ESTE EXEMPLAR CORRESPONDE À REDAÇÃO FINAL D
TESE DEFENDIDA POR João Bosco
GONÇALVES E APROVADA PEL
COMISSÃO JULGADORA EM 06 12 2004

ORIENTADOR

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

Desenvolvimento de um Sistema de Controle Adaptativo e Integrado para Locomoção de um Robô Bípede com Tronco

Autor: João Bosco Gonçalves
Orientador: Prof. Dr. Douglas Eduardo Zampieri
Co-orientador:

12/2004

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE ADAPTATIVO E INTEGRADO PARA LOCOMOÇÃO DE UM ROBÔ BÍPEDE COM TRONCO

Autor: João Bosco Gonçalves
Orientador: Prof. Dr. Douglas Eduardo Zampieri

Curso: Engenharia Mecânica
Área de Concentração: Mecânica dos Sólidos e Projeto Mecânico

Tese de doutorado apresentada à comissão de Pós Graduação da Faculdade de Engenharia Mecânica, como requisito para a obtenção do título de Doutor em Engenharia Mecânica.

Campinas, 2004.
S.P. – Brasil.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

G586d

Gonçalves, João Bosco
Desenvolvimento de um sistema de controle adaptativo e integrado para locomoção de um robô bípede com tronco / João Bosco Gonçalves.--Campinas, SP: [s.n.], 2004.

Orientador: Douglas Eduardo Zampieri.
Tese (Doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Robôs – Sistemas de controle. 2. Redes neurais(Computação). I. Zampieri, Douglas Eduardo. II. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. III. Título.

Título em inglês: Development of an integrated adaptive control system for a biped robot with a trunk
Palavras-chave em Inglês: Robot control, Atificial neural networks, Robot biped
Área de concentração: Controle de sistemas dinâmicos
Titulação: Doutor em Engenharia Mecânica
Banca examinadora: Douglas Eduardo Zampieri, Atair Rios Neto, Eurípedes Guilherme de Oliveira Nóbrega, Fernando Von Zuben e Luiz Sandoval Góes.
Data de defesa: 06/12/2004

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

TESE DE DOUTORADO

Desenvolvimento de um Sistema de Controle Adaptativo e Integrado para Locomoção de um Robô Bípede com Tronco

Autor: **João Bosco Gonçalves**
Orientador: **Prof. Dr. Douglas Eduardo Zampieri**

Prof. Dr. Douglas Eduardo Zampieri, Presidente
DMC/FEM/UNICAMP

Prof. Dr. Atair Rios Neto
EMBRAER

Prof. Dr. Euripedes Guilherme de Oliveira Nobrega
DMC/FEM/UNICAMP

Prof. Dr. Fernando Von Zuben
FEEC/UNICAMP

Prof. Dr. Luiz Sandoval Góes
ITA

Campinas, 06 de dezembro de 2004

Dedicatória:

Dedico este trabalho ao Dr. Messias Gonçalves e Silva (*in memoriam*).

Agradecimentos

Ao Professor Dr. Douglas Eduardo Zampieri que não mediu esforços durante a sua orientação e que, com muita paciência, dedicação e profissionalismo, traçou as linhas mestras para a concretização deste trabalho.

Ao Professor Dr. Pedro Paulo Leite do Prado pelo constante incentivo e brilhantes sugestões que muito enriqueceram o desenvolvimento deste trabalho.

Ao Professor MC Domingos Sávio Aguiar que, com a competência que lhe é peculiar, muito me auxiliou na elaboração do projeto mecânico do robô bípede.

Ao Professor Dr. Álvaro Manoel Soares pelos momentos memoráveis de discussões sobre a condução deste trabalho.

Aos Professores membros da banca cujas contribuições permitiram enriquecer ainda mais o texto.

Á Universidade de Taubaté pelo aporte financeiro e pelos recursos laboratoriais gentilmente fornecidos.

Abstract

GONÇALVES, João Bosco, *Development of an Integrated Adaptive Control System for a Biped Robot With a Trunk*, Campinas: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2004. 258 p. Tese (Doutorado)

The main objective of this work is to project a biped robot machine with a trunk. The mathematical model was realized by considering two sub-systems: the legs and the trunk. The trajectories of the trunk are planned to compensate torques inherent to the dynamic gait, permitting to preserve the dynamic balance of the biped robot. An automatic generator of trajectory for the trunk was developed that processes the information of positions and accelerations imposed to the legs. A gait generator was developed that uses the capacity of the biped robot to execute three-dimensional movements, causing a steady dynamic gait without the effective use of the trunk. The automatic generator of trajectory for the trunk actuates, if the generated do not keep the dynamic balance, reestablishing he steady dynamic gait. A neural network reference model for the adaptive control was projected, which utilizes an RBF neural network and a stability evaluation is based on the criterion of Lyapunov. The system of control and the automatic generator of trajectories for the trunk are integrated, composing the adaptive mechanisms developed to solve the way of dynamic walking.

***Keywords*:** zero moment point, gait synthesis, robot biped, postural stability, artificial neural network.

Resumo

GONÇALVES, João Bosco, *Desenvolvimento de um Sistema de Controle Adaptativo e Integrado para Locomoção de um Robô Bípede com Tronco*, Campinas: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2004. 255 p. Tese (Doutorado)

Este trabalho concebeu um robô bípede composto por uma sucessão de elos rígidos interconectados por 12 articulações rotativas, permitindo movimentos tridimensionais. O robô bípede é constituído por dois subsistemas: *tronco* e *membros inferiores*. A modelagem matemática foi realizada em separado para cada um dos subsistemas, que são integrados pelas forças reativas de vínculo. Nossa proposta permite ao robô bípede executar a andadura dinâmica utilizando o tronco para fornecer o balanço dinâmico (estabilidade postural). De forma inédita, foi desenvolvido um gerador automático de trajetória para o tronco que processa as informações de posições e acelerações impostas aos membros inferiores, dotando o robô bípede de reflexos. Foi desenvolvido um gerador de marcha que utiliza a capacidade do robô bípede de executar movimentos tridimensionais, implicando andadura dinamicamente estável sem a efetiva utilização do tronco. O gerador automático de trajetória para o tronco entra em ação se a marcha gerada não mantiver o balanço dinâmico, restabelecendo uma marcha estável. Foi projetado um sistema de controle adaptativo por modelo de referência que utiliza redes neurais artificiais. A avaliação de estabilidade é feita segundo o critério de *Lyapunov*. O sistema de controle e o gerador automático de trajetórias para o tronco são integrados, compondo os mecanismos adaptativos desenvolvidos para solucionar o modo de andar dinâmico.

Palavras-chaves: ponto de momento nulo (ZMP), planejamento de andadura, robô bípede estabilidade postural, redes neurais artificiais.

Índice

Lista de Figuras

Lista de Tabelas

Nomenclatura

Letras Latinas

$\underline{\mathbf{A}}$	Matriz associada ao vetor de estado, modelo de referência.
$\underline{\mathbf{B}}$	Matriz associada ao vetor de excitação, modelo de referência.
$\mathbf{A}$	Matriz associada ao vetor de estado.
a_k	Parâmetro de Denavit-Hartenberg associado a k-ésima junta
$\mathbf{B}$	Matriz associada ao vetor de excitação.
$\mathbf{C}$	Vetor de forças centrifugas e de *Coriolis*
$\mathbf{D}$	Matriz de inércia.
d_k	Parâmetro de Denavit-Hartenberg associado a k-ésima junta
$\mathbf{e}$	Vetor de erro de rastreamento.
e_k	k-ésimo elo
F	Força
f	Freqüência de oscilação.
$\mathbf{F}$	Vetor de acoplamento dissipativos.
g	Aceleração da gravidade
$\mathbf{G}$	Vetor de carregamento gravitacional.
g^o	Função de ativação da camada intermediária
g^s	Função de ativação da camada de saída
j_k	k-ésima junta
m	Massa
M	Quantidade de pontos-via que descrevem uma certa trajetória.

