferramentas para simulação robótica Phi, descrito no capítulo 2.

Quando se pensa em integrar modelos de sistemas neurais criados por grupos de pesquisa diversos, é necessário levar em consideração que cada um poderá usar tipos diferentes de neurônios como base. É muito importante que o modelo seja adaptado de forma a usar uma base comum a todo o resto do sistema, aumentando sua coesão. Esta base, portanto, deve ser genérica o suficiente para suprir as características requisitadas por cada modelo. No capítulo 3 encontra-se uma análise do estado da arte dos modelos de neurônios. Na seção 3.6 é proposto um modelo de neurônio com propriedades interessantes à integração de modelos de sistemas neurais, constituindo a segunda contribuição deste trabalho.

Os circuitos neurais que são os responsáveis pela geração dos sinais que formam a base para o controle da locomoção se chamam Geradores Centrais de Padrões. No capítulo 4 eles são descritos e é feita a análise do estado da arte dos modelos desse tipo de circuito. Na seção 4.2 é proposto um gerador de padrões que faz uso do modelo de neurônio descrito na seção 3.6, compondo a arquitetura neural que usamos para a locomoção.

No capítulo 5 os detalhes da implementação e os resultados dos experimentos práticos são analisados.

O capítulo 6 conclui com uma análise deste trabalho, e aponta a direção que trabalhos futuros tomarão.

2 FERRAMENTAS E MÉTODOS

Há inúmeros formatos de corpos dentre os mamíferos quadrúpedes, cada um adaptado às pressões que o ambiente lhe impôs, geração após geração. Alguns são mais velozes, outros têm mais destreza, e talvez o ponto médio entre essas qualidades se encontre nos felinos, motivo pelo qual os escolhemos como fonte de inspiração para nosso modelo de robô.

Não existe, até a data corrente, materiais que mimetizem as características biofísicas de músculos reais (ASHLEY, 2003). Mas mesmo admitindo-se o uso de motores, os custos materiais para a construção de um robô, que tivesse um nível de complexidade suficientemente interessante para nossa pesquisa, seriam muito elevados. Além disso, do ponto de vista da pesquisa sobre sistemas de controle, não é prático trabalhar com robôs reais, que necessitam de manutenção freqüente, interrompendo o processo investigatório. Para contornar esses problemas, é comum criar modelos matemáticos da dinâmica do robô, e simulá-los computacionalmente.

Na seção 2.1 é introduzido o ambiente de simulação no qual o comportamento de nosso robô é avaliado. Uma breve introdução a algoritmos genéticos, usados na evolução dos pesos das redes neurais mapeadoras (seção 5.2), é apresentada na seção 2.2. A seção 2.3 mostra resumidamente como um organismo vivo se movimenta pelo controle muscular, fornecendo as referências para a criação de um robô com funcionalidade similiar. As características do corpo do robô são apresentadas na seção 2.4.

2.1 O Ambiente de Simulação: *Phi Robotics Simulation Framework*

Há alguns anos não havia alternativa senão criar modelos dinâmicos *ad hoc*, pois os recursos computacionais eram severamente limitados, e deveria-se aproveitar ao máximo o tempo de execução do computador. Atualmente se dispõe de uma certa flexibilidade na relação entre facilidade de uso e eficiência, o que possibilitou que sistemas mais genéricos fossem desenvolvidos para a simulação de robôs.

Há uma grande disponibilidade de aplicações comerciais para simulação robótica hoje em dia. Muitas exigem um grau de precisão suficientemente grande para justificar o investimento de muitos recursos para sua fabricação, o que as torna inerentemente caras. É o caso de softwares desenvolvidos para modelar robôs que atuam em linhas de montagem, aplicações cirúrgicas, militares e aeroespaciais. Mas o foco da pesquisa robótica atualmente está em sistemas adaptativos, que precisam lidar com imprecisões e imprevisibilidade. Para isso pode-se usar modelos muito mais simples e eficientes, e a preocupação com a precisão pode ser relaxada.

Programas com essas especificações existem em variados preços, que vão de pouco mais de uma centena de dólares, como o caso do simulador Webots da Cyberbotics (MICHEL, 2004), a até mesmo centenas de milhares de dólares.

Muitos pesquisadores e entusiastas da robótica, que não dispõem dos recursos, ou que acreditam que estes devam ser empregados de forma melhor, tentam suprir suas necessidades criando seus próprios simuladores alternativos. A maior parte deles é licenciada sob a *General Public License* (GPL), para que outras pessoas possam usufruir e contribuir para seu desenvolvimento. Porém o resultado desses esforços costuma ser restrito a um domínio específico de aplicação, e geralmente são de difícil uso e expansão.

O conjunto de ferramentas para simulação robótica Phi começou como um desses projetos, e visava ser um simulador específico para futebol de robôs. Desde o início ele se caracterizou por ser um software livre, licenciado sob a GPL, e por usar apenas bibliotecas e ferramentas de desenvolvimento livres. À medida que o projeto foi evoluindo, e maior familiaridade com as bibliotecas envolvidas se estabeleceu, pôde-se perceber o quanto cada uma delas era poderosa independentemente, e como seria grande a contribuição em criar um aplicativo que usasse os recursos de todas de forma integrada.

O resultado foi um simulador suficientemente genérico para ser usado em um amplo espectro de aplicações, da pesquisa de sistemas de controle à visão computacional, do auxílio ao ensino fundamental de física a até mesmo a simples e rápida criação de animações gráficas.

2.1.1 Desenvolvimento

O projeto começou com a análise da biblioteca *Open Dynamics Engine* (ODE Webpage, 2005), de simulação da dinâmica de corpos rígidos e do cálculo de colisões, escrita originalmente por Russel Smith. Ela incorpora os principais elementos para uma simulação robótica confiável, isto é, estável e semelhante ao que ocorreria no mundo real sob as mesmas condições. A forma como foi projetada torna fácil a adição de novas funcionalidades que venham a ser necessárias. Os testes iniciais foram feitos sobre os exemplos que vinham na distribuição do ODE, que

dispõe de uma pequena biblioteca gráfica (*DrawStuff*) que encapsula a interface da biblioteca OpenGL.

Gradualmente os códigos dos exemplos foram sendo substituídos por uma estrutura um pouco mais genérica, e a visualização foi separada em uma arquitetura cliente/servidor, para que a simulação pudesse ser executada em computadores sem interface gráfica, o que facilitaria a distribuição do processamento.

Uma vez tendo uma estrutura básica para a manipulação dos elementos disponibilizados pelo ODE, partiu-se para a definição da linguagem que seria usada para a entrada de configurações e descrição dos modelos dinâmicos. A escolhida foi a linguagem de *script* Lua (LUA Webpage, 2005), desenvolvida na PUC-Rio, por (FIGUEIREDO; IERUSALIMSCHY; CELES, 1996). Seu pequeno interpretador é extremamente fácil de ser incorporado a programas escritos em C/C++, e a flexibilidade da linguagem a torna excelente tanto para a entrada de dados e configurações, quanto à programação de sistemas de simples a média complexidades. Logo ficou evidente que a combinação de ODE com Lua resultaria num sistema poderoso o suficiente para modelar dispositivos robóticos complexos, e como forma de comprovar isso, em pouco tempo um robô quadrúpede com 16 articulações foi criado.

Para melhorar a apresentação gráfica do mundo virtual sendo simulado, o renderizador por traçado de raios *Persistence of Vision Raytracer* (POVRAY Webpage, 2005) foi integrado. Isso possibilita que, com um pouco de habilidade artística, se consiga descrever cenas com um grau de detalhe suficientemente grande para a geração de imagens foto-realísticas, úteis para a pesquisa de visão computacional.

Desde o início do projeto até o estado atual, três grandes reestruturações ocorreram, cada uma tornando o uso do sistema mais fácil e menos restritivo, mas também mudando radicalmente a forma de especificar os modelos. Atualmente a linguagem de descrição das cenas está estabilizada, e desenvolvimentos futuros poderão ser apenas aditivos, mantendo a compatibilidade com versões anteriores.

2.1.2 Componentes e Fluxo de Dados

Os diversos módulos que compõem o Phi trocam dados de três formas: por ligação direta de bibliotecas, por *sockets TCP* e através do sistema de arquivos e chamadas de processos.

Na figura 2.1 se pode ver a estrutura de ligações entre os principais módulos, que são:

- libdxp: encapsula diversas bibliotecas em objetos C++, como as de *sockets*, *threads*, do interpretador Lua, entre outras.

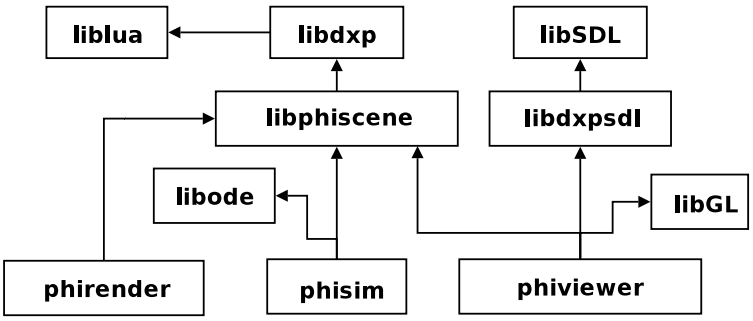

Figura 2.1: Estrutura de ligações com as bibliotecas ($X \rightarrow Y$: X depende de Y)

- libdxpsdl: biblioteca utilitária para o uso da *Simple Directmedia Layer* (SDL Webpage, 2005), que faz a abstração do hardware gráfico e do gerenciador de janelas.

- libphiscene: biblioteca responsável pela representação interna do visual da cena, e pela troca de dados entre os componentes que a usam. É capaz de gerar arquivos com o registro de toda uma simulação de forma compacta, para depois ser visualizada de outras maneiras. Também é o elemento que faz a versão da representação interna da cena em uma descrição em *scene description language* do POVRay. Implementa a arquitetura cliente/servidor usando *sockets*.

- phisim: é o aplicativo que interpreta as descrições dos modelos dinâmicos, cria e gerencia os objetos do ODE, e atua como servidor de cenas, através do uso da libphiscene. Na figura 2.2 pode-se ver como se processa a troca de informações durante a simulação: é possível manipular a cena diretamente, pelo uso de elementos apenas gráficos, ou indiretamente, aplicando forças sobre os objetos dinâmicos.

- phiviewer: usa a libphiscene para ser um cliente de cena, realizando a visualização em tempo real da simulação por OpenGL. Recebe tanto dados por *sockets*, provenientes de um processo phisim em execução, quanto por arquivos de registros. Possibilita a navegação livre pelo espaço tridimensional virtual, assim como pode seguir câmeras instaladas no ambiente estático ou nos elementos dinâmicos da cena. Provê um mecanismo (rústico, atualmente) para o envio de comandos ao simulador.

- phirender: distribui em vários computadores o processamento da geração de animações com imagens foto-realísticas, de simulações gravadas previamente em um arquivo de registro.

- phieditor: não apresentado na figura 2.1, é um utilitário para a edição de arquivos de descrição dos modelos dinâmicos. Integrado ao phisim, phiviewer e ao POVRay por chamada de processos, torna rápida a criação de cenas mais complexas. Ao contrário de todos os outros módulos, que foram escritos em C++, este aplicativo foi feito em C#, usando o compilador e bibliotecas do *mono* (www.go-mono.com) e GTK#.

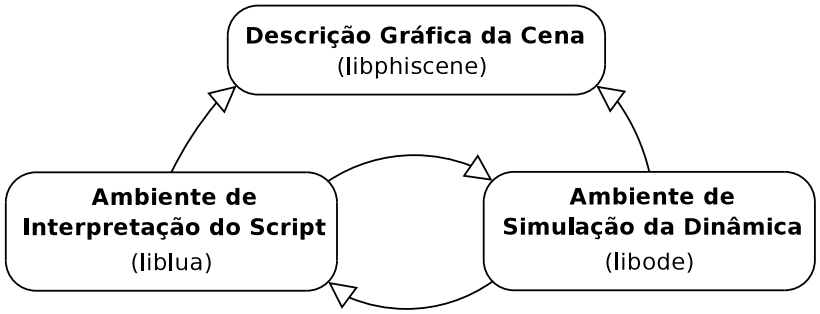

Figura 2.2: Contextos de dados independentes e o fluxo entre eles, na execução do phisim

As formas de visualização da simulação proporcionado pelo Phi, a partir de um arquivo de especificação do modelo dinâmico, são apresentadas na figura 2.3. O arquivo de especificação é interpretado pelo phisim, que pode gerar um arquivo de registro (cena.log, na figura) e/ou conectar-se com o visualizador phiviewer. O arquivo de registro pode ser visualizado posteriormente pelo phiviewer e/ou processado pelo phirender, que chama o POVRay para gerar a seqüência de imagens da simulação.

2.1.3 Elementos da Simulação

Os dados manipulados pelo simulador são, basicamente, primitivas geométricas, conexões (articulações) entre elas, suas propriedades físicas, e propriedades visuais da cena.

Atualmente, as primitivas disponíveis são: esfera, paralelepípedo, cilindro, cápsula, plano e raio; e elas podem ser combinadas para criar objetos compostos. Há três categorias de funcionalidade de primitivas:

- Corpos rígidos: participam completamente do cálculo da dinâmica, tendo sua posição e rotação influenciadas pelas colisões e outras forças aplicadas.

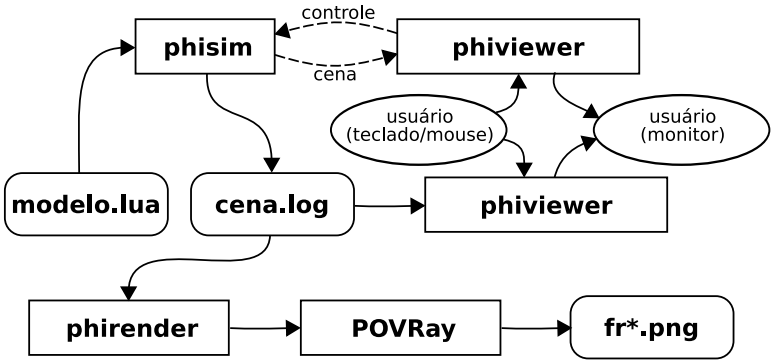

Figura 2.3: Fluxo dos dados da simulação

- Cascas de colisão: estáticas, participam parcialmente do cálculo da dinâmica, servindo como barreiras para os corpos rígidos.
- Primitivas gráficas: não participam da dinâmica, mas possibilitam a adição de elementos pertinentes à visualização, auxiliando a rápida interpretação de variáveis do experimento sendo realizado.

As propriedades físicas que afetam a dinâmica são: massa, coeficiente de atrito e coeficiente de restituição, além do formato do corpo. Há também dois parâmetros de controle do cálculo que não são reais, mas que servem para obter resultados aproximados de efeitos reais como, por exemplo, a absorção de força por uma superfície macia. Esses parâmetros são o coeficiente de mesclagem de forças e o parâmetro de redução de erro. Dentre as propriedades que não influenciam a dinâmica estão a cor, textura e acabamento dos materiais.

As conexões entre os corpos rígidos possibilitam que se criem dispositivos com funcionalidades tão diversas quanto um simples triciclo a até mesmo um complexo robô hexápode, com múltiplas articulações. Os tipos de conexões disponíveis atualmente são:

- *Slider* (deslizador): permite que a posição relativa dos corpos varie sobre um eixo (como um pistão).
- *Hinge* (dobradiça): permite que a rotação relativa dos corpos varie sobre um eixo (também é o caso das rodas).
- *Universal*: permite a rotação relativa dos corpos sobre dois eixos.
- *Ball-and-socket* (esfera-e-encaixe): permite a rotação sobre os três eixos.
- *Hinge2* (eixo-duplo): uma dobradiça com amortecimento, para orientar o eixo de rotação de uma roda. Simula a funcionalidade da roda e da suspensão de um carro.

Todos esses elementos são manipulados através de funções e métodos no *script* Lua. A forma como ocorre a integração entre o *script* e o programa *host* (o que contém o interpretador) é detalhada na seção 2.1.4.

2.1.4 A Linguagem *Script*

Toda comunicação entre o *script* e o programa *host* se faz através de funções e métodos, que foram implementados em C++ e registrados no interpretador pela interface de programação da biblioteca Lua. Além das bibliotecas que acompanham o interpretador Lua, como as de matemática, manipulação de *strings* e de acesso ao sistema operacional, o *script* tem à sua disposição as seguintes bibliotecas específicas:

- *Material*: usada para especificar os aspectos visuais e físicos das primitivas geométricas.
- *Space*: auxilia a orientação e posicionamento das primitivas.
- *Primitive*: tem as funções para criar e os métodos para manipular as primitivas.
- *Body*: disponibiliza a função para conversão de uma primitiva em uma casca de colisão ou em um corpo rígido, além de métodos para adição de forças e outras manipulações físicas.
- *Joint*: todas as funções e métodos para criar e manipular as conexões entre corpos rígidos estão agrupadas nesta biblioteca.
- *TCP Server*: oferece uma forma de comunicação com outras aplicações, através de *sockets*.
- *Scene*: para a manipulação de luzes, câmeras e parâmetros da simulação.

No total são cerca de cem funções e métodos, atualmente, que podem ser usados no escopo global do *script* e interpretados durante a inicialização do simulador, ou nos escopos locais de funções que são chamadas durante a simulação, como a *iteration*, chamada a cada passo de simulação, e a *sceneUpdate*, chamada logo antes das atualizações da descrição da cena. Como exemplo, apresentamos um código mínimo para adicionar um robô que segue os padrões do *FIRA Small League MiroSot* (figura 2.4) de futebol de robôs:

```
function robot()
   local bot={}                    -- cria uma nova tabela Lua
   bot.chassisMat=material.new()   -- cria um novo material
   color(0.8)                      -- cor cinza, 80%
   friction(0.1)                   -- coeficiente de atrito em 0.1
```

```
    translate(0,0,0.035)             -- move o sistema de referencia para cima, 3.5cm
    box(0.07,0.07,0.07)              -- cria um cubo de 7cm de lado
    mass(0.2)                        -- ajusta sua massa em 200g
    bot.chassis=colide()             -- transforma a primitiva em um corpo rigido
    bot.wheelMat=material.new()      -- novo material
    noSlip()                         -- atrito infinito (sem derrapagens)
    color(0.7,0.7,1)                 -- cor azul claro
    translate(0,-0.034,-0.015)       -- move para o ponto de encaixe da primeira roda
    local c1=cursor(0,1,0)           -- cria um cursor orientado para o eixo das rodas
    cylinder(0.02,0.004)             -- cria um cilindro (2cm de raio, 0.4cm de compr.)
    mass(0.01)                       -- ajusta sua massa em 10g
    bot.wheelLeftBody=colide()       -- transforma a primitiva em um corpo rigido
    bot.wheelLeft=hinge(bot.chassis, bot.wheelLeftBody, c1) -- cria a junta de rotacao
    translate(0,0.068,0)             -- move para o ponto de encaixe da segunda roda
    local c2=cursor(0,1,0)           -- cria um cursor orientado para o eixo das rodas
    cylinder(0.02,0.004)             -- cria a segunda roda
    mass(0.01)                       -- mesma massa, 10g
    bot.wheelRightBody=colide()      -- transforma a primitiva em um corpo rigido
    bot.wheelRight=hinge(bot.chassis, bot.wheelRightBody, c2) -- segunda junta
    return bot                       -- retorna a tabela com os elementos criados
end
```

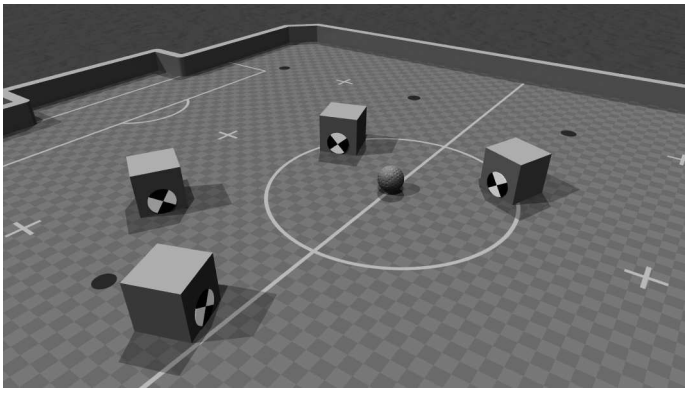

Figura 2.4: Simulação de futebol de robôs

2.2 Algoritmos Genéticos

Algoritmos de busca e optimização, baseados na evolução biológica, são chamados *computações evolutivas*. As principais características da optimização por computação evolutiva são:

- baseadas em múltiplos pontos de busca, ou candidatos à solução (procura baseada em população)
- conduzidas usando operadores inspirados pela evolução biológica, como *mutação* e *crossover*

- baseadas em busca probabilística e operações probabilísticas
- conduzidas usando pouca informação a respeito do espaço de busca

Paradigmas típicos que se enquadram em computação evolutiva incluem algoritmos genéticos, estratégias evolutivas, programação evolutiva e programação genética.

Dentre a terminologia tipicamente usada na literatura de algoritmos genéticos, encontram-se as seguintes expressões:

- *cromossomo*: vetor que representa uma solução específica para uma determinada tarefa
- *gene*: uma característica fenotípica descrita em um cromossomo
- *seleção*: escolha dos cromossomos para a próxima geração
- *indivíduo*: um cromossomo
- *população*: total de indivíduos
- *função de adaptação*: avalia o quanto uma solução é apropriada para uma tarefa
- *fenótipo*: expressão de um gene em um valor do contexto da tarefa
- *genótipo*: representação do gene, como uma seqüência de bits ou de ácidos nucléicos

Algoritmos genéticos representam suas soluções na forma de cromossomos, e prcuram a melhor solução usando os operadores de seleção, mutação e *crossover*.

As principais vantagens desse método são:

- rápida convergência à solução ótima global
- capacidade superior de procura em superfícies de busca complexas
- capacidade de procura em superfícies de busca em que não há informação de gradiente

Em compensação, ao aproximar-se da solução ótima global, esse método se torna muito lento, exatamente o contrário dos métodos baseados em gradiente. Há diversas propostas de algoritmos que juntam os dois métodos, mas, obviamente, eles só funcionam se existe um gradiente disponível.

2.3 A Biomecânica do Movimento

Para agir sobre o ambiente, animais dispõem de atuadores especializados, os músculos. Há três tipos de músculos existerpo humano: liso, cardíaco e esquelético (ou estriado). No entanto, apenas sobre os esqueléticos exercemos controle voluntário.

Os músculos esqueléticos são subdivididos em várias tiras paralelas, estas formadas por várias células multi-nucleadas em forma de fios, chamadas *fibras musculares*. Um mamífero tem uma fibra muscular típica de diâmetro entre 50 e 100 μm (aproximadamente o diâmetro de um fio de cabelo humano), e comprimento entre 2 e 6 cm. Portanto um músculo típico é formado por centenas de milhares, e até mesmo milhões de fibras musculares, dispostas paralelamente e, no caso de músculos mais longos, em série.

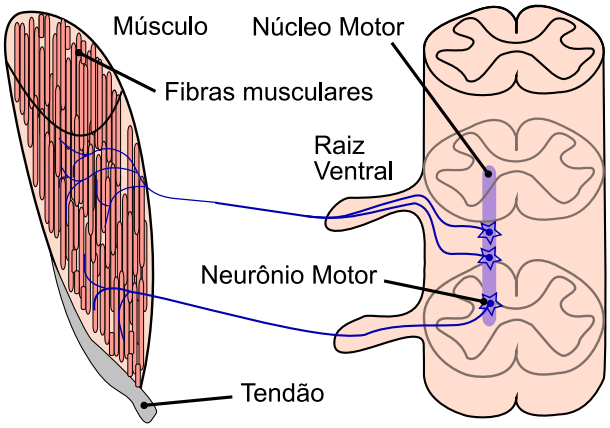

Figura 2.5: Controle muscular

O controle de um músculo típico é realizado por cerca de cem neurônios motores, cujos corpos celulares estão localizados em aglomerados chamados *núcleos motores* (figura 2.5), que estão distribuídos pela espinha e tronco cerebral. O axônio de cada neurônio motor sai da espinha por uma raiz ventral, e ramifica-se ao chegar no músculo, inervando entre 100 e 1000 fibras musculares, espalhadas por grande parte do músculo. Com exceção do período de desenvolvimento, cada fibra muscular é normalmente inervada por apenas um único neurônio motor. O conjunto de fibras controladas por um único neurônio motor é chamado de *unidade muscular*, e junto com seu neurônio é chamado de *unidade motora*.

Quando um neurônio motor dispara, o neurotransmissor acetilcolina é liberado nas sinapses neuromusculares, fazendo com que todas as fibras se contraiam. Rapidamente a acetilcolina é metabolizada nos botões sinápticos, relaxando as fibras e as deixando prontas para um novo disparo do neurônio motor.

A força de uma contração muscular depende do número de fibras selecionadas para a ação e da freqüência de disparos dos neurônios motores. Nos músculos humanos a freqüência fica entre 5 e 25 Hz, o que parece pouco para uma contração contínua, mas a assincronicidade dos disparos dos diversos neurônios motores faz com que as forças das diversas fibras se mesclem, proporcionando uma contração suave.

Alguns músculos têm bastante liberdade de movimentos, como o conjunto de músculos que compõem a língua. No entanto, a maior parte dos músculos está associada a ossos, que restringem e direcionam a atuação da força. Os dispositivos formados por essa combinação têm usos específicos, conforme sua morfologia. Na figura 2.6 podemos ver um modelo do mecanismo básico da movimentação de braços e dedos.