P	Peso total do robô
p	Vetor posição
p_c	Taxa de cruzamento.
PD	Pé direito
PE	Pé esquerdo
p_m	Taxa de mutação.
q	Vetor de ângulos relativos.
q_j^e	Variável da j-ésima junta, relativa ao enésimo elo
R	Matriz de rotação
s	Polarização
T	Matriz de transformação homogênea entre elos consecutivos
T	Matriz de transformação homogênea entre elos sucessivos
t	Variável para o tempo
u	Saída da camada de intermediaria
u	Vetor de controle.
v	Pesos da camada de saída
y	Variável de saída

Letras Gregas

$\aleph$	Tensor de inércia, kg m^2
$\underline{\Omega}$	Vetor de variáveis de estado, associado ao modelo de referência
λ	Comprimento do passo, m
$\Re$	Conjunto dos números reais
ξ	Fator de amortecimento.
σ	Fator de escala
ω	Freqüência natural, rad/s
$\Im$	Funcional de otimização
ε	Identificador do tipo de junta, igual a 1 se, revoluta ou zero caso contrário

ρ	Massa específica, kg/m^3
Φ	Matriz de regressão
η	Momento, N m
β	Momentum
δ	Parâmetro de projeto da rede neural, critério de parada
μ	Parâmetro de projeto, ponderação dos funcionais de otimização.
δ	Período, s
θ	Posição angular, rad
α	Taxa de aprendizagem de uma rede neural
μ	Valor Médio
ϖ	Velocidade angular, rad/s
υ	Velocidade de trajeto na direção horizontal, m/s.
τ	Vetor de excitação.
Γ	Vetor de forças de reação, N
Δ	Vetor de incertezas paramétricas
Δ	Vetor de Perturbações
Γ	Vetor de termos não-lineares associados ao modelo do robô bípede.
Ω	Vetor de variáveis de estado
υ	Volume do corpo rígido, m^3
$\rho(.)$	Função de ativação de base radial
∇J	Gradiente
α_k	Parâmetro de Denavit-Hartenberg associado a k-ésima junta
θ_k	Parâmetro de Denavit-Hartenberg associado a k-ésima junta
τ_M	Termo de excitação, baseado no modelo matemático do robô bípede.
τ_R	Termo de robustez, visa eliminar os desvios no processo de identificação.
v_s	Velocidade horizontal, m/s
${}^{k-1}\Delta_k$	Deslocamento entre a k-ésima junta e a k-ésima mais um
${}^{k-1}\Delta_k$	Deslocamento entre a k-ésima junta e a k-ésima mais um

Abreviações

c_θ	$\cos(\theta)$
s_θ	$\sin(\theta)$
EP	Programação Evolutiva
GA	Algoritmo Genético
SCG	Sistema de Coordenadas Global (ou Inercial)
sgn()	Função sinal.

Siglas

CoG	Centro de Gravidade
CoM	Centro de Massa
CoP	Centro de Pressão
GAT	Gerador Automático de Marcha
GCoM	Projeção do centro de massa no solo
ZMP	Ponto de momento nulo (*Zero moment point*)

Capítulo 1
Introdução

1. Robôs Bípedes e Aspectos Motivadores deste Trabalho

Locomoção é um processo que fornece autonomia de deslocamento a robôs móveis. Há diversas estratégias de locomoção propostas e implementadas. Robôs terrestres podem mover-se por intermédio de rodas ou pernas, havendo diferentes estratégias para uma mesma aplicação. Por exemplo, dependendo do número de pernas adotadas. Contudo, há considerações que acabam por definir uma determinada estratégia, como, por exemplo, a rapidez com que o robô deverá se mover; o tipo de superfície em que irá operar; a existência de obstáculos, etc.

Robô bípede pode ser classificado pelo modo de andar, destacando-se o andar passivo (*passive walking*) e o andar ativo (*active walking*). No andar passivo o robô bípede se movimenta em declives e é impulsionado apenas pelas forças gravitacionais. No andar ativo, o robô bípede é provido de motores, permitindo autonomia de movimentos. Em ambos os casos, são possíveis o andar estático ou o andar dinâmico.

Considere-se a projeção do centro de massa global GCoM (*ground projection of the center of mass*) do robô bípede sobre o solo (superfície de apoio). Se a projeção estiver circunscrita aos contornos da área de contato entre o pé de apoio e o solo (região de estabilidade), o robô bípede apresentará o andar estático. Caso contrário, apresentará o andar dinâmico. A amostragem de

abordagens da literatura especializada mostra que há preferências pelo modo de andar ativo e dinâmico.

Neste trabalho, é objetivo conceber um robô bípede, propor e implementar estratégias que assegurem o modo de andar dinâmico. A abordagem de assegurar o modo de andar dinâmico e de gerar trajetórias desejadas (quaisquer) para os membros inferiores permite dotar o robô bípede de tronco (pêndulo invertido) visando compensar forças inerciais e gravitacionais inerentes ao modo de andar dinâmico (Kuo, 1999; Sugihara *et al.*, 2002). A trajetória do tronco pode ser determinada considerando a dinâmica de contato entre o pé e a superfície de apoio. A modelagem do tronco deve levar em consideração a dinâmica dos membros inferiores. Assim, é estabelecido um sistema de equações diferenciais não lineares e acopladas, a partir do qual a trajetória do tronco pode ser determinada, para quaisquer trajetórias planejadas para os membros inferiores. A Figura 1.1 descreve o problema correspondente ao modo de andar dinâmico.

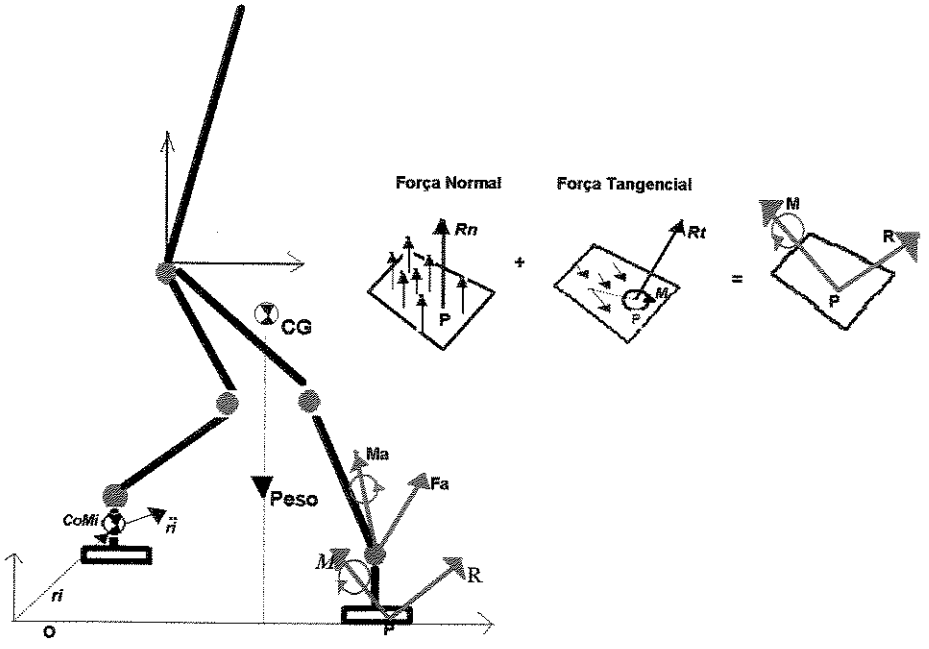


Figura 1.1 – Descrição do problema relativo ao modo de andar dinâmico

Para cada elo, são medidas as forças inerciais e gravitacionais em seu correspondente centro de massa (CoM), em relação a um sistema de coordenada inercial O. A interação entre o pé de apoio e o solo produz forças dinâmicas de reação contidas na área de contato entre o pé de

apoio e o solo (região de estabilidade). Essas forças reativas possuem componentes normais e tangenciais que podem ser descritas por um momento (M) e por uma força (R), aplicadas ao ponto P (centro de pressão, CoP). Em contrapartida, toda a dinâmica do robô bípede pode ser descrita por um momento (Ma) e por uma força (Fa) aplicadas ao tornozelo (como indicado na Figura 1.1). Considera-se que o pé de apoio está em repouso, analisando as componentes das forças reativas e das forças devidas à dinâmica do robô bípede, em relação ao ponto P, pode-se concluir que:

1. A componente de momento na direção vertical deve estar contrabalançada pelas componentes de Fa e Ma, adicionada à fricção entre o pé e o solo;

2. As componentes de Fa e Ma, nas direções do plano de apoio em relação ao ponto P, devem ser nulas para garantir o balanço dinâmico do robô bípede (estabilidade postural), evitando rotações do pé de apoio.

Da segunda conclusão resulta a seguinte definição: O ponto P designa o ponto de momento nulo (ZMP, *Zero Moment Point*) nas direções do plano de apoio (XY), tal que a somatória de momentos M_x e M_y, resultantes das forças inerciais e gravitacionais, são nulas.