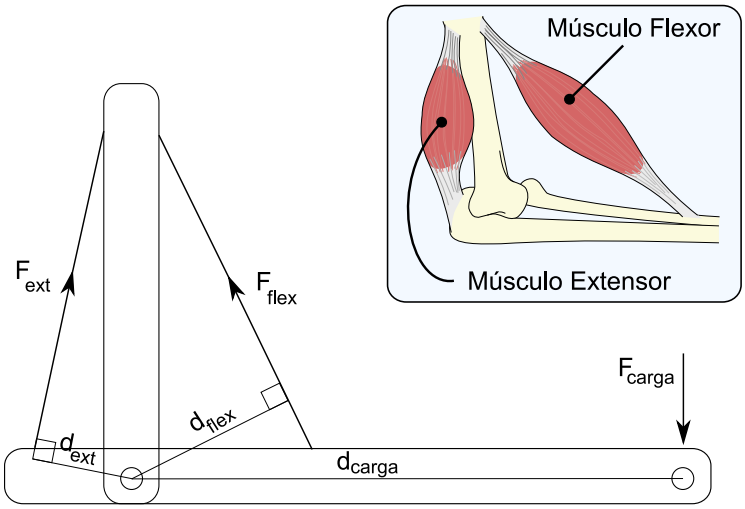

Figura 2.6: Modelo de dispositivo atuador baseado em alavancas

Cada músculo produz um torque sobre a junta, calculado como o produto da força contrátil e da distância entre o centro de rotação da junta até o ponto mais próximo sobre a reta da direção da força, distância essa chamada de braço de momento (*moment arm*), conforme as equações

$$Torque_{ext} = F_{ext} \cdot d_{ext}$$
$$Torque_{flex} = F_{flex} \cdot d_{flex}.$$

O torque total é calculado como a soma dos torques gerados por todos os músculos que atuam sobre a junta. No caso da figura 2.6, os músculos são antagonistas, e portanto produzem torque em sentidos opostos, resultando na seguinte equação para o torque total:

$$Torque_{total} = Torque_{flex} - Torque_{ext}.$$

Para que o membro fique parado, sustentando uma carga, o torque total gerado pelos músculos tem que anular o torque gerado pela carga, conforme a equação

$$Torque_{total} - F_{carga} \cdot d_{carga} = 0.$$

2.4 O Robô Quadrúpede

Não nos preocupamos em modelar com precisão nenhuma espécie em particular, o objetivo era criar um robô com um número grande de graus de liberdade que apenas lembrasse um felino. Para isso, tomamos como referência diversas fotos encontradas livremente pela internet, e ao projetar o modelo, tentamos respeitar as proporções e ângulos entre as partes do corpo conforme o que foi observado.

O modelo de robô usado neste trabalho (figura 2.7) tem 16 graus de liberdade na forma de 4 juntas do tipo dobradiça em cada um dos membros, e cada uma das juntas está associada a um servo-motor. Toda a estrutura é formada usando apenas a composição de geometrias primitivas, especificamente esferas, cilindros e paralelepípedos. Sua massa total é de aproximadamente 8 Kg, e sua altura (sobre as quatro patas) é de pouco menos de um metro. Uma esfera foi usada para representar a massa relativa ao peito, deslocando o centro de gravidade um pouco para baixo e para frente.

O controle do robô é efetuado informando-se os ângulos desejados em cada junta aos seus respectivos servo-motores, que aplicam uma força proporcional à diferença com o ângulo atual (limitado em 35N), reduzindo esta diferença gradualmente até chegar ao ângulo desejado.

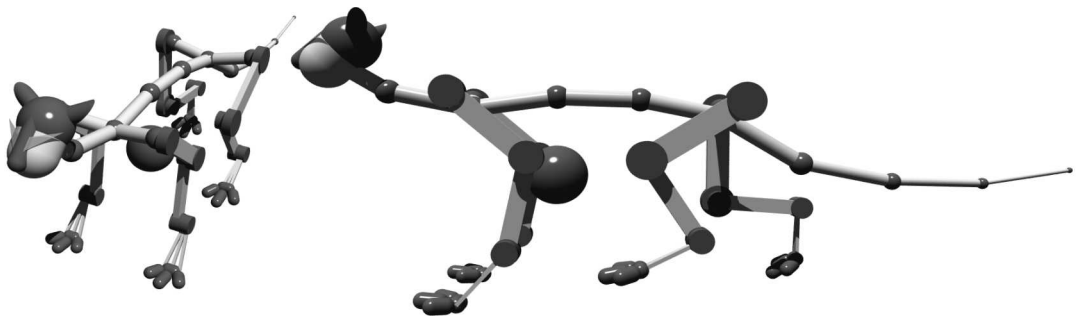

Figura 2.7: Visões diagonal e lateral do robô simulado

Esta abstração de controle permite uma aplicação praticamente sem alterações em um robô real, pois servo-motores reais funcionam de forma similar aos simulados, e apesar de estar longe da forma como a movimentação é gerada em seres vivos, a pura geração dos padrões de contrações musculares (capítulo 4) é análoga à geração de padrões de ângulos, e portanto este modelo foi suficiente para o desenvolvimento deste trabalho.

No entanto o uso de ângulos torna difícil a integração do gerador de padrões com outros sistemas, como reflexos e modulações supra-espinhais. Uma abstração baseada em músculos permite uma solução muito mais elegante, e portanto trabalhos futuros serão desenvolvidos usando um novo modelo de robô. Seu desenvolvimento está quase concluído, sua estrutura está pronta, resta apenas posicionar corretamente os músculos. A figura 2.8 mostra uma versão preliminar. Uma votação para a escolha de um nome para o novo modelo foi realizada pela internet, e o nome vencedor foi Nyau, com 25.3% de 75 votos.

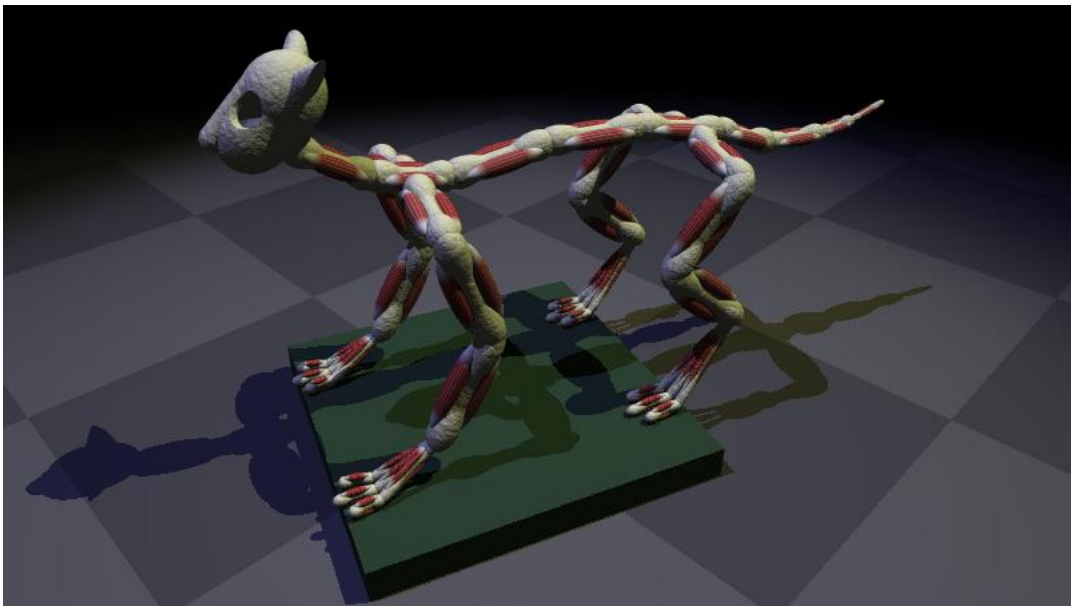

Figura 2.8: O novo modelo de robô: Nyau

3 O MODELO DE NEURÔNIO E DE REDE NEURAL

O neurônio é a célula responsável pelo processamento mais básico de sinais, servindo como uma chave que se ativa conforme a configuração dos sinais que recebe. Um cérebro humano tem aproximadamente 10^{11} neurônios, os quais podem ser classificados em pelo menos 10^3 tipos diferentes, e no entanto todos compartilham a mesma estrutura básica. A complexidade do comportamento humano depende menos da especialização de um neurônio que da precisa rede de contatos que formam os diferentes circuitos neurais.

A seção 3.1 resume o princípio de funcionamento de um neurônio biológico. Nas seções 3.2, 3.3 e 3.4 são apresentados modelos de neurônios com características diversas. Uma classificação de sistemas neurais é apresentada na seção 3.5, seguido pela proposta de um novo modelo de rede, na seção 3.6.

3.1 Neurônio Biológico

Um neurônio típico, como o mostrado na figura 3.1, tem quatro regiões morfologicamente determinadas: o corpo celular, os dendritos, o axônio, e os botões sinápticos. Cada uma dessas partes tem uma função distinta na geração e comunicação de sinais entre as células nervosas.

O corpo do neurônio é o responsável por suas funções metabólicas, e contém o núcleo, onde está armazenado o código genético, assim como o retículo endoplasmático, uma extensão do núcleo responsável pela síntese de proteínas. Do corpo do neurônio, geralmente se projetam vários dendritos curtos e um axônio longo e tubular. Os dendritos se ramificam em forma de raízes, e são os principais responsáveis pela recepção de sinais de outros neurônios. O axônio se projeta para longe do corpo celular, a distâncias que variam de meros 0,1mm até 3m, e é o principal meio de condução dos sinais para outros neurônios.

A tensão elétrica no citoplasma de neurônios em estado de repouso varia conforme sua função, podendo estar entre -40mV e -90mV, sendo o valor mais freqüente o de -65mV, em relação ao ambiente externo à célula. Para se manter

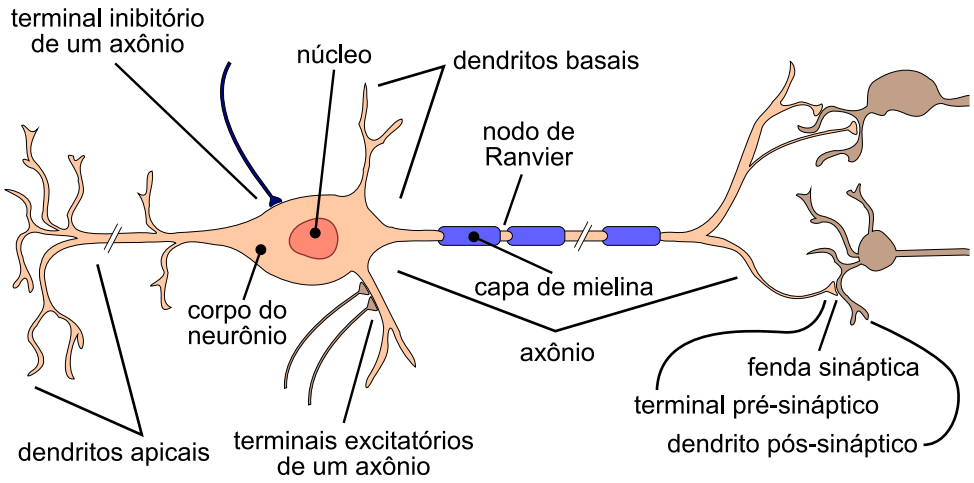

Figura 3.1: Estrutura de um neurônio

nesse estado, neurônios dispõem de mecanismos na membrana celular, responsáveis por trocar íons com o meio externo. Quando há uma redução rápida no potencial (despolarização), a célula fica mais propensa a gerar um sinal.

Os sinais, chamados potenciais de ação (*action potentials*), são impulsos elétricos rápidos, transitórios e booleanos (isto é, ou há ou não há fluxo de íons), com uma amplitude de 100mV e duração de aproximadamente 1ms. Um princípio chave do funcionamento do cérebro é o fato do processamento das informações não ser dependente da forma ou amplitude dos sinais, e sim dos caminhos pelos quais eles percorrem. Assim, sinais tão diversos, como os provenientes da audição ou da visão, são processados pelos neurônios da mesma forma.

Potenciais de ação são iniciados em uma região de disparo especializada, localizada no primeiro segmento do axônio, e de lá eles são conduzidos pelo axônio a velocidades entre 1 e 100m/s, sem falhas ou distorções, graças à amplificação que ocorre em cada nodo de Ranvier. Próximo do seu final, o axônio se divide em ramos finos para formar pontos de comunicação com outros neurônios, locais estes chamados *sinapses*. Há dois tipos de sinapses: elétricas e químicas.

A distância entre o terminal do axônio e o dendrito, chamado fenda sináptica (*synaptic cleft*), é de aproximadamente 3,5nm na sinapse elétrica, ocorrendo inclusive uma continuidade citoplasmática, o que permite que a corrente de íons do potencial de ação pré-sináptico flua com baixa resistência até o corpo do neurônio pós-sináptico. Essa continuidade também faz com que possa haver comunicação no sentido inverso, do dendrito para o axônio.

Na sinapse química, a comunicação ocorre pela liberação de moléculas cha-

madas *neurotransmissores*, que são produzidos pelo neurônio e armazenados em vesículas nos botões sinápticos. Quem determina se o estímulo químico será excitatório ou inibitório são os *neuro-receptores*, que são componentes da membrana do neurônio pós-sináptico, influenciando direta ou indiretamente o fluxo de íons. Esse mecanismo possibilita a amplificação dos sinais, visto que, com um potencial de ação desencadeando a abertura das vesículas, milhares de moléculas neurotransmissoras podem ser despejadas na fenda sináptica, o que possibilita que mesmo um neurônio pequeno ative um neurônio muito maior. A fenda sináptica, neste caso, tem entre 20 e 40nm, o que junto com o tempo da reação química introduz um atraso na comunicação de no mínimo 0,3ms, em geral entre 1 a 5ms ou mais. Essa dimensão de fenda também faz com que praticamente não haja influência elétrica do potencial de ação sobre o neurônio pós-sináptico, e inibe a comunicação no sentido inverso. Os efeitos de uma sinapse química podem permanecer durante poucos segundos a até mesmo vários minutos.

Sinapses podem ter seu efeito ampliado conforme a freqüência de ativações do neurônio pré-sináptico (FIELDS, 2005). Diferentes processos químicos são desencadeados dependendo do período em que ocorrem as despolarizações, sinalizando ao núcleo celular a síntese de proteínas com funcionalidades diversas, entre elas o fortalecimento das sinapses pelo efeito chamado de potenciação de longo prazo (LTP). A LTP precoce é uma modificação temporária da sinapse que pode permanecer por algumas horas, formando uma memória de curto prazo. Memórias permanentes são realizadas pela chamada LTP tardia.

3.2 Unidade de Limiar Linear de McCulloch e Pitts

Este foi o primeiro modelo computacional de neurônio (e também primeiro modelo de autômato de estados finitos) proposto. Em 1943, o psiquiatra e neurofisiologista Warren McCulloch combinou esforços com o matemático Walter Pitts, e juntos criaram o trabalho seminal "*A Logical Calculus of the Ideas Immanent in Nervous Activity*"(MCCULLOCH; PITTS, 1943), que consistia na proposta de uma rede de unidades simples, as quais se supunha que capturavam os funcionamentos essenciais de um neurônio biológico.

A unidade de limiar linear calcula o valor de sua entrada como a soma ponderada das saídas de todas as outras unidades que convergem a ela. Se esse valor for maior que o limiar θ, a unidade se ativa (sua saída y recebe o valor 1), se não permanece inativa (y com valor 0). Na figura 3.2, as variáveis x representam as saídas das outras unidades, w os pesos associados a cada uma delas, e y a saída, calculada pelo sistema:

$$y = \begin{cases} 1 & \text{se } \theta < \sum_{i=1}^{n} x_i w_i \ , \\ 0 & \text{caso contrário.} \end{cases}$$

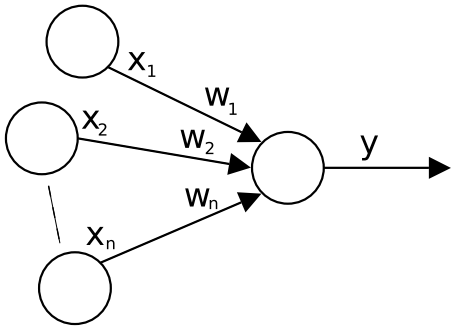

Figura 3.2: Modelo de neurônio de McCulloch e Pitts

O modelo de McCulloch e Pitts ganhou popularidade principalmente pelo fato da unidade de limiar linear conseguir representar operadores booleanos, bastando para isso ajustar os pesos das conexões e o valor do limiar, como exemplificado pela figura 3.3.

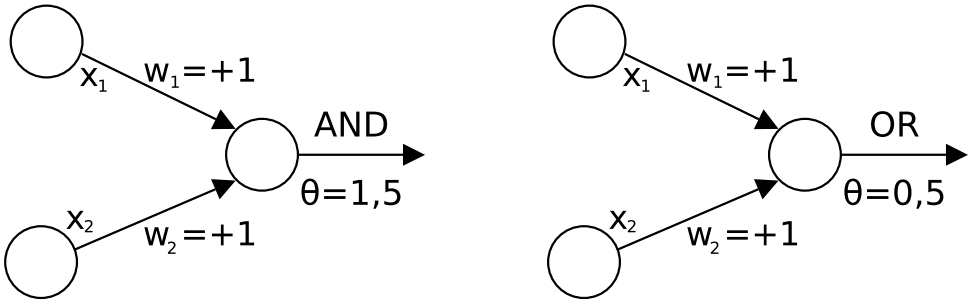

Figura 3.3: Operadores booleanos *and* e *or*

Ao combinar diversas unidades em uma rede lógica, é possível realizar as mais diversas e complexas computações. No entanto, o modelo não dispõe de uma forma sistemática para determinar os pesos e limiares, sendo necessário que esses valores sejam projetados manualmente.

3.3 *Perceptron*

O que hoje em dia se conhece como *perceptron* é apenas um dos modelos que Frank Rosenblatt apresentou em 1959, em seu livro "*Principles of Neurodynamics*".

São essencialmente unidades de limiar linear equipadas com um procedimento para a determinação dos pesos das conexões através de exemplos. O limiar θ é transformado no peso de conexão w_0 para facilitar o algoritmo de aprendizagem pelo seguinte artifício matemático:

$$\begin{aligned} \theta &< \sum_{i=1}^{n} x_i w_i \\ 0 &< \sum_{i=1}^{n} x_i w_i - \theta \\ 0 &< \sum_{i=1}^{n} x_i w_i + x_0 w_0 \qquad\qquad \text{com } x_0 = 1 \end{aligned}$$

Com $w_0 = -\theta$ chegamos à equação da ativação do perceptron (3.1), representado no diagrama da figura 3.4. O valor de w_0 é usualmente chamado de viés (*bias*) na literatura, pois é o valor inicial da entrada sem as outras conexões.

$$y = \begin{cases} 1 & \text{se } 0 < \sum_{i=0}^{n} x_i w_i \text{ com } x_0 = 1, \\ 0 & \text{caso contrário.} \end{cases} \tag{3.1}$$

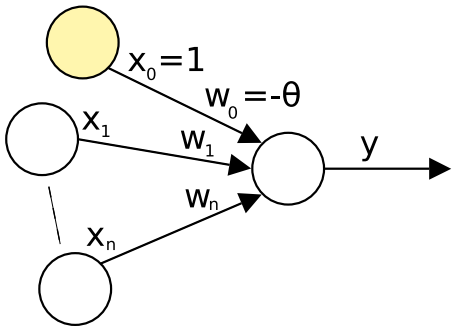

Figura 3.4: *Perceptron* de Rosenblatt

Rosenblatt provou que se há solução para o problema de classificação (ou seja, é um problema linearmente separável), o algoritmo 1 encontrará um conjunto de pesos que o resolva em um tempo finito.

Para os outros casos, é necessário o uso de mais *perceptrons*, formando uma rede, normalmente referenciada como *multilayered perceptrons*. Uma conseqüência de usar *perceptrons* em camadas é que o algoritmo de aprendizagem deve ser diferente. O algoritmo mais conhecido para o treinamento de *multilayered perceptrons* é o

Algorithm 1 Aprendizagem do *perceptron*

1. escolha uma taxa de aprendizagem α entre 0 e 1.
2. defina um conjunto de exemplos, compostos das entradas $x_1, ..., x_n$ e da saída y_d desejada.
3. inicialize os pesos $w_0, ..., w_n$ com valores aleatórios pequenos.
4. para cada iteração t, selecione um padrão do conjunto de exemplos.
5. calcule y conforme a equação 3.1.
6. se $y = y_d$, vá para o passo 7, senão atualize os pesos conforme:

$$w_i(t+1) = w_i(t) + \alpha(y_d - y) \cdot x_i \tag{3.2}$$

7. se o erro estiver aceitável, parar, caso contrário, voltar ao passo 4.

de retro-propagação de erro (*error backpropagation*), no entanto este método usa *perceptrons* com uma função de transferência (a função que calcula y a partir de $\vec{x}$ e $\vec{w}$) diferente da equação 3.1. Neste caso a função deve ser não-linear, mas contínua.

3.4 *Spiking Neurons*

Neurônios da categoria do *perceptron* não têm estados, o resultado da computação da rede é o cálculo do fluxo instantâneo de valores pelas sinapses até os neurônios de saída. Em contraste, *spiking neural networks* (conhecidas também como *pulsed neural networks*) são redes em que cada neurônio acumula pulsos de ativação, e se esse potencial acumulado (que decai com o tempo) exceder um determinado limiar, ele próprio gera um pulso para os neurônios com os quais mantém sinapses (BOHET, 2003). Cada sinapse tem um peso w e um atraso d. Sendo x o potencial acumulado, ele decai segundo a equação:

$$\frac{dx}{dt} = -kx$$

Como o único momento em que o potencial acumulado poderia exceder o limiar α é no momento da chegada de um novo pulso, o decaimento só precisa ser calculado nessas ocasiões, conforme: $x_t = x_{lt}e^{k(t-lt)} + w_i$, sendo lt o tempo de chegada do último pulso e w_i o peso da sinapse em consideração. Finalmente, se $x > \alpha$ o neurônio dispara e ao seu potencial é atribuido uma energia de repouso: $x = x_r$.

Diferentemente dos perceptrons, que usam intensidades como sinais de entrada, redes neurais pulsadas usam codificação por tempo. A entrada é um vetor com os tempos dos pulsos (e.g. $\{t_i, t_j, ..., t_k\}$), e o tempo preciso do pulso de saída pode ser interpretado como a computação sobre a entrada. Foi mostrado teoricamente que a performance desse método é ao menos tão boa quanto a de redes que usam neurônios convencionais (MASS, 1997).

3.5 Tipos de Modelos

O nível de detalhamento de um modelo de sistema conexionista deve ser dependente de sua aplicação. (KOHONEN, 1989) sugere esta divisão:

1. Modelo em nível de neurônio, não acoplado: onde os mais cuidadosos estudos se concentram na dinâmica e nas propriedades adaptativas de cada neurônio, de forma a descrever o neurônio como unidade que responde seletivamente a padrões de entrada e tem algumas funções elementares de memória.

2. Modelo em nível de rede neural: onde neurônios idênticos são interconectados para exibir um dos seguintes comportamentos: função de filtragem (e.g. operador de projeção ou estimador ótimo); sistema retro-alimentado não-linear, que converge para um dos seus auto-vetores (*eigenstates*), correspondendo a uma solução para um problema de optimização; definição dos auto-valores (*eigendata*) da entrada. Devido a esses modelos serem computacionalmente mais pesados, eles precisam usar modelos simplificados de neurônios como unidades básicas de processamento, e concentrar na descrição do efeito coletivo básico.

3. Modelo em nível de sistema nervoso (organizacional): em geral duas ou mais redes do tipo 2 com diferentes propriedades são combinadas para demonstrar comportamentos mais complexos ou abstratos de percepção sensora, como classificação automática, formação de conceitos, funções motoras, ou controle da estabilidade global. Este último é um importante aspecto, pois o sistema precisa ser mantido em um estado global estável o tempo inteiro, e exibir, conforme o caso, as funções identificáveis como atenção, motivação, entre outros efeitos psicológicos. O objetivo primário é o estudo de fenômenos sistêmicos mais qualificados.

4. Modelo em nível de operações mentais: os mais abstratos, freqüentemente descrevendo operações, procedimentos, algoritmos e estratégias no chamado "nível de processamento de informações humano". Estes modelos são uma forma de descrever os processos básicos da cognição, pensamento, solução de

problema, etc., e são relacionados proximamente com a pesquisa em Inteligência Artificial.

Devido aos erros introduzidos pela simplificação, não é recomendada a integração de modelos distantes mais que dois níveis.

3.6 Modelo de Rede Proposto

O objetivo deste modelo não é ser semelhante a um neurônio biológico (nível individual), e sim ser uma ferramenta prática para a simulação do comportamento de populações de neurônios (nível sistêmico). Ele deve ser capaz de incorporar matematicamente os modelos mais comuns de neurônios de forma simples.

Mas consideramos indispensáveis duas características de neurônios biológicos: a capacidade de inibir populações inteiras de neurônios e a presença de estado (na forma da concentração iônica).