O principal objetivo deste trabalho é contribuir na área de robôs bípedes que exploram o modo de andar dinâmico. Foi concebido um robô bípede dotado de tronco, empregando um sistema de controle adaptativo por modelo de referência baseado em redes neurais artificiais de função de base radial. Esse tipo de rede neural permite a atualização de seus pesos de modo a garantir a estabilidade do sistema em malha fechada, segundo *Lyapunov*. Agregado à malha de controle, foi projetado e implementado um gerador automático de trajetória para o tronco que emprega uma rede neural artificial MLP e permite atualizar as condições de posição e velocidade para o tronco, a partir da evolução temporal dos membros inferiores do robô bípede. Um gerador automático de marcha adaptável às condições locais do terreno foi concebido e implementado, baseado em algoritmos genéticos.

2. Estado da Arte em Robôs Bípedes no Modo de Andar Dinâmico

Esta seção tem por objetivo apresentar uma amostragem da pesquisa bibliográfica que foi realizada sobre robôs bípedes que exploram o andar dinâmico. Para facilitar a apresentação, os trabalhos foram divididos em três grandes áreas de pesquisas: indicadores de estabilidade postural, síntese de marcha dinâmica e projeto de sistemas de controle.

2.1. Indicadores de Estabilidade Postural

Goswami (1999) avaliou o problema de rotação do pé durante a fase de mono-apoio. A motivação desse trabalho está no fato de que a rotação do pé é um indicador da instabilidade postural, que deve ser levado em consideração quando uma marcha dinamicamente estável é planejada. Foi proposto um indicador da margem de estabilidade, *FRI point* (acrônimo de *foot-rotation indicator*), definido como sendo um ponto sobre o solo (superfície de contato) onde atuam as forças de reação, decorrentes do contato do pé sobre o solo. Para manter o pé em equilíbrio dinâmico, esse ponto deverá estar contido no polígono suporte definido pela área de contato entre o pé e o solo (região de estabilidade). Porém o *FRI point* pode estar localizado além dos limites do polígono suporte, cuja distância pode ser utilizada como um indicador da margem de estabilidade postural.

Tipicamente, movimentos do tronco são utilizados para garantir estabilidade postural a robôs bípedes (Kajita *et al.*, 1991; Park *et al.*, 1998; Takanishi, 1989; entre outros). O movimento do tronco pode ser computado por intermédio do ZMP (acrônimo de *zero-moment point*), definido por um ponto circunscrito ao polígono suporte para o qual o momento total, devido às forças inercial e gravitacional, é nulo. Diversos pesquisadores empregaram o ZMP como um critério para garantir estabilidade postural (Yamaguchi *et al.*, 1993; Li *et al.*, 1992; entre outros). Em suma, o algoritmo utilizado pode ser descrito por três passos: planejar uma trajetória para o ZMP, planejar os movimentos para os membros inferiores do robô bípede; e computar a trajetória do tronco a partir das informações geradas nos passos acima. Em geral, a solução do último passo é obtida por um algoritmo iterativo que descreve uma solução periódica para o movimento do tronco em termos da Série de *Fourier* (Takanishi, 1989).

2.2. Planejador de Marcha

Em Huang *et al.* (2001) é proposto um método para planejar marchas, avaliando-se as condições do solo e a estabilidade postural. As especificações dos atuadores foram previstas para as marchas avaliadas. São formuladas as equações de movimentos e restrições que devem ser impostas ao pé em balanço e ao quadril, para que o robô bípede possa se locomover em diversos tipos de ambientes – solos com desníveis, com escadas, ou com obstáculos. Diferentes tipos de movimentos para o pé em balanço podem ser produzidos variando alguns parâmetros restritivos, por exemplo, o comprimento do passo ou a altura máxima em relação ao solo. As trajetórias do pé em balanço e do quadril são planejadas no espaço cartesiano. As posições angulares são obtidas via cinemática inversa. Entretanto, as especificações restritivas e a geração da trajetória para o pé em balanço são realizadas de início. Então, a trajetória do quadril é computada por um algoritmo iterativo, que visa obter a maior margem de estabilidade possível. A busca dessa trajetória está baseada na variação de apenas dois parâmetros, definidos no plano sagital e na direção horizontal, que definem a localização do quadril nas fases inicial e final da marcha, em relação ao sistema inercial. A cinemática inversa é aplicada e são definidas as velocidades e as acelerações angulares por intermédio de interpolação polinomial. Após, o critério ZMP é computado. A iteração do algoritmo é finalizada quando os dois parâmetros são incrementados em todos os seus limites. Então, seleciona-se a trajetória do quadril que melhor satisfaça a condição de estabilidade postural (ZMP). Simulações computacionais foram realizadas para validar o algoritmo proposto, utilizando-se informações físicas de um robô bípede experimental cujo peso é de 83 kg e com 12 graus de liberdade. Testes experimentais também foram realizados, revelando a necessidade de combinar o planejador de marcha com um sistema de controle em tempo real.

Em Marchese *et a.l.* (2001) é apresentado e discutido um procedimento para planejar os movimentos de um robô bípede modelado como uma cadeia cinemática aberta de sete elos, interconectados e atuados por seis juntas rotacionais, incluindo o tronco. O algoritmo desenvolvido permite gerar trajetórias dinamicamente estáveis e restringe-se aos movimentos do robô no plano sagital, durante a fase mono-apoiado. Essas duas hipóteses são consideradas consistentes com o modo de andar humano, cujos movimentos mais característicos ocorrem no

referido plano. Salienta-se que, nessa fase, o problema de estabilidade postural é mais relevante. Em resumo, são impostas as trajetórias para as juntas do tornozelo em balanço e para a junta do tronco, mantendo-o sempre perpendicular ao solo. As trajetórias para as demais juntas são calculadas a partir de especificações de posições no plano cartesiano para o pé em balanço e o quadril e, então, a cinemática inversa é empregada. Especificações de velocidades no plano cartesiano são também incluídas, e, por intermédio de Jacobianos, são calculadas as respectivas velocidades angulares. Um sistema de controle proporcional retro alimentado é utilizado, visando garantir rastreamento da trajetória planejada. O algoritmo proposto foi verificado por simulações computacionais, mostrando-se factível. São apontadas como principais características dessa abordagem a flexibilidade no planejamento de trajetórias – escolhendo-se, previamente, posições e velocidades no plano cartesiano – e o baixo esforço computacional – por não utilizar o modelo dinâmico do robô no processo de síntese. Os autores consideram-na com forte potencial para aplicações práticas.

Hasegawa *et al.* (2000) concebeu e implementou um gerador de marcha empregando algoritmos evolucionários. O principal propósito nesse artigo é planejar movimentos naturais e estáveis em várias condições de solo. Segundo os autores, movimento natural é aquele que minimiza a energia total consumida durante o processo de marcha. Assim a otimização de energia é levada em consideração, bem como o critério ZMP é avaliado para garantir estabilidade postural. A principal motivação dessa pesquisa está relacionada à grande dificuldade de se encontrar trajetórias ótimas para sistemas dinâmicos complexos, representados por equações diferencias não-lineares e fortemente acopladas. Foi destacado que a busca de trajetórias ótimas para robôs bípedes é uma questão de otimização numérica e de combinação, pois a marcha consiste de combinações de posturas intermediárias descritas por posições angulares. Assim, é proposto um algoritmo evolucionário hierárquico composto por duas camadas: uma que procura estabelecer posições angulares que estabeleçam as posturas intermediárias, e outra que visa combinar as posturas intermediárias para gerar a marcha. Basicamente, posturas candidatas são geradas por programação evolucionária (EP), interpolando posturas que permitem a marcha. Um algoritmo genético (GA) é empregado para avaliar e aplicar mutações a posições angulares, que descrevem posturas intermediárias, visando minimizar o consumo total de energia durante a marcha. O indivíduo (cromossomo) na camada EP – *gerador de configurações* – representa uma

configuração interpolada contendo valores numéricos do domínio $\Re$ para posições angulares (doze ângulos de junta, no caso do robô bípede utilizado, incluindo o movimento do tronco). O indivíduo na camada GA – *gerador de trajetórias* – encapsula duas informações: a configuração interpolada e um dígito binário (zero ou um) para sinalizar a validade (ou não) da configuração interpolada. O processo inicia-se pela camada EP, gerando indivíduos de forma aleatória e aplicando operadores de mutação e seleção segundo uma função de aptidão. Os indivíduos de EP são transferidos à camada GA e operadores de recombinação, mutação e seleção são empregados para resolver o problema de otimização de energia. Certas configurações interpoladas são transferidas à camada EP estabelecendo assim a sinergia entre as camadas. A princípio, a validação da estratégia proposta foi verificada em simulações computacionais para três casos hipotéticos: marcha em superfície plana, em superfície com inclinação positiva e, por fim, com inclinação negativa (nesses dois últimos casos, inclinações em torno de cinco graus). A verificação experimental da marcha em superfície plana foi realizada em um robô bípede experimental, com 24 kg e 1,20 m de altura, composto por duas pernas, a pelve e um tronco solidário à pelve. Doze juntas rotativas permitem ao robô mover-se no plano frontal e sagital. Um sistema de controle PD é empregado para rastrear as trajetórias de referência. Os bons resultados verificados em simulações foram confirmados no experimento, cujos dados são: passo de 0,30 m de comprimento a cada 5,0 s, potência total nos atuadores limitada entre 0 e 12 W, as posições angulares limitadas em $\pm 25^{\circ}$. Foi destacado, entretanto, que a demanda computacional é bastante elevada.