O modelo de neurônio é a 5-upla $< CL, AP, f(x), AS, PS >$, onde:

- CL é um valor racional que representa o nível de carga (*charge level*) atual do neurônio.
- AP é o potencial de ação (*action potential*), também um valor racional, que representa o nível de ativação do neurônio (sua saída). É uma abstração da taxa de disparo de um neurônio real.
- $f(x)$ é a função que transfere CL para AP, e pode ser tanto a linear

$$f_{lin}(x) = x \cdot gain + offset$$

como a sigmóide

$$f_{tanh}(x) = \tanh\left(\frac{x + offsetx}{linearSize}\right) \cdot range + offsety.$$

- AS é o conjunto de sinapses aditivas (*additive sinapses*).
- PS é o conjunto de sinapses de potenciação (*potentiation sinapses*).

Na figura 3.5 vemos a representação de uma rede com um neurônio s com suas três sinapses projetadas ao neurônio d, cada uma com características diferentes.

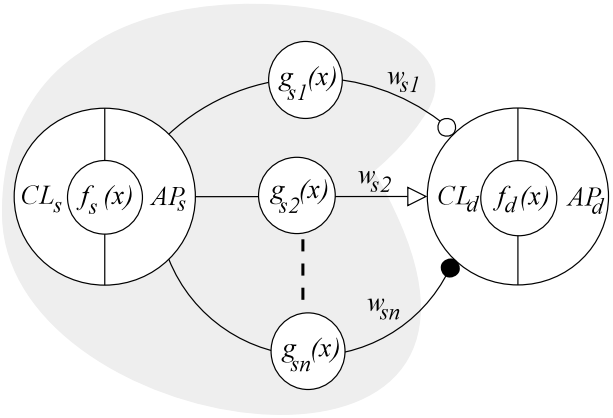

Figura 3.5: Representação do neurônio s com suas três sinapses para o neurônio d

Cada sinapse consiste de uma função de transferência linear $g(x) = x \cdot gain + offset$ (chamada inter-neurônio), um peso w e o endereço do neurônio destino d, formando a 3-upla $< g(x), w, d >$.

As sinapses *aditivas excitatórias*, representadas na figura 3.5 pela curva com um círculo aberto no final, são aquelas em AS com $w > 0$. As sinapses *aditivas inibitórias*, representadas como uma curva com um círculo fechado no final, são aquelas em AS com $w < 0$. As sinapses desenhadas com uma seta aberta no final são as de *potenciação excitatória*, que estão contidas em PS e têm $|w| > 1$. As sinapses de *potenciação inibitória*, que não foram mostradas na figura 3.5, são representadas por uma curva com uma seta fechada no final, têm $|w| < 1$ e estão em PS.

A rede neural é o conjunto de todos os neurônios, representada por N. O fluxo dos sinais pela rede é dividido em cinco etapas, executadas sincronamente:

1. *adição*: ao nível de carga de cada neurônio é acrescentado o somatório do cálculo de todas as sinapses aditivas que chegam a ele, como mostra a equação

$$\forall n \in N, \forall as \in AS_n : CL_{d_{as}} := CL_{d_{as}} + g_{as}(AP_n) \cdot w_{as}$$

2. *potenciação*: o nível de carga é modulado pelas sinapses potenciadoras, conforme a equação

$$\forall n \in N, \forall ps \in PS_n :$$
$$CL_{d_{ps}} := g_{ps}(AP_n) \cdot w_{ps} \cdot CL_{d_{ps}} + (1 - |g_{ps}(AP_n)|) \cdot CL_{d_{ps}}$$

Como se pode ver, se $g_{ps}(AP_n) = 1$, $CL_{d_{ps}}$ é completamente modulado (multiplicado por w_{ps}). Se $g_{ps}(AP_n) = 0$, o valor de $CL_{d_{ps}}$ permanece inalterado.

3. *ativação*: o potencial de ação de cada neurônio é calculado.

$$\forall n \in N : AP_n := f_n(CL_n)$$

4. *aprendizado*: esse passo é reservado para a atualização dos pesos sinápticos, mas não foi necessário neste trabalho.

5. *repouso*: ao nível de carga de cada neurônio é atribuído o valor zero.

$$\forall n \in N : CL_n := 0$$

Um ciclo de execução típico consiste em atribuir ao nível de carga de alguns neurônios os valores dados por sensores, iterar a rede neural conforme descrito acima, e então usar os valores de potenciais de ação de alguns neurônios como entrada para atuadores.

Neste trabalho usamos uma freqüência de execução equivalente a 1KHz, para ser coerente com o passo de tempo usado em nossos sistema de simulação física, que foi de 1ms.

Uma observação interessante a respeito deste modelo de rede é a possibilidade de reproduzir a funcionalidade de aproximador universal de um *perceptron* de múltiplas camadas, bastando para isso copiar sua topologia e adaptar os parâmetros de seus neurônios, além de se levar em conta o atraso na propagação dos sinais na rede.

4 ARQUITETURA NEURAL PARA GERAÇÃO DE PADRÕES

Um Gerador Central de Padrões (CPG) é uma rede neural capaz de gerar um padrão de atividade motora mesmo na ausência da entrada de sinais fásicos provenientes de receptores periféricos. CPGs foram identificados e analisados em mais de 50 sistemas motores rítmicos (KANDEL; SCHWARTZ; JESSELL, 2000), incluindo os responsáveis pelo controle de comportamentos tão diversos quanto a locomoção (nadar, rastejar, caminhar, voar, etc.), respiração e alimentação (mastigar, engolir, sugar o seio, etc.). Os sinais gerados por CPGs são normalmente modulados em função do estado de sensores periféricos e de outras regiões do sistema nervoso central.

Uma das primeiras hipóteses sobre o mecanismo de geração de padrões foi feita por Charles Sherrington em 1906, em uma publicação onde ele propunha que todos os movimentos eram baseados em reflexos simples, e movimentos complexos não passavam de encadeamentos de reflexos. Esta visão permaneceu aceita até 1961, quando Donald Wilson conseguiu isolar o circuito neural que gerava os padrões de ativação musculares do vôo de um gafanhoto (WILSON, 1961). Atualmente é aceito que CPGs e reflexos desempenham funções paralelas objetivando uma movimentação adaptativa.

Na figura 4.1 pode-se ver o esquema do circuito neural de um segmento da medula espinhal da lampréia, responsável por seu nado. A atividade na rede é iniciada por axônios provenientes da formação reticular, no tronco cerebral. Em cada lado da rede, neurônios excitatórios (E) ativam os neurônios motores (NM) e duas classes de inter-neurônios inibitórios (I e L). Os axônios dos inter-neurônios do tipo I cruzam o segmento inibindo todos os neurônios da metade contra-lateral, garantindo que apenas os músculos de um lado estejam ativos em um determinado momento, enquanto o outro lado permanece em repouso. Os inter-neurônios do tipo L inibem os do tipo I no mesmo lado e, devido a diferenças em suas características temporais, geram um período de atividade seguido por um período de repouso. Conexões seriais entre os diversos segmentos propagam o sinal com um

pequeno deslocamento na fase, e sempre mantendo a diferença de fase de 180° entre os dois lados. O movimento de nado gerado também assemelha-se ao do rastejar de algumas cobras.

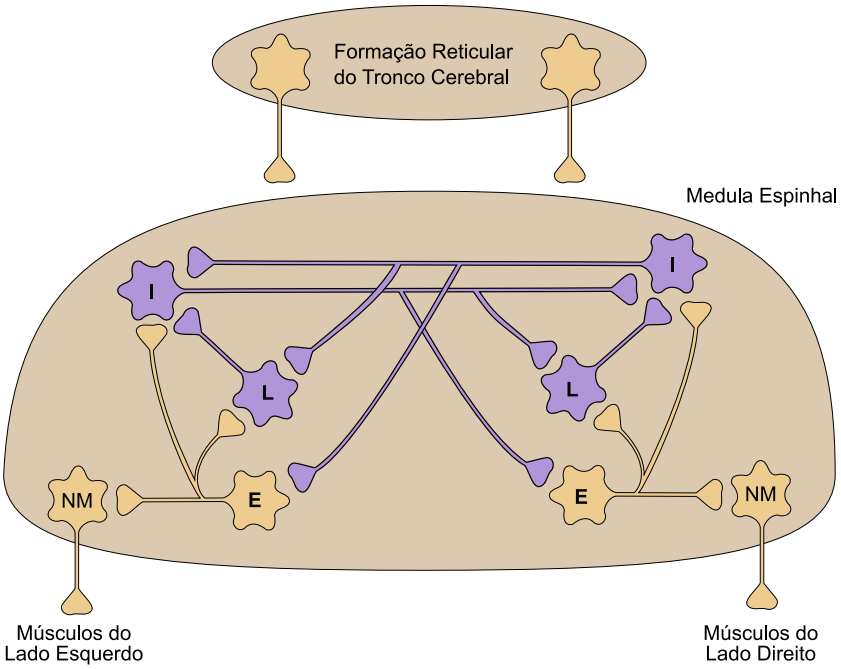

Figura 4.1: Segmento de CPG da lampréia

Isolar CPGs em mamíferos é bem mais complexo, e ainda não pôde ser feito. No entanto, estudos clínicos indicam que mesmo em humanos o princípio é o mesmo.

Na seção seguinte alguns modelos neurais de geradores de padrões são analisado.

4.1 Modelos Neurais de CPGs

(AMES, 2003) propõe o uso de Redes Neurais Recorrentes de Tempo Contínuo (CTRNN) com a determinação de seus parâmetros usando Algoritmos Genéticos. CTRNNs são feitas com modelos de neurônios regidos por equações diferenciais contínuas e não-lineares, e foi provado que são aproximadores universais de qualquer sistema dinâmico contínuo (FUNAHASHIM; NAKAMURA, 1993). A saída dos neurônios de uma rede completamente conectada em função do tempo é defi-

nida pela equação

$$\frac{dy_i}{dt} = \frac{1}{\tau_i}\left(-y_i + \sum_{j=1}^{N} w_{ji}\sigma(y_j + \theta_j) + I_i\right)$$

onde N é o número de neurônios, i varia de 1 a N, y_i é a saída do neurônio i, w_{ji} é o peso da conexão entre o neurônio j e o neurônio i, τ_i é a constante de tempo do neurônio i, θ_j é o viés (*bias*) do neurônio j, I_i é uma entrada externa ao neurônio i, e $\sigma(x)$ é a função logística $1/(1 + e^{-x})$.

Uma desvantagem deste método é a ausência de controle da velocidade de reprodução do padrão. (SRINIVASAN; GANDER; WOOD, 1992) criaram um modelo capaz de gerar padrões em freqüências intermediárias às quais o circuito foi treinado. A rede apresentada é uma derivação das redes seqüenciais de (JORDAN, 1986), sem conexões reflexivas na entrada e a saída da rede é conectada à sua entrada com atraso de uma unidade temporal (figura 4.2).

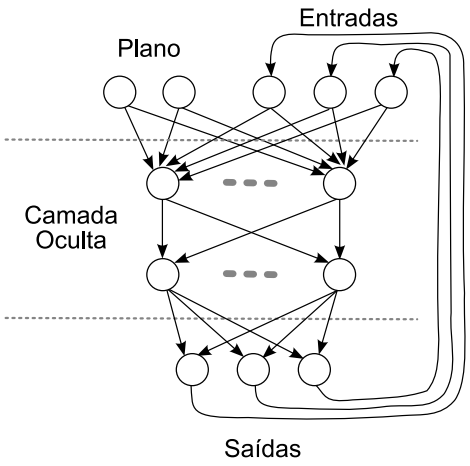

Figura 4.2: Diagrama da rede seqüencial de Jordan adaptada

Os neurônios do plano representam a modulação supra-espinhal, tanto em termos da velocidade da reprodução do padrão, quanto da escolha do padrão em si. A rede é treinada pelo algoritmo de retropropagação de erro (*error backpropagation*) convencional, desconsiderando-se as conexões da saída para a entrada. No conjunto de treinamento podem existir padrões em diferentes velocidades, bastando associar um valor correspondente a um neurônio do plano. Por exemplo, seja n o neurônio do plano, treina-se um padrão de velocidade τ com $n = 0$, e

treina-se o mesmo padrão com velocidade 2τ com $n = 1$. Durante a reprodução dos padrões, basta indicar um valor intermediário de n para alterar a velocidade correspondentemente.

4.2 Arquitetura Proposta

O chaveamento de circuitos causado pela inibição recíproca, vista no CPG da lampréia, lembra muito o funcionamento de uma Máquina de Estados Finitos. Para investigar as propriedades de um gerador com essa característica, criamos uma nova arquitetura neural.

O método que desenvolvemos para a geração de padrões baseia-se em um conjunto de dois circuitos simples: um gerador de funções (subseção 4.2.1) e um mapeador universal (subseção 4.2.2).

4.2.1 Geração de Funções

O circuito neural mostrado na figura 4.3, composto por seis neurônios que seguem o modelo apresentado na seção 3.6, é capaz de gerar sinais em forma de dente-de-serra e triangulares, assim como detectar e sinalizar quando metade de um ciclo foi atingido e desligar-se ao final do ciclo.

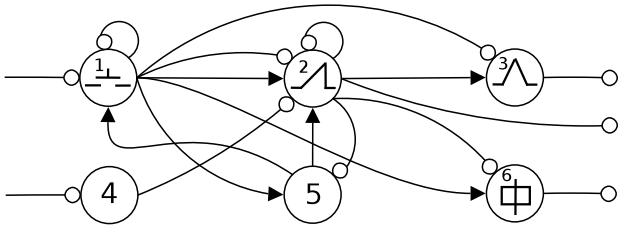

Figura 4.3: Circuito gerador das funções

Cada neurônio, assim como suas sinapses e parâmetros das funções de transferência, são descritos em detalhes abaixo:

1. *lock*: este neurônio sigmóide tem seus parâmetros ajustados de forma a agir como uma chave. Os parâmetros $range = 0.5$ e $offsety = 0.5$ escalam e deslocam a função para que sua saída fique no intervalo $(0, 1)$. Ao se realimentar a função com sua saída, tendo $linearSize = 0.001$ e $offsetx = -0.5$, valores de entrada menores que 0.5 convergem para uma saída próxima de zero, e valores de entrada maiores que 0.5 convergem para um saída próxima de um, em poucas iterações.

Após receber um pulso de ativação externo, a sinapse excitatória reflexiva o mantém em um estado ativo ($AP_{lock} \approx 1$). Este neurônio tem uma sinapse excitatória para o neurônio *sawTooth* (2), e seu peso $w = \gamma$ define o tempo que será necessário para $AP_{sawTooth}$ atingir o valor um. Também tem uma sinapse com peso $w = 1$ para o neurônio *triangle* (3), que forma o sinal base para uma modulação posterior. Sinapses de pontenciação inibitória são projetadas aos neurônios *sawTooth* (2), *checkHalf* (6) e *checkEnd* (5), todas com $g(x) = 1 - x$ e $w = 0$, significando que, se *lock* não estiver ativo, o nível de carga dos referidos neurônios estará sempre nulo.

A atividade no neurônio é encerrada ($AP_{lock} \approx 0$) ao receber um pulso inibitório do neurônio 5, *checkEnd*.

2. *sawTooth*: neurônio linear com parâmetros padrão ($offset = 0$ e $gain = 1$). Seu potencial de ação inicia vazio (valor zero) mas, com sua sinapse excitatória reflexiva, qualquer nível de carga diferente de zero é acumulado. Como recebe a ativação constante de *lock*, sua saída forma o sinal dente-de-serra com o passar do tempo. As sinapses para os neurônios *checkHalf* (6) e *checkEnd* (5) fornecem a base para a análise da fase. Uma sinapse de potenciação inibitória, com $g(x) = 2x-1$ e $w = 0$, modula o nível de carga do neurônio *triangle* (3), formando o sinal triângulo com o passar do tempo. Se $AP_{sawTooth} = 0$, $g(0) = -1$ faz $CL_{triangle} = (1-|-1|) \cdot CL_{triangle} = 0$. Quando $AP_{sawTooth} = 0.5$, $g(0.5) = 0$, inibindo a modulação. Ao final $g(1) = 1$, fazendo $CL_{triangle} = 0$ novamente. O potencial de ação de *sawTooth* volta a zero ao final do ciclo, quando *lock* ativa sua sinapse de potenciação inibitória.

3. *triangle*: linear, parâmetros padrão. Apenas reflete seu nível de carga (que já foi modulado em forma de triângulo).

4. *speedControl*: linear, parâmetros padrão. Este neurônio serve como uma estação de escala para um controle de velocidade normalizado externo, projetando sua sinapse excitatória ao neurônio *sawTooth* (2) com peso $w = \gamma$. Se $AP_{speedControl} = 1$ a velocidade é duplicada, se $AP_{speedControl} = -1$ a sinapse com *sawTooth* anula a excitação proveniente de *lock* e interrompe o aumento de $AP_{sawTooth}$. No gráfico à esquerda da figura 4.4 estão as saídas do circuito sem a influência deste neurônio, e no gráfico à direita são apresentadas as mesmas saídas, desta vez com *speedControl* sendo excitado com valores progressivamente decrescentes, de 1 a -1.

5. *checkEnd*: neurônio sigmóide, com parâmetros ajustados para ficar ativo ao receber um sinal de *sawTooth* (2) com valor maior que um ($range = 0.5$, $offsety = 0.5$, $offsetx = -1$, $linearSize = 0.001$). Quando este neurônio é

ativado, suas projeções de potenciação inibitória para os neurônios *lock* (1) e *sawTooth* (2) encerram as atividades no circuito.

6. *checkHalf*: sigmóide, com $range = 0.5$, $offsety = 0.5$, $offsetx = -0.5$ e $linearSize = 0.001$, ativo quando a sua entrada é maior que 0.5. A saída deste neurônio é usada para iniciar a próxima meia-fase do padrão. Ele é desligado pela sinapse de potenciação inibitória proveniente de *lock* (1).

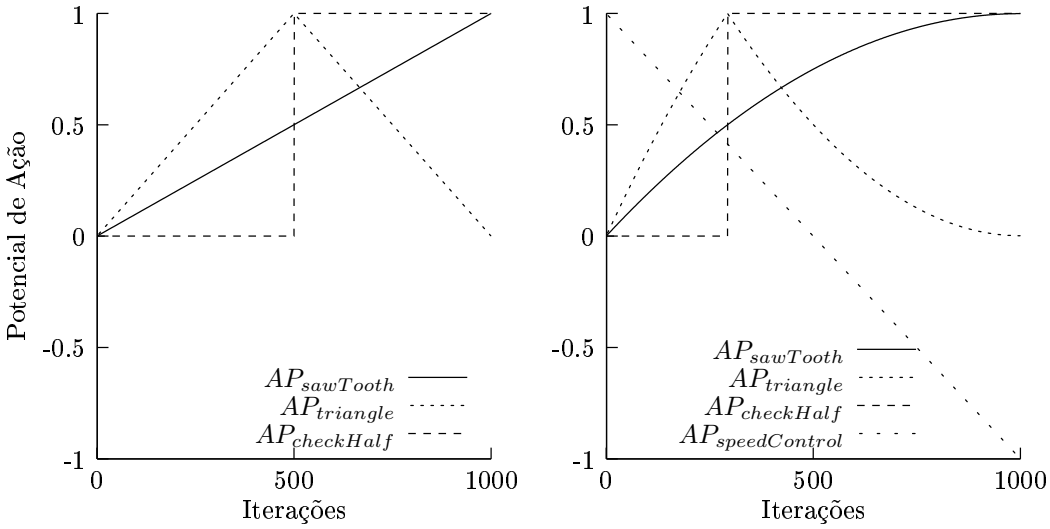

Figura 4.4: Gráficos da saída do circuito com $\gamma = 10^{-3}$, o da direita mostrando a influência do neurônio de controle de velocidade

4.2.2 Geração de Padrões

Para gerar (reconstruir) um padrão de sinal unidimensional *signal'*, fazemos a interpolação de várias funções de mapeamento, cada uma responsável por um trecho do sinal original *signal*. Primeiro precisamos definir a duração desses trechos, representada por λ. Então escolhemos o início de cada trecho ($segBegin_i$, $i \in [1, 2, ..., n]$) de forma que intersecte o anterior a partir da metade:

$$segBegin_i = (i - 1)\lambda/2$$

Por fim, se determina o número n de trechos necessários para cobrir o padrão inteiro. Perceba que, em um padrão cíclico, λ deve ser um divisor do período do padrão.

As funções de mapeamento são definidas como:

$$map_i(t) \approx signal(segBegin_i + t \cdot \lambda) \qquad t \in [0, 1]$$

A interpolação é realizada com o auxílio de funções de pertinência:

$$triangle_i(\tau) = \begin{cases} 1 - 2\left|\frac{\tau - segBegin_i}{\lambda} - 0.5\right| & \text{se } 0 < \tau - segBegin_i < \lambda, \\ 0 & \text{caso contrário} \end{cases}$$

Na figura 4.5 um exemplo das funções *triangle*, para $\lambda = 3$ segundos, é apresentado.

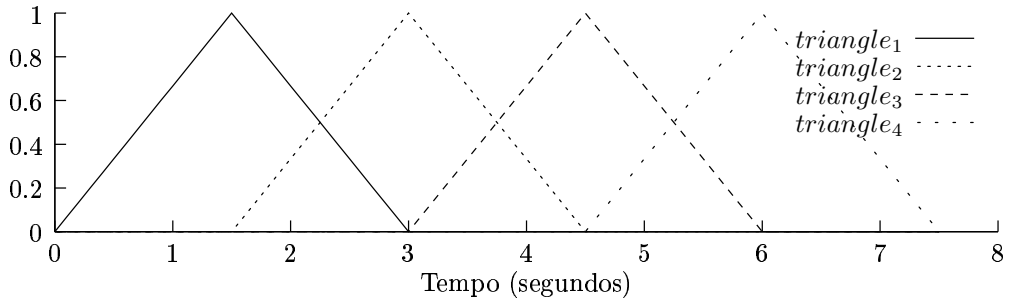

Figura 4.5: Exemplo de funções *triangle* com $\lambda = 3$ segundos.

O sinal pode então ser gerado para qualquer instante τ entre $segBegin_2$ e $segBegin_n + \lambda/2$ através de

$$signal'(\tau) = \sum_{i=1}^{n} map_i\left(\frac{\tau - segBegin_i}{\lambda}\right) \cdot triangle_i(\tau)$$

Como no início e o no final apenas um mapeador fica ativo, há um crescimento gradual da amplitude do sinal para τ entre 0 e $segBegin_2$, e um decrescimento gradual para τ entre $segBegin_n + \lambda/2$ e o final do sinal.

O circuito neural que realiza o mapeamento $map_i(\tau)$ e a modulação por $triangle_i(\tau)$ pode ser visto na figura 4.6. Os neurônios da camada intermediária (neurônios sigmóides 1 a k) representam um *perceptron* de múltiplas camadas, que sabidamente é um aproximador universal de funções (HORNIK; STINCHCOMBE; WHITE, 1990), e seus parâmetros podem ser determinados por *backpropagation* ou outro método de optimização. Usamos para esta tarefa algoritmos genéticos, por se adequar bem em nossa implementação.

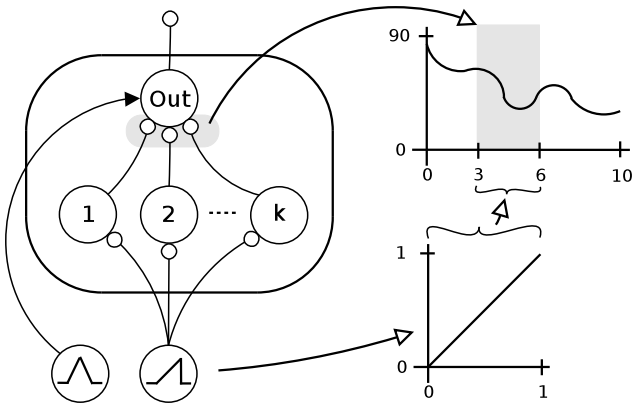

Figura 4.6: Circuito para o mapeamento e interpolação dos sinais e um exemplo de mapeamento (terceiro trecho, $\lambda = 3$)

O padrão inteiro é reproduzido pela ativação sucessiva dos geradores de funções cascateados (neurônio *checkHalf* de um conectado ao *lock* do próximo), conectados aos seus respectivos mapeadores. Padrões de sinais com várias dimensões podem ser gerados apenas conectando mais mapeadores aos geradores de funções, como mostra a figura 4.7. Para padrões cíclicos, basta que se conecte o *checkHalf* do último gerador ao *lock* do primeiro. Para iniciar a reprodução do padrão, é necessário que uma sinapse dê um pulso de ativação no neurônio *lock* do trecho inicial (ou daquele que se deseja começar). Para interromper a reprodução do padrão, deve-se ativar sinapses de potenciação inibitória nos neurônios *lock* de todos os geradores.