Em Chevallereau *et al.* (1999) é proposto um gerador ótimo de marcha restrita ao plano sagital para um robô bípede modelado como uma cadeia cinemática aberta e composto por seis elos interconectados por cinco juntas rotativas atuadas, quadril (2), tronco (1) e pernas (2). Cada perna é composta por dois elos, não possuindo pés e tornozelos. As trajetórias para os membros inferiores são avaliadas de forma independente à trajetória do tronco, que é definida levando-se em consideração a dinâmica do robô e as forças atuantes no centro de pressão (CoP). Durante a marcha, é considerada instantânea a transição entre as fases mono-apoiada e bi-apoiada do robô bípede. Para evitar o emprego da cinemática inversa, o algoritmo proposto trabalha no espaço das juntas. O movimento angular de cada junta é descrito por uma função polinomial de quarta-ordem, cujos coeficientes são utilizados como parâmetros do processo de otimização. A condição

de periodicidade da marcha é utilizada objetivando-se reduzir o número de parâmetros a otimizar e obter trajetórias angulares apenas para um ciclo da marcha, sendo os demais ciclos uma repetição do anterior. Foram considerados três critérios distintos para o processo de otimização: velocidade de movimento, torque mínimo (integral da norma do torque por unidade de deslocamento) e energia mínima (integral do valor absoluto do trabalho de forças externas por unidade de deslocamento). O *software Matlab*® foi empregando para resolver o conjunto de equações e restrições, permitindo computar as trajetórias ótimas. São fornecidas posições e velocidades iniciais bem como duração do movimento da perna em balanço. A comprovação do procedimento proposto foi realizada por simulações computacionais que utilizaram informações físicas de um protótipo em construção.

Em Choi *et al.* (1999) a geração de trajetórias ótimas é abordada por um ponto de vista diferente em relação aos trabalhados anteriores. A principal indagação pertinente a esse trabalho é: "Dado um conjunto de posturas espaciais que permitam realizar uma marcha, quais são as velocidades e acelerações em pontos-via[1], pertencentes às posturas espaciais, que garantam transições contínuas entre pontos-via adjacentes?" Neste contexto, é proposto um problema de otimização de parâmetros relativos a funções polinomiais de quinta ordem que visam interpolar pontos-via adjacentes no plano cartesiano, incluindo o ponto inicial e a meta. A solução proposta é baseada no emprego de algoritmos genéticos que procuram minimizar a soma da diferença de velocidades e acelerações computadas em diferentes instantes. A cinemática inversa é empregada para computar as trajetórias angulares relativas à solução ótima. De forma compacta, foi proposto um *simulador numérico* que objetiva planejar a marcha e computar posições, velocidades e acelerações angulares que garantam estabilidade postural satisfazendo o critério ZMP. O simulador é composto por um *gerador de trajetória,* baseado em algoritmos genéticos, denominado *treinador genético,* por um módulo que *resolve os modelos dinâmico e cinemático,* por um *planejador de trajetórias* para o sistema de estabilização postural e por um módulo de *verificação de estabilidade postural.* Outro aspecto interessante nesse trabalho diz respeito ao próprio robô bípede, acrônimo IWR-III, com 0,685 m de altura e 47 kg. Ele é constituído por seis elos que compõem os membros inferiores mais um quadril. A estabilização postural é realizada por intermédio de duas massas deslizantes montadas em um suporte solidário à pelve, tendo cada

[1] São denominados pontos-via um conjunto de pontos que unem as extremidades de uma trajetória.

uma a restrição de deslocar-se no plano frontal apenas em um percurso de pelo menos a metade do comprimento da pelve. As juntas rotativas são acionadas por oito servo-motores AC, acoplados por intermédio de redutores. O robô também possui um DSP TMS320C31 da *Texas Instruments*® responsável pelo sistema de controle do robô. Os resultados em simulações e experimentos demonstram que as trajetórias contínuas geradas permitiram rastrear com maior precisão a trajetória planejada para o ZMP. Destaque-se que, nesse trabalho, não há referências a um servo-controlador.

Dasgupta *et al.* (1999) analisa o sistema de locomoção humana e apresenta uma metodologia para adaptar as informações do modo de caminhar humano, capturadas e armazenadas, para acionar um robô bípede. Para a geração de informações relevantes ao sistema de locomoção humana, pontos de interesses são selecionados e marcados sobre os membros inferiores. A marcha é gravada sob diversos ângulos. As informações coletadas são processadas para obter as correspondentes coordenadas cartesianas, a partir das quais é possível computar as posições angulares, via cinemática inversa. O procedimento proposto para adaptar as informações coletadas é desenvolvido em duas partes: a primeira consiste em determinar a trajetória do ZMP, baseado nas informações armazenadas, e a segunda em computar um fator de correção para as posições angulares, tal que minimize a norma do erro quadrático entre as trajetórias do ZMP desejada e a real. O fator de correção é descrito por uma Série de *Fourier*, em que as amplitudes multiplicativas das funções senos e co-senos e os correspondentes valores de freqüência são os parâmetros de uma função de otimização. A validação da metodologia proposta foi realizada por simulação computacional, utilizando um modelo cinemático de um humanóide com quarenta graus de liberdade (um *software* comercial fornece suporte a essa fase). Para o caso analisado, demonstrou-se que a trajetória para o ZMP pode ser caracterizada por uma equação de reta.

2.3. Sistemas de Controle

Em *Li* (1992) foi especificada uma estratégia de controle que capacita o robô bípede a assumir o modo de andar dinâmico, assegurando a estabilidade postural durante todo o movimento. O modelo mecânico é composto, basicamente, por duas partes: membros inferiores (pernas constituídas por coxas (2), canelas (2) e pés (2) e membro superior (tronco). Os membros

inferiores têm por finalidade realizar a marcha, e o membro superior fornece estabilidade postural ao robô bípede. Fundamentalmente, o método de controle consiste em posicionar adequadamente o tronco. As seguintes etapas são necessárias: planejamento da marcha (de acordo com as restrições do ambiente: obstáculos, etc.), cálculo do movimento do tronco (tal que o ZMP seja obtido) e cálculo dos ângulos de juntas. Foi construído um experimento que permite monitorar os valores dos momentos quando o pé toca o solo, por intermédio de sensores. O movimento do tronco é computado em duas etapas. Durante a realização da marcha, a localização do tronco é computada por meio da transformada rápida de *Fourier* (algoritmo 1), baseado na marcha pré-estabelecida. Em seguida, é computada a diferença das posições real e pré-estabelecida para a marcha (algoritmo 2). O sinal de erro é utilizado para re-posicionar o tronco, empregando-se o algoritmo 1. Nessa mesma linha de pesquisa há várias referências entre as quais se destacam Yamaguchi et al. (1993) e Takanishi (1989).

Em Shibata *et al.* (2000) é descrito um procedimento que objetiva controlar robôs bípedes, reduzindo as forças de impacto provocadas pela interação do pé com o solo. São propostas duas leis de controle lineares e não acopladas que visam controlar a posição do quadril e a localização do centro de gravidade (CoG) das pernas. A força de impacto é reduzida, controlando-se a aceleração do CoG, por meio do ajuste de um parâmetro de projeto. A validação da abordagem proposta foi verificada por simulações computacionais e implementada em um robô bípede de oito graus de liberdade, pesando 28 kg e medindo 0,92 m de altura.