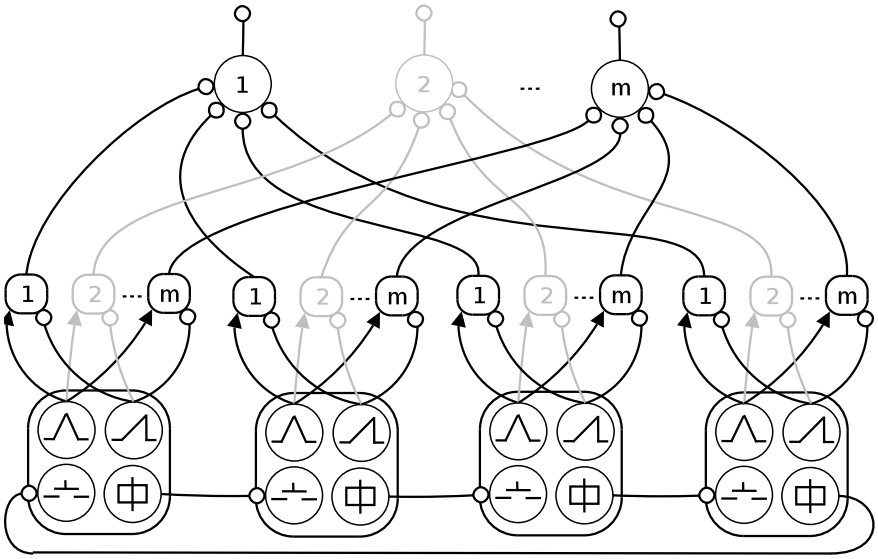

Figura 4.7: Circuito para a geração de um padrão m-dimensional cíclico dividido em quatro trechos (segunda dimensão desenhada em cinza para facilitar a visualização)

5 IMPLEMENTAÇÃO E EXPERIMENTOS

O modelo de rede neural descrito na seção 3.6 foi implementado em C++, na forma de uma biblioteca chamada Psi. Sua interface é extremamente simples, contendo métodos para: interpretar um arquivo com a descrição da arquitetura de uma rede (um *script* Lua); atribuir intensidades de ativação a neurônios sensores; coletar intensidades de ativação de neurônios atuadores; realizar a iteração do fluxo dos sinais na rede. Adicionalmente, há metodos para aplicação de Algoritmos Genéticos sobre os parâmetros dos neurônios (mas não sobre a topologia da rede). Abaixo pode-se ver o código fonte da descrição do neurônio *lock*, descrito na subseção 4.2.1, como exemplo:

```
lock={
   neuron=sigmoid;
   linearSize=0.001;
   range=0.5;
   offsety=0.5;
   offsetx=-0.5;
   synapses={
      lock={ target='lock'; };
      increase={  target='sawTooth'; weight='$tao'; };
      reset={
         target={'sawTooth','checkPhaseEnd','checkPhaseHalf'};
         offset=1; gain=-1;
         potentiator=true; weight=0;
      };
   };
};
```

Um aplicativo foi desenvolvido para gerenciar o fluxo de sinais entre a biblioteca Psi e o simulador Phi, que também foi usado como uma biblioteca. Em sua inicialização ele interpreta a descrição do corpo do robô e do ambiente ao qual ele está submetido, e interpreta a descrição da arquitetura neural descrita na subseção 4.2.2, cujos mapeadores foram evoluídos (conforme descrito na seção 5.2) para reproduzir os padrões criados conforme a seção 5.1. A execução continua com o laço de simulação, e a cada 40ms de tempo simulado o estado do mundo tridimensional é registrado em um arquivo. A simulação encerra por intervenção manual.

O comportamento do robô pode ser observado durante a execução do simulador ou posteriormente usando o arquivo de registro, pelo visualizador *phiviewer*. Os arquivos de registro interessantes para a documentação do projeto foram processados, e vídeos de 25 *frames* por segundo foram criados. Na seção 5.3 é feita uma análise desses vídeos.

5.1 Desenho dos Padrões

A seqüência de movimentos necessária para que um corpo se locomova é altamente dependente de sua forma. Para determinar um padrão adequado ao nosso robô, tomamos como referência algumas seqüências de fotos do fotógrafo inglês Eadweard James Muybridge (1830-1904), que ficou famoso por retratar cenas em movimento (figura 5.1).

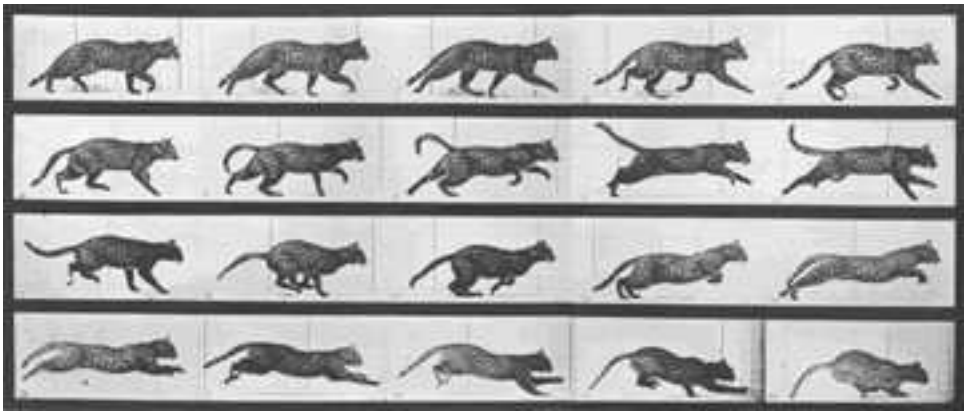

Figura 5.1: Seqüência de fotos de um gato

Fizemos então um aplicativo para que os padrões de ângulos das articulações pudessem ser determinados, a partir do posicionamento de cada uma dessas articulações. As posições são determinadas por curvas de Bézier, que servem como atratores. A melhor configuração de ângulos é aquela com a menor soma das distâncias euclidianas entre a posição das articulações e suas respectivas curvas de Bézier. Por simplicidade, a optimização deste problema foi realizada por Algoritmos Genéticos, onde os cromossomos eram formados pelos ângulos das articulações e o principal operador era o de mutação. A figura 5.2 mostra a edição do movimento dos membros dianteiros.

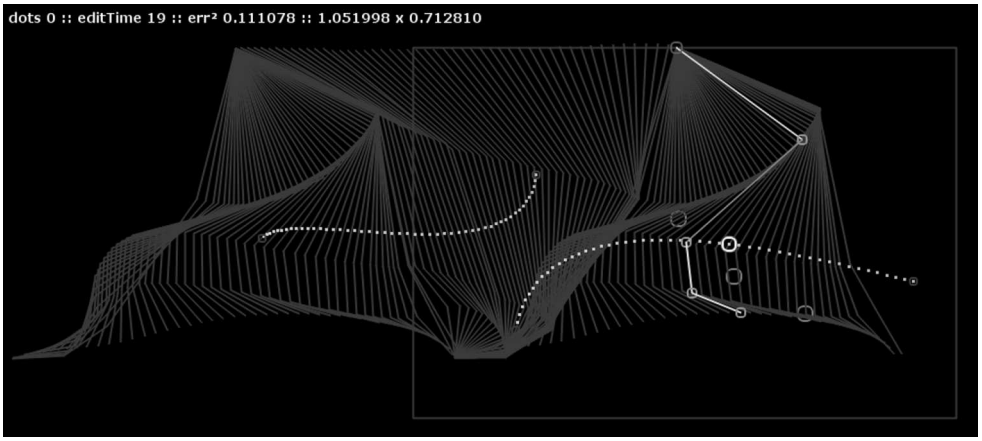

Figura 5.2: Edição do padrão de locomoção dos membros dianteiros

Tendo o padrão de ângulos completo (figura 5.3), pôde-se concluir a arquitetura neural, com a adaptação dos mapeadores.

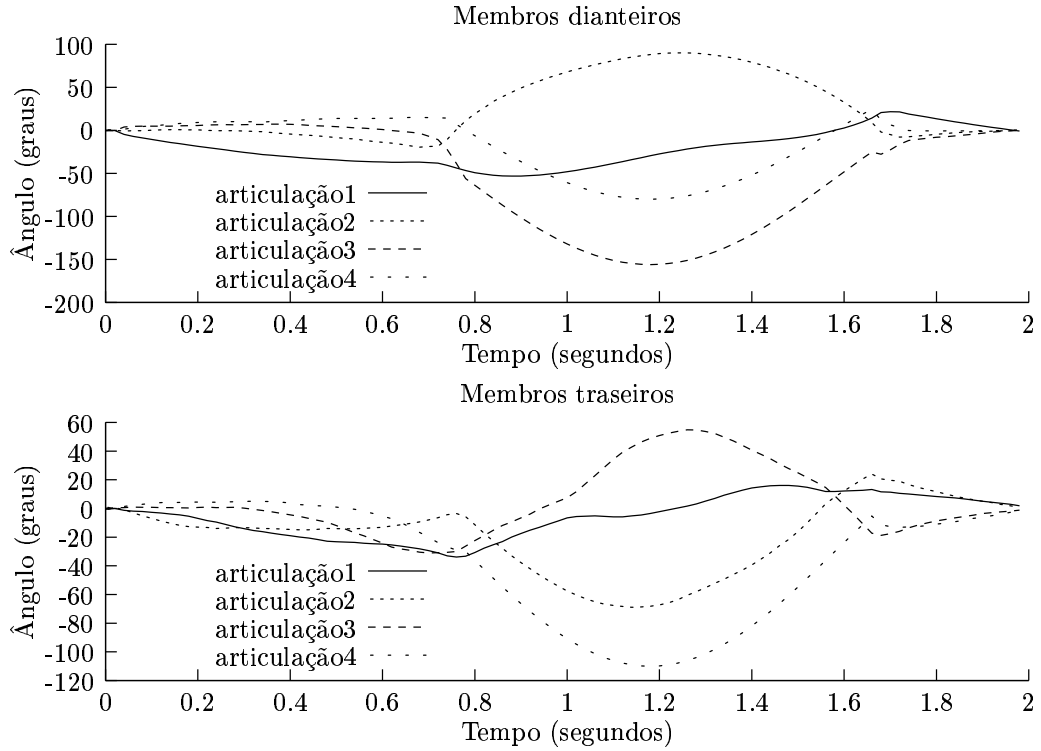

Figura 5.3: Padrões de sinais dos membros dianteiros e traseiros

5.2 Evolução dos Mapeadores

Um aplicativo foi desenvolvido especificamente para a evolução dos pesos das sinapses nas redes neurais mapeadoras, usando a biblioteca Psi como base. Sua funcionalidade consistia em:

- abrir um arquivo com a descrição da rede neural, o qual marcava certos parâmetros dos neurônios como pertencentes a um genoma, e conseqüentemente passíveis de alteração pela rotina de adaptação.
- ler um arquivo de código genético previamente salvo, caso existisse, para continuar a sua evolução. Cada gene era referenciado por um identificador único, como $cns\_walk\_p1\_begin\_modulator\_adaptor1\_base$.
- ler o arquivo com a referência dos padrões de locomoção.
- ativar sinais de entrada apropriados na rede, e coletar suas saídas para posterior comparação.
- aplicar o algoritmo genético até que o fim da execução do programa fosse manualmente solicitado. A função de avaliação usada calculava a soma das distâncias euclidianas entre o padrão gerado pela rede e o padrão de referência, quanto menor esse valor, mais adaptado o indivíduo se encontrava. Os principais operadores de busca usados foram a mutação e o elitismo (manter o código genético dos vencedores para competir com os mutados).
- salvar periodicamente o código genético do melhor indivíduo encontrado até então.
- apresentar graficamente o padrão gerado pelo melhor indivíduo.

Após 19.114.487 gerações os padrões estavam suficientemente próximos dos padrões de referência, e pôde-se começar a fase de experimentação.

5.3 Experimentos

Todos os vídeos dos experimentos estão disponíveis no CD anexo. A listagem dos arquivos e os meios de visualizá-los estão descritos no apêndice A.

O primeiro teste realizado, documentado no arquivo 20040620-1-Skating.avi, foi feito usando apenas o padrão de caminhada dos membros traseiros. Às quatro patas e ao solo foi atribuído um coeficiente de atrito infinito, de forma a não haver deslizes, e como os membros dianteiros não deveriam se mover, foi necessário que se fixasse uma pequena prancha com atrito zero, para que pudesse se deslocar para frente. Há também uma borda imperceptível aos lados do caminho onde o

robô anda, para que as movimentações laterais fossem contidas. O resultado foi bem animador, pois provou que a arquitetura neural gerava os padrões de forma satisfatória.

O segundo teste, 20040620-2-NoSlip.avi, mostrou que o coeficiente de atrito infinito impedia que o robô se deslocasse livremente, usando o padrão de movimentação que havia sido projetado (figura 5.4). O teste seguinte, 20040622-1-OnIce.avi, foi realizado para verificar o que aconteceria no outro extremo, com ausência total de atrito. Em ambos os casos o equilíbrio não era mantido e o robô caía.