Enfatizando-se a dinâmica de contato produzida entre o pé e o solo, restrições são formuladas e um modelo dinâmico para o robô bípede é derivado (Goddard *et al.*, 1992). Com base no modelo obtido, foi projetado um sistema de controle retro alimentado cuja finalidade é controlar as forças de reação. A motivação desse trabalho é suportada por pesquisas sobre locomoção humana, cujos resultados relacionam o aumento de velocidade da marcha com o aumento da magnitude das forças de reação produzidas pelo contato do pé com o solo. Os autores apontam também que, na locomoção humana, as forças de reação são ativamente controladas, com magnitudes que podem chegar a 85% do peso do corpo, em marcha normal, e até 86% acima do peso do corpo, em marcha rápida. O sistema de controle linear projetado é baseado em uma versão aproximada do modelo dinâmico do robô bípede. Assim, levando-se em consideração as

restrições, são determinados o movimento nominal e as correspondentes trajetórias nominais para as posições angulares. Simulações computacionais, utilizando um modelo dinâmico de quatro graus de liberdade, foram realizadas para comprovação da abordagem proposta.

Em Takanashi (1989) é proposto um sistema de controle que permite a um robô bípede, consistindo de membros inferiores e um tronco, locomover-se sobre uma superfície livre de distúrbios (desníveis, depressões, etc.). A abordagem utilizada consiste de duas partes principais: na primeira, os movimentos dos membros inferiores e as coordenadas cartesianas do ZMP são planejados, um algoritmo computa os componentes cartesianos relativos ao tronco, descrevendo o movimento do mesmo. A segunda parte consta de um algoritmo de controle que realiza o modo de andar definido pelos deslocamentos dos membros inferiores e do tronco. Uma equação de movimento para o robô bípede é obtida aplicando-se o Princípio de D'Alembert. A partir da qual são isoladas as respectivas variáveis cartesianas do tronco, o que resulta em um sistema de duas equações diferenciais de 2ª ordem, acopladas e não-lineares. Simplificações são consideradas e um modelo não-linear e não acoplado é derivado para o movimento do tronco. É destacado que os membros inferiores e a trajetória do ZMP descrevem movimentos periódicos. Representações em Série de *Fourier* e utilizando-se a transformada rápida de *Fourier* (FFT) permitem computar uma solução aproximada periódica para o modelo simplificado. Mas o autor adverte que essa abordagem só apresenta bons resultados, medidos em termos do erro entre o ZMP desejado e o calculado, quando o tronco do robô bípede é longo ou quando apenas baixas velocidades de movimentos são consideradas. O autor generaliza a abordagem, ficando independe de velocidades e ou do comprimento do tronco. Para tal, foi proposto um algoritmo para obter uma solução periódica estrita do modelo não-linear. A convergência desse algoritmo é demonstrada em simulação computacional. A eficiência é definida em termos de números de interações necessárias à convergência, relatada como sendo em torno de oito. Não foi relatado o tempo computacional gasto.

Uma abordagem diferente (Cheng *et al.*,1997) pesquisou o desenvolvimento de sistemas de controle via algoritmos genéticos para otimizar o processo de caminhar do robô bípede. O projeto do controlador de trajetórias de juntas e o planejamento do modo de andar foram formulados como um problema de busca de parâmetros, e um algoritmo genético foi utilizado para fornecer

uma melhor solução para o problema. Projetos com diferentes critérios de otimização do processo de andar foram estudados, tais como, deslocar-se com maior velocidade, caminhar sobre superfícies inclinadas, e mover-se com um tamanho de passo pré-estabelecido. A validação dos resultados, obtida por simulação computacional e por comparação de resultados com outras pesquisas congêneres, demonstrou a capacidade dos algoritmos genéticos em encontrar soluções melhores que as utilizadas para a comparação.

Em Hasegawa *et al.* (2000) foi tratada a questão de gerar um movimento estável e natural ao robô bípede, independente da superfície de apoio. Um sistema de controle hierárquico baseado em algoritmos genéticos foi desenvolvido com a finalidade de otimizar o consumo de energia e proporcionar movimento estável e natural, levando-se em consideração a localização pré-definida do ZMP. A geração de trajetórias foi formulada como um problema de otimização de energia, com restrições. O emprego de algoritmos evolucionários hierárquicos, compostos por duas camadas: a primeira responsável por minimizar a energia consumida pelos atuadores (implementada com algoritmos genéticos) e a segunda, capaz de gerar a interpolação ótima de posições angulares (implementada com programação evolucionária), permitiu resolver o problema em questão. Entretanto, o aproveitamento dessa solução é prejudicado devido ao tempo computacional requerido na geração do movimento necessário ao processo de andar, como indicado pelos próprios pesquisadores.

3. Principais Características deste Trabalho

Este trabalho explora um robô bípede composto por uma sucessão de corpos rígidos interconectados por articulações rotativas. O robô bípede é constituído por dois subsistemas: os *membros inferiores* e o *tronco*. Onze elos interconectados em série por dez juntas rotativas formam os membros inferiores. Cada perna é composta por um *quadril* (com dois graus de liberdade, permitindo movimentos nos planos frontal e sagital), um *joelho* (com um grau de liberdade no plano sagital), um *tornozelo* (com dois graus de liberdade idênticos aos do quadril) e por um pé. Os quadris estão conectados à *pelve*. Solidário à pelve há um pêndulo invertido (*tronco*), provido de articulações que permitem movimentos pendulares tridimensionais. Totalizando, o robô bípede possui dez elos rígidos e doze graus de liberdade.

A modelagem é realizada para cada subsistema em separado, que são relacionados ao serem consideradas as forças reativas que ocorrem na junção do tronco à pelve devido aos movimentos independentes do tronco e dos membros inferiores. As forças de reação provocadas pelos movimentos dos membros inferiores são consideradas como perturbações ao movimento do tronco e vice-versa, conforme ilustra a Figura 1.2 (*computo das forças reativas*). O sistema de controle permite que as trajetórias de referência (computadas pelo *gerador de marcha*) sejam rastreadas, eliminando perturbações.

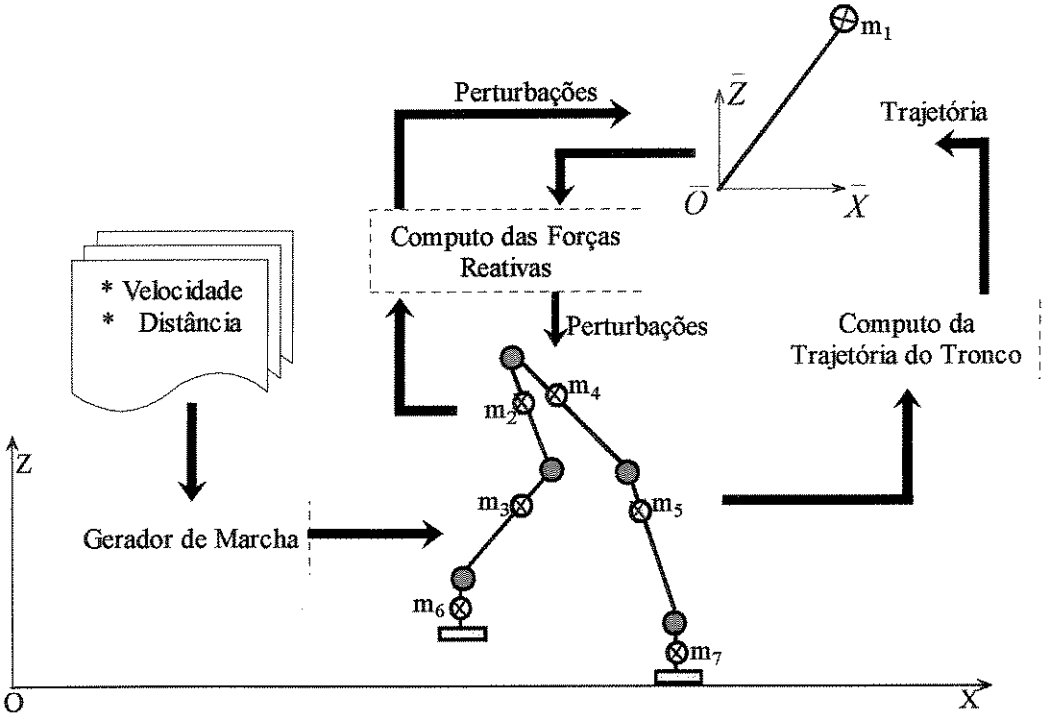


Figura 1.2 – Principais Característica do Sistema Integrado de Controle

Outro importante fator da abordagem desenvolvida reside na geração de trajetória para o tronco (*computo da trajetória do tronco*), computada durante o ato de caminhar, a partir do processando de informações de posições e acelerações impostas aos membros inferiores, que permite dotar o robô bípede de reflexos (provindos de sinais de sensores, por exemplo). Nossa concepção consiste de um *gerador de marcha* para fornecer as trajetórias de referências, utilizando a capacidade do robô bípede de posicionar os membros inferiores no espaço tridimensional, o que implica em movimentos dinamicamente estáveis sem a efetiva utilização do tronco. Se, por qualquer motivo, a marcha gerada não permitir estabilidade postural, o *gerador de trajetória para o tronco* entra em ação, deslocando o tronco para uma posição que restabeleça

uma marcha estável. Nesse contexto, foi desenvolvida uma solução diferente das demais apresentadas na literatura especializada, pois permite computar a trajetória para o tronco, considerando o critério ZMP para a estabilidade postural, durante a marcha e que emprega uma rede neural MLP.

Destaca-se, também, o projeto do sistema de controle adaptativo por modelo de referência. Redes neurais artificiais de base radial são empregadas e associadas ao projeto de um sistema de controle adaptativo e robusto. A estabilidade é avaliada segundo o critério de *Lyapunov*. O *sistema de controle* e o *gerador automático de trajetórias para o tronco* são integrados, compondo os mecanismos adaptativos propostos para a solução no modo de andar dinâmico.

Desenvolveu-se também um *modelador automático* que implementa no ambiente *Maple V*® o formalismo de *Newton-Euler* (Craig, 1995), fornecendo suporte computacional à obtenção de modelos dinâmicos literais.

Este trabalho está organizado na seguinte forma: o segundo capítulo descreve um projeto mecânico de um robô bípede, considerando o modo de andar dinâmico. É empregado o formalismo de *Denavit-Hartemberg* para descrever suas características cinemáticas e são derivadas a cinemática inversa e a modelagem dinâmica. A análise de estabilidade é formulada, considerando-se uma aproximação para o modelo não-linear. No terceiro capítulo, são apresentados alguns conceitos básicos sobre redes neurais (RN) e algoritmos genéticos (AG). É proposto um gerador automático de marcha, formulado como um problema de otimização e resolvido via AG. O quarto capítulo é dedicado ao projeto de um sistema de controle adaptativo, baseado em modelos de referência. A lei de controle é constituída por termos baseados no modelo dinâmico da planta, no modelo de referência e em incertezas. Redes neurais artificiais de base radial são empregadas para identificação *on-line* de alguns fenômenos físicos, permitindo assegurar estabilidade em malha fechada, no sentido de *Lyapunov*. O quinto capítulo é destinado às implementações computacionais do sistema de controle integrado. Simulações computacionais e análises são realizadas. Completando este trabalho, o sexto capítulo traz conclusões e perspectivas para trabalhos futuros.

Capítulo 2
Modelagem e Análise do Robô Bípede

Neste capítulo um robô bípede dotado de tronco é concebido. Considera-se o robô bípede composto por elos rígidos formando uma articulação mecânica de cadeia aberta. Assim, é empregado o formalismo de *Denavit-Hartemberg* para descrever suas características cinemáticas e são derivadas a cinemática inversa e a modelagem dinâmica. A análise de estabilidade é formulada considerando-se uma aproximação do modelo não-linear.

Em geral, um modelo dinâmico é descrito por n equações diferenciais não homogêneas, não-lineares e acopladas que descrevem a evolução temporal das variáveis generalizadas, que permite:

- Projetar mecanismos robóticos empregando simulações computacionais;
- Projetar e validar estratégias de controle;
- Empregar sistemas de controle baseados em modelos;
- Desenvolver um gerador de marcha para o robô bípede.

2.1. Robô Bípede

A concepção da articulação mecânica do robô bípede foi elaborada com a utilização do *software Solidworks*® (Predabon *et al.*, 2003), cuja filosofia básica de trabalho está vinculada à elaboração de um protótipo tridimensional subdivido em subsistemas conectados entre si por intermédio de restrições aos movimentos relativos. Considerando-se o protótipo ou os subsistemas, seus parâmetros físicos, massa, volume e momentos de inércia, são calculados automaticamente, a partir das características dos materiais a serem empregados, das formas e respectivas dimensões geométricas e dos sistemas de coordenadas previamente definidos.

A articulação mecânica que permite reproduzir marchas é constituída de onze elos interconectados por dez juntas rotativas, denominados "membros inferiores". Solidário ao quinto elo há um pêndulo invertido (tronco), contendo duas juntas perpendiculares entre si que permitem um movimento pendular tridimensional. Ao todo, o robô bípede possui doze juntas motoras[1]. Foi previsto que os motores estejam distantes das respectivas juntas e que os movimentos do eixo do motor sejam transferidos ao elo empregando-se correia dentada e engrenagens. Permitindo assim localizar o centro de massa na parte superior de cada elo.

A Figura 2.1 ilustra a idealização do protótipo mecânico em apresentação. O modelo está parametrizado: as distâncias entre centros das juntas estão representadas por a_j, para j = 0, 1,..., 12, sendo que a_{11} e a_{12} descrevem os comprimentos dos elos associados ao pêndulo invertido. Os rótulos *a* e *b* estão associados ao comprimento dos pés; assim como, os rótulos *c*, *d* e *e* descrevem as localizações dos motores alocados sobre os membros inferiores.

A Tabela 2.1 apresenta as medidas dos pés e das localizações dos motores, ilustradas na Figura 2.1.

Tabela 2.1 – Algumas medidas de comprimento, $\times 10^{-3}$ m.

a	b	c	d	e
182	93	100	50	100

[1] Juntas rotativas associadas a motores.