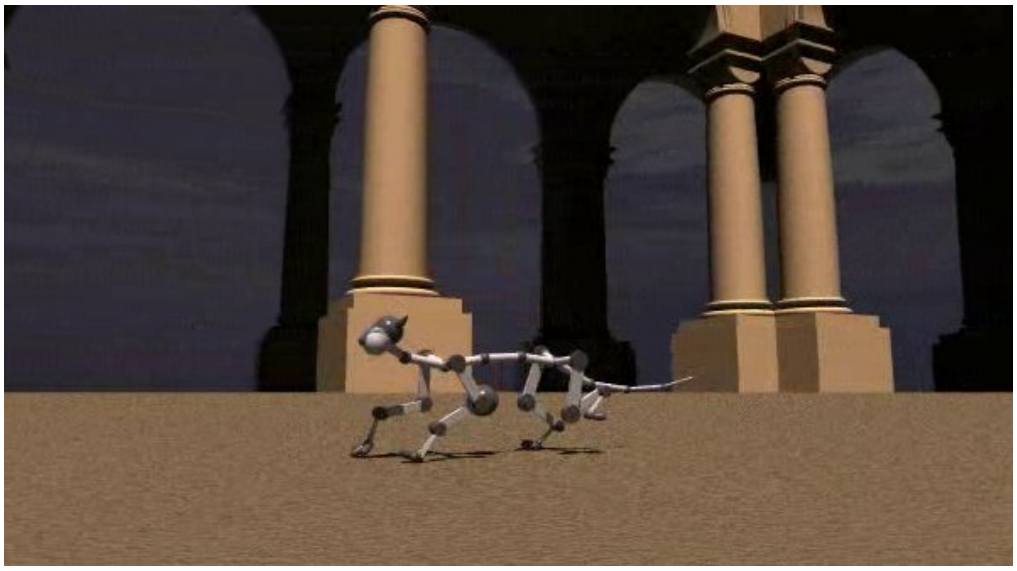

Figura 5.4: Tentativa frustrada de caminhada (20040620-2-NoSlip.avi)

Finalmente, no teste 20040622-2-Walk.avi, o coeficiente de atrito foi configurado de forma a permitir que a pata escorregasse um pouco, mas após a velocidade aumentar, a força imprimida sobre o solo era suficiente apenas para impulsionar o corpo para frente. O vídeo 20040622-3-Walk.avi mostra exatamente a configuração anterior, mas com a freqüência de reprodução do padrão dobrada. Estes dois testes mostram que é possível uma locomoção quadrúpede relativamente estável usando apenas geração de sinais, sem modulações.

Experimentando com outros ritmos de passadas, fizemos o robô correr no teste 20041009-1-Run.avi (figura 5.5). Neste caso percebe-se a necessidade de modulações provenientes de circuitos que controlam o equilíbrio, pois em pouco tempo de ação o robô perde a sincronia com o momento certo de exercer a força sobre o solo, e acaba tropeçando.

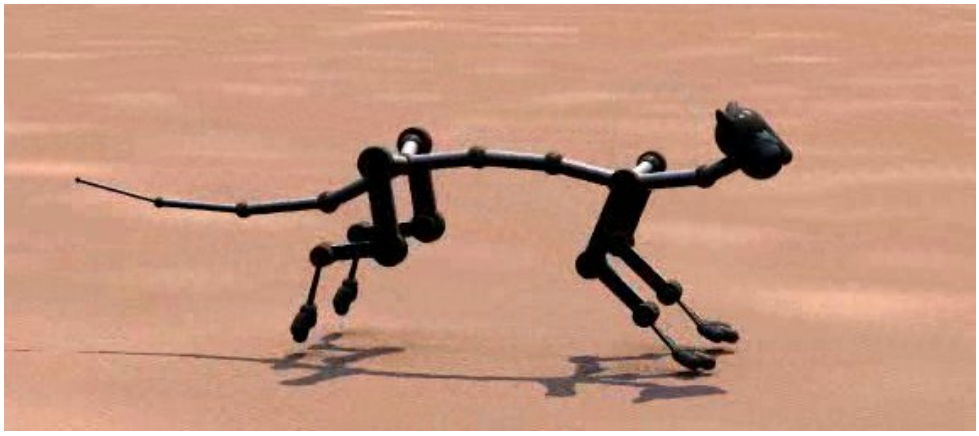

Figura 5.5: Robô correndo (20041009-1-Run.avi)

Em todos esses testes, as únicas articulações que tinham liberdade de movimento eram as dos membros. No teste 20041015-1-Ginga.avi se permitiu que as articulações da coluna fossem mais elásticas, resultando num movimento visualmente muito mais natural.

Uma característica interessante que o modelo de gerador de padrões usado apresenta e que o diferencia dos demais é a capacidade de controlar a velocidade da reprodução independentemente da velocidade do padrão usado no treinamento. Foram feitos alguns testes em que a velocidade era controlada por *joystick*, permitindo uma desaceleração até a suspensão temporária completa do movimento, ou uma aceleração até uma velocidade relativamente grande, limitada apenas pelas características físicas do robô.

A desvantagem mais significativa apresentada pelo modelo, na forma como está, foi a incapacidade de adaptar a fase do padrão a ser reproduzido conforme o estado inicial dos membros, algo que um modelo baseado em atratores, como o de Srinivasan, trata naturalmente.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho serviu para fundar as bases sobre as quais o projeto de simulação de um sistema nervoso será desenvolvido, e foi bem sucedido em seu objetivo. Um corpo com movimentos básicos, gerados por um sistema nervoso extremamente simples, está funcional.

O desenvolvimento do corpo tornou necessário que se desenvolvessem ferramentas adequadas para sua simulação, resultando em uma poderosa aplicação para simulação robótica, o Phi, e na publicação (BASSO, 2005).

A arquitetura neural que desenvolvemos resultou na publicação (BASSO; ENGEL, 2005), e apresenta uma técnica original para a solução do problema de geração de padrões.

O próximo item a ser integrado ao modelo será o controle de reflexos vestibulares anti-gravitacionais, para possibilitar que a locomoção se realize em terrenos inclinados, e para manter a estabilidade como um todo. A seguir, pretende-se adicionar a modulação por circuitos supra-espinhais, como o controle de velocidade, ritmo e direção (tronco encefálico), e optimização do custo do movimento (cerebelo). Uma vez que estes sistemas estejam funcionais, será abordado o problema do planejamento motor.

Vários obstáculos serão encontrados durante este desenvolvimento, talvez até forçando que se tome um caminho substancialmente diferente deste planejado. Não há dúvidas, porém, que as soluções que puderem ser encontradas serão de grande valor para a comunidade científica.

A inclusão de novas habilidades ao modelo, como o sistema de equilíbrio citado acima, fará com que seja necessário modificar o corpo, com a adição de um sistema vestibular (nos termos de um robô real: giroscópio e acelerômetro). Esta observação serve para que se perceba que apesar deste trabalho ter feito os elementos que serão usados como base para trabalhos futuros, estes elementos não devem ser imutáveis. O conhecimento adquirido com a pesquisa de níveis mais altos proporcionará idéias de melhorias sobre os níveis mais baixos, e estas melhorarão o funcionamento dos níveis mais altos, recursivamente. Mas ao contrário do

que acontece nos sistemas *top-down*, nossas mudanças seguem uma especificação validada por centenas de milhões de anos: o sistema nervoso animal.

REFERÊNCIAS

AMES, J. C. **Design Methods for Pattern Generator Circuits**. 2003. Dissertação (Mestrado em Ciência da Computação) — Case Western Reserve University.

ASHLEY, S. Músculos Artificiais. **Scientific American Brasil**, [S.l.], v.18, p.50–57, 2003.

BASSO, D. M. Phi: ferramentas para simulação robótica. In: WORKSHOP SOBRE SOFTWARE LIVRE, WSL, 6., 2005. **Anais...** [S.l.: s.n.], 2005. p.279–284.

BASSO, D. M.; ENGEL, P. M. Uma Abordagem Conexionista para Geração de Padrões. In: ENCONTRO NACIONAL DE INTELIGÊNCIA ARTIFICIAL, ENIA, 5., 2005. **Anais...** [S.l.: s.n.], 2005. p.732–741.

BOHET, S. M. **Spiking Neural Networks**. 2003. Tese (Doutorado em Ciência da Computação) — University of Leiden.

DARWIN, C. **The origin of species**: by means of natural selection. [S.l.]: Prometheus Books, 1991.

FIELDS, R. D. Memórias que Ficam. **Scientific American Brasil**, [S.l.], v.3, n.34, p.60–67, 2005.

FIGUEIREDO, L. H. de; IERUSALIMSCHY, R.; CELES, W. Lua: an extensible embedded language. **Dr. Dobb's Journal**, [S.l.], v.21, n.12, p.26–33, 1996.

FUNAHASHIM, K.; NAKAMURA, Y. Approximation of dynamical systems by continuous time recurrent neural networks. **Neural Networks**, [S.l.], v.6, p.801–806, 1993.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks. **Neural Networks**, [S.l.], v.3, p.551–560, 1990.

JORDAN, M. I. Attractor dynamics and parallelism in a connectionist sequential machine. In: ANNUAL CONFERENCE OF THE COGNITIVE SCIENCE SOCIETY, 8., 1986. **Proceedings...** [S.l.: s.n.], 1986. p.531–546.

KANDEL, E.; SCHWARTZ, J.; JESSELL, T. **Principles of Neural Science**. [S.l.]: Mc.Graw Hill, 2000.

KOHONEN, T. **Self-organization and Associative Memory**. [S.l.]: Springer-Verlag, 1989.

LUA Webpage. Disponível em: <http://www.lua.org>, acesso em: 21 de Julho de 2005.

MASS, W. Networks of spiking neurons: the third generation of neural network models. **Neural Networks**, [S.l.], v.10, n.5, p.1659–1671, 1997.

MATURANA, H. **Da Biologia à Psicologia**. [S.l.]: Artes Médicas, 1998.

MCCULLOCH, W.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, [S.l.], n.5, p.115–133, 1943.

MICHEL, O. Webots: professional mobile robot simulation. **International Journal of Advanced Robotics Systems**, [S.l.], v.1, n.1, p.39–42, 2004.

ODE Webpage. Disponível em: <http://www.ode.org>. Acesso em: 21 jul. 2005.

PHI Webpage. Disponível em: <http://basso.inf.br/phi/>. Acesso em: 21 jul. 2005.

POVRAY Webpage. Disponível em: <http://www.povray.org>. Acesso em: 21 jul. 2005.

SDL Webpage. Disponível em: <http://www.libsdl.org>. Acesso em: 21 jul. 2005.

SRINIVASAN, S.; GANDER, R.; WOOD, H. A movement pattern generator model using artificial neural networks. **IEEE Trans. Biomed. Eng.**, [S.l.], v.39, n.7, p.716–22, July 1992.

WILSON, D. M. The central nervous control of flight in a locust. **J. Exp. Biol.**, [S.l.], n.38, p.471–490, 1961.

APÊNDICE A CD ANEXO

A.1 Listagem dos Arquivos

```
20030919-1.avi  20040427-1.avi                  20041012-1-RunOnIce.avi
20030920-1.avi  20040428-1-Soccer.avi           20041013-1-TheCity.jpg
20030920-2.avi  20040501-1.avi                  20041013-2-TheCity2.jpg
20030923-1.avi  20040502-1.avi                  20041013-3-TheCity3.jpg
20031019-1.png  20040503-1.avi                  20041013-4-TheCity4.jpg
20031019-2.png  20040620-1-Skating.avi          20041013-5-TheCity5.jpg
20031019-3.png  20040620-2-NoSlip.avi           20041014-1-TheCity6.jpg
20031019-4.png  20040622-1-OnIce.avi            20041014-2-TheCity7.jpg
20031021-1.avi  20040622-2-Walk.avi             20041015-1-Ginga.avi
20031127-1.avi  20040622-3-Walk.avi             20050317-1-Nyau.jpg
20031203-1.avi  20040622-4-Ramp.avi             20050322-1-DorsalMuscles.jpg
20031203-1.png  20040622-5-testWalkFast.avi     20050322-2-SpinalMuscles.jpg
20031203-2.avi  20040622-6-testWalkSlowRamp.avi 20050322-3-NeckMuscles.jpg
20031203-2.png  20040623-1-Door.avi             20050322-4-MuscleControl.jpg
20031204-1.avi  20040705-1-NewtonCradle.avi     20050403-1-Dominos.avi
20031204-2.avi  20040706-1-Dominos.avi          20050403-2-Robot.avi
20031204-3.avi  20040714-1-Bouncer.avi          20050403-3-Camera.avi
20031205-1.png  20040717-1-3d.jpg               20050404-1-Car.avi
20031205-2.png  20040720-1-Dominos.avi          Dissertacao.pdf
20040425-1.avi  20040720-2-Dominos.avi          MPlayer-1.0pre7.tar.bz2
20040425-2.avi  20041009-1-FirstRun.png
20040425-3.avi  20041009-1-Run.avi
20040426-1.avi  20041010-1-TrotOnIce.avi
```

Os vídeos também estão disponíveis na galeria de mídia da página de internet do simulador Phi (PHI Webpage, 2005).

A.2 Reprodução dos Vídeos

Todos os vídeos foram comprimidos no formato MPEG4, e podem ser reproduzidos no GNU/Linux usando o aplicativo MPlayer, cujo pacote com códigos fontes e instruções de instalação encontra-se no CD. Para a reprodução dos vídeos no Windows recomenda-se a instalação do *codec* DivX, disponível em http://www.divx.com.