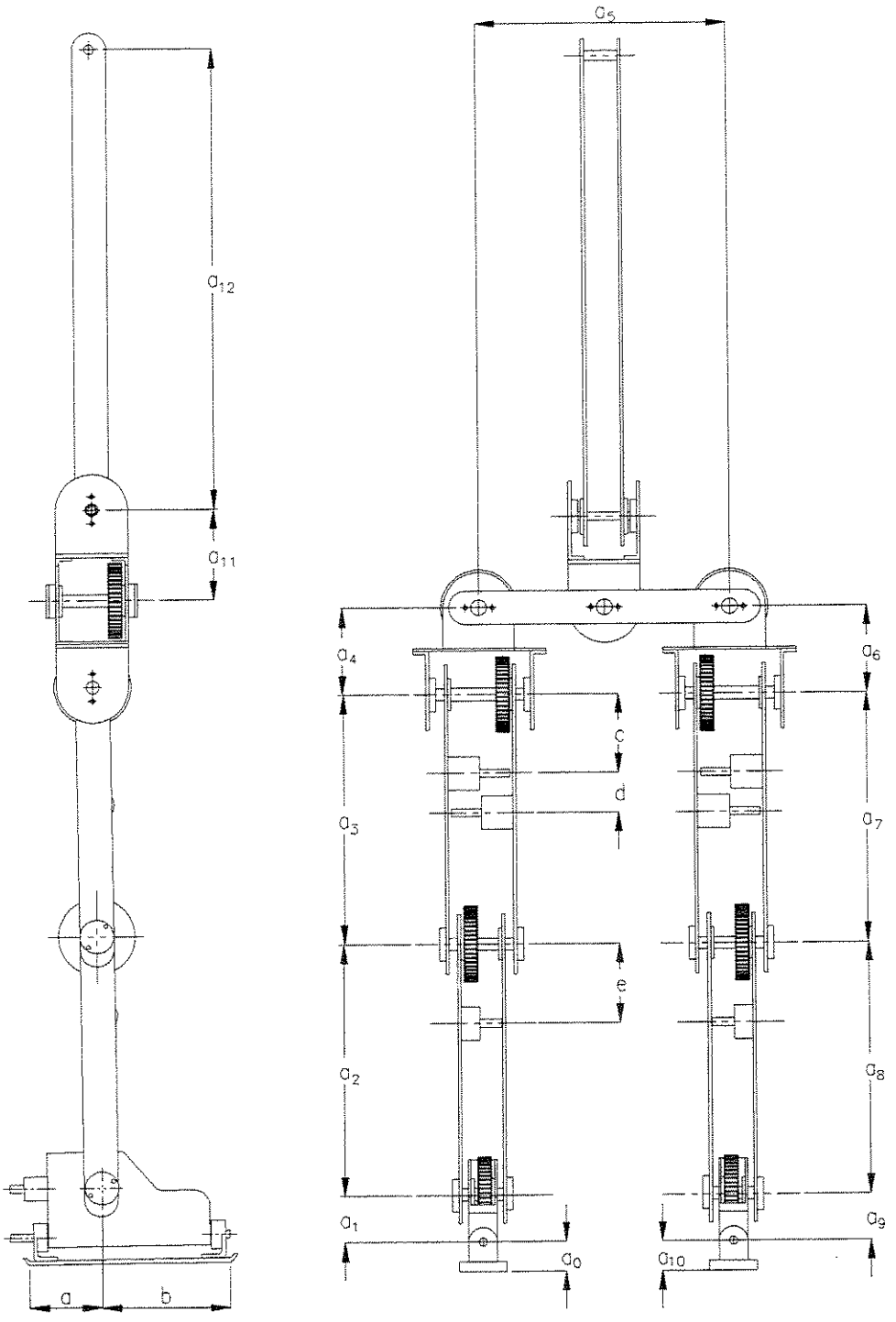


Figura 2.1 – Arquitetura mecânica do robô bípede.

A Tabela 2.2 informa os valores dos parâmetros físicos para o modelo proposto. O enésimo elo possui momento de inércia computado em relação ao enésimo sistema cartesiano, localizado no correspondente centro de massa e alinhado ao sistema cartesiano localizado à junta enésima mais um. Os momentos de inércia cruzados são nulos.

Tabela 2.2 – Parâmetros físicos do robô bípede.

	m	kg	Momentos de Inércia, kg m^2.			Centros de Massa, m.			%	SC
	$\times 10^{-3}$	$\times 10^{-2}$	$\times 10^{-3}$			$\times 10^{-2}$			$\times 10^{-2}$	
Elo_j	a_j	M	I_{11}	I_{22}	I_{33}	x_C	y_C	z_C	M/MT	
0	36	51	4	4	0	-2	0	-4	335	LD0
1	61	93	3	1	3	-1	0	0	610	LD1
2	316	88	1	9	9	-7	0	0	577	LD2
3	316	112	2	11	10	-9	0	-1	735	LD3
4	110	98	3	3	3	-3	0	-2	643	LD4
5	316	65	8	2	6	0	0	-1	427	TR
6	110	98	3	3	3	3	0	2	643	LE5
7	316	112	2	11	10	9	0	1	735	LE6
8	316	88	1	9	9	7	0	0	577	LE7
9	61	93	3	1	3	1	0	0	610	LE8
10	36	51	4	4	0	2	0	4	335	LE9
11	680	575	0	430	430	52	-2	-1	3773	TR
Massa Total, MT.		1524								

2.2. Modelo Cinemático do Robô Bípede

A cinemática permite estudar o movimento da estrutura mecânica, desconsiderando as forças generalizadas que o provocam, dadas as características geométricas dos elos e os tipos de juntas. Nesse contexto, aparecem duas situações distintas de interesse da robótica. Sendo conhecidos, em um robô de *n* graus de liberdade, as respectivas características geométricas dos elos e os tipos de juntas, a cinemática direta determina a posição e a orientação do órgão terminal, se as *n* variáveis generalizadas de juntas forem conhecidas. A cinemática inversa determina as *n* variáveis generalizadas de juntas, se a descrição espacial do órgão terminal em relação a um sistema de coordenadas inercial for conhecida. Em ambas as situações, o problema é de natureza não-linear. A Figura 2.2 ilustra essas situações.

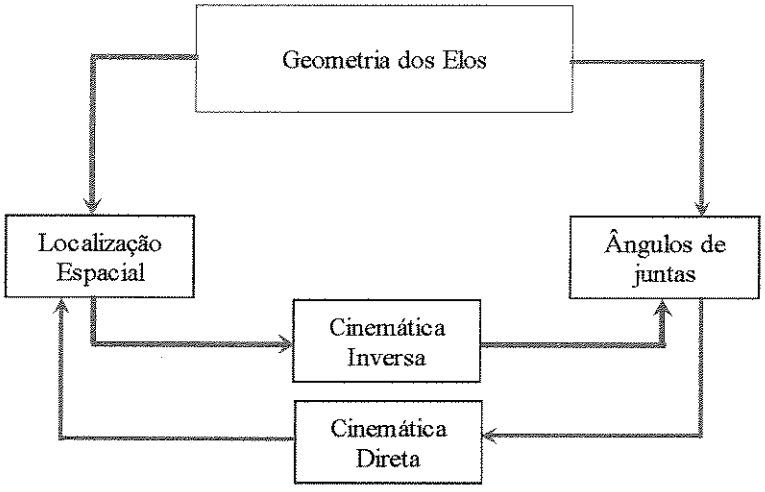


Figura 2.2 – Problema da cinemática de robôs.

O procedimento *Denavit-Hartemberg* (Schilling, 1990) sistematiza e generaliza a solução para a cinemática direta aplicada a robôs de cadeia aberta de *n* graus de liberdade. A cinemática inversa não possui solução única, sendo mais complexa e fortemente dependente da estrutura mecânica. As variáveis generalizadas podem ser obtidas por relações geométricas. Todavia, soluções analíticas nem sempre existem requerendo algoritmos iterativos (Takahashi *et al.*, 2000; Antonelli *et al.*, 2000). Apesar desses algoritmos serem de cunho geral, não há *a priori* garantias de convergência para a solução desejada, e o seu custo computacional é geralmente grande.

2.2.1. Procedimentos Sistemáticos *Denavit-Hartemberg*

Em um robô consistindo de elos rígidos unidos tanto por juntas de translação como de rotação, de forma seqüencial, o número de graus de liberdade é definido pelo número de pares de juntas mais elos, sendo que cada par define um grau de liberdade.

Os elos são enumerados na ordem crescente, partindo da base ao órgão terminal. O elo 0 (zero) é aquele que está solidário à base, onde é estabelecido um sistema de coordenadas inercial. O sistema de coordenadas do k-ésimo elo (designado por e_k) é afixado à junta k-ésima mais um, designada por j_{k+1}. O eixo z_k é paralelo ao eixo de movimento de j_{k+1}; os demais são ortogonais entre si. Logo abaixo, são apresentados os procedimentos sistemáticos que estabelecem as regras para impor os sistemas de coordenadas e para determinar os parâmetros geométricos, respectivamente.

PS. 2.1 – Determinação dos Sistemas de Coordenadas

a. Impor um sistema inercial de coordenadas à base do robô, que por simplificação é paralelo ao sistema de coordenadas $\{x_1, y_1, z_1\}$;

b. Impor um sistema de coordenadas $\{x_k, y_k, z_k\}$ a e_k afixando-o a j_{k+1}, cuja origem é localizada na intersecção da normal comum entre os eixos z_k e z_{k-1}. O eixo z_k está ao longo do eixo de movimento de j_{k+1};

c. Estabelecer o eixo x_k utilizando a Equação (2.1) ou ao longo da normal comum entre os eixos z_k e z_{k-1} se forem paralelos; no sentido de j_{k-1} para j_k;

$$x_k = \pm \frac{z_{k-1} \times z_k}{\left\| z_{k-1} \times z_k \right\|} \tag{2.1}$$

d. Estabelecer o eixo y_k utilizando a Equação (2.2), completando o sistema de coordenadas segundo a regra da mão direita:

$$y_k = \pm \frac{z_{k-1} \times x_k}{\left\| z_{k-1} \times x_k \right\|}; \tag{2.2}$$

e. Estabelecer um sistema de coordenadas ao órgão terminal, que por simplificação é idêntico ao último sistema.

PS. 2.2 – Parâmetros Cinemáticos

a. Determinar o ângulo α_k medido no sentido anti-horário entre os eixos z_{k-1} e z_k, em torno do eixo x_k;

b. Determinar a menor distância a_k obtida pela intersecção dos eixos z_{k-1} e z_k, medida ao longo do eixo x_k;

c. Determinar o ângulo θ_k medido no sentido anti-horário entre os eixos x_{k-1} e x_k, em torno do eixo z_{k-1}. Se a junta for de rotação, θ_k é uma variável de junta;

d. Determinar a menor distância d_k obtida pela intersecção dos eixos z_{k-1} com x_k, medida ao longo do eixo z_{k-1}. Se a junta for de translação, d_k é a variável de junta.

O conjunto de parâmetros $\{\alpha_k \quad a_k \quad \theta_k \quad d_k\}$ está associado a e_k, permitindo descrever univocamente os seus aspectos geométricos. Os dois primeiros parâmetros são constantes, determinados pela geometria do elo; os dois subseqüentes descrevem a posição relativa entre elos adjacentes.

A Figura 2.3 ilustra a disposição dos eixos de coordenadas para cada junta, de acordo com o procedimento **PS 2.1** (Determinação dos Sistemas de Coordenadas). Baseado na Figura 2.3, a Tabela 2.3 apresenta os parâmetros de *Denavit-Hartemberg*, de acordo com o procedimento **PS. 2.2** (Parâmetros Cinemáticos).

Tabela 2.3 – Parâmetros de Denavit-Hartemberg.

	Elos									
θ	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}
α	½ π	0	0	- ½ π	0	- ½ π	0	0	½ π	0
d	0	0	0	0	0	0	0	0	0	0
a	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}

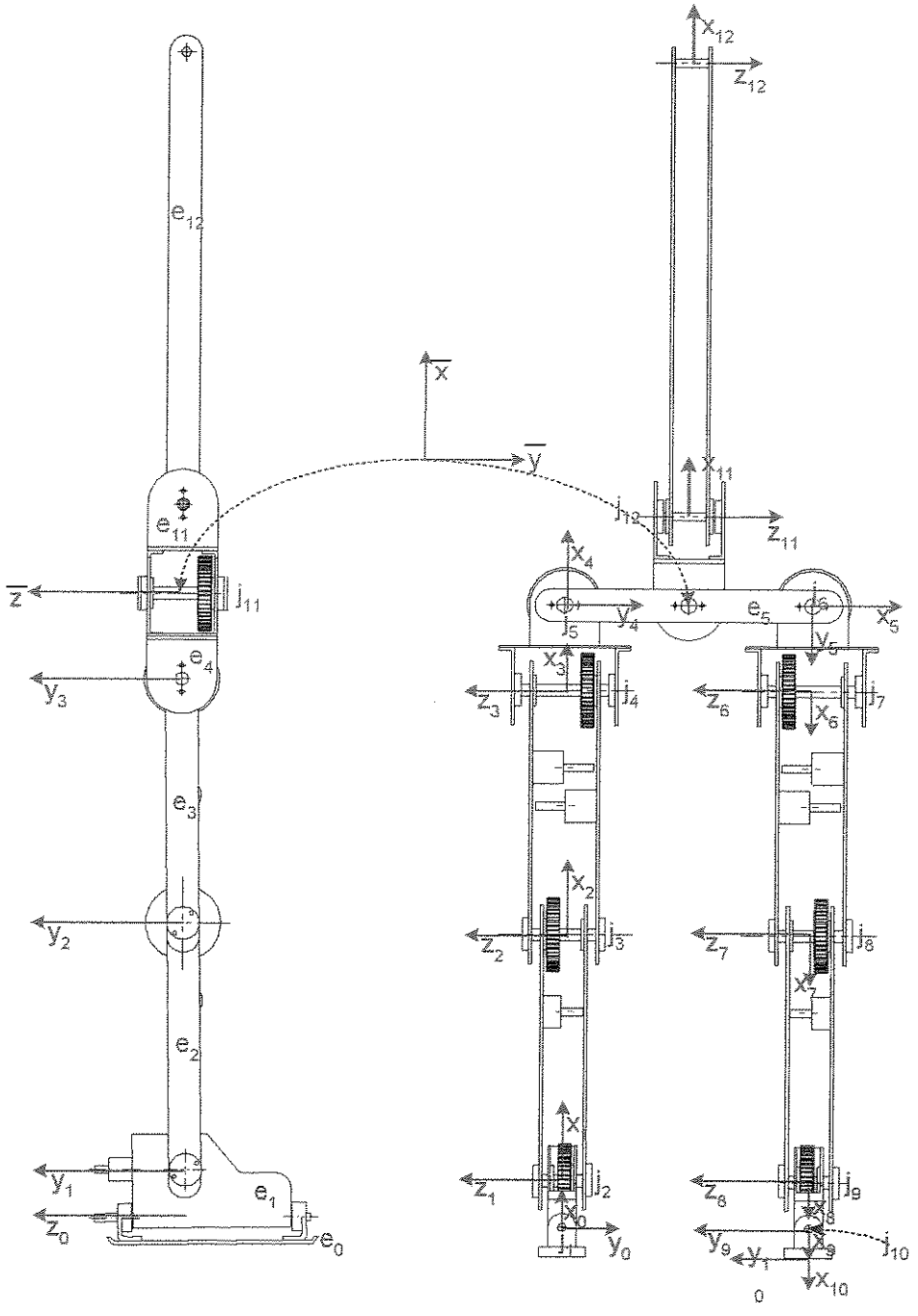


Figura 2.3 – Aplicação dos procedimentos de Denavit-Hartemberg (convenção: eixos coordenados em vermelho).

2.2.2. Cinemática Direta

Impondo-se os sistemas de coordenadas e determinando-se os parâmetros geométricos estabelecidos nos procedimentos sistemáticos **PS 2.1** e **PS 2.2**, a localização espacial do enésimo elo, relativa ao sistema de coordenadas *k*, é descrita em relação ao sistema de coordenadas *k-1* depois de executadas as operações de translações e rotações que seguem:

a. Rotação de θ_k graus em torno do eixo z_{k-1}, alinhando-se os eixos x_{k-1} e x_k;

b. Translação de d_k ao longo do eixo z_{k-1}, tornando os eixos x_{k-1} e x_k coincidentes;

c. Translação de a_k ao longo do eixo x_k, tornando concordantes as origens dos sistemas de coordenadas $_k$ e $_{k-1}$;

d. Rotação de α_k graus em torno do eixo x_k, alinhando-se os respectivos sistemas de coordenadas.

A Equação (2.3) representa as operações descritas acima, aplicando matrizes de transformações homogêneas.

$$ {}^{k-1}\mathbf{A}_k = \mathbf{H}_{\theta_k}\mathbf{H}_{d_k}\mathbf{H}_{\alpha_k}\mathbf{H}_{a_k} \equiv \begin{bmatrix} c_{\theta_k} & -c_{\alpha_k}s_{\theta_k} & s_{\alpha_k}s_{\theta_k} & a_k c_{\theta_k} \\ s_{\theta_k} & c_{\alpha_k}c_{\theta_k} & -s_{\alpha_k}c_{\theta_k} & a_k s_{\theta_k} \\ 0 & s_{\alpha_k} & c_{\alpha_k} & d_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3} $$

A cinemática direta é determinada pela Equação (2.4), que descreve as transformações sucessivas entre elos adjacentes partindo da base em direção ao órgão terminal.

$$ {}^{0}\mathbf{T}_n \equiv {}^{0}\mathbf{A}_1\,{}^{1}\mathbf{A}_2 \cdots {}^{n-1}\mathbf{A}_n = \prod_{k=1}^{n} {}^{k-1}\mathbf{A}_k \equiv \begin{bmatrix} {}^{0}\mathbf{R}_n & {}^{0}\mathbf{p}_n \\ 0 & 1 \end{bmatrix} \tag{2.4} $$

Assim, a localização espacial do órgão terminal em relação à base é completamente descrita pela matriz de rotação ${}^{0}\mathbf{R}_{n}$ e pelo vetor posição ${}^{0}\mathbf{p}_{n}$.

2.2.3. Cinemática Inversa

Considerando-se um robô de *n* graus de liberdade e conhecendo-se *a priori* a descrição espacial do órgão terminal e as características geométricas dos elos, as variáveis generalizadas de juntas necessárias para concretizar o movimento são obtidas por meio de cinemática inversa.

Em se tratando de robô bípedes (conforme a Figura 2.3), a descrição no espaço das juntas pode ser determinada se as localizações espaciais de SC4 e de SC10 forem conhecidas em relação ao sistema global (SG) O-XYZ. Nesse sentido, as hipóteses que seguem são consideradas.

$\mathbf{H}_{2.1}$ – É conhecida a descrição espacial de SC4 em relação ao SG, a todo instante durante o movimento;

$\mathbf{H}_{2.2}$ – É conhecida a descrição espacial de SC10 relativa ao SG, a todo instante durante o movimento;

$\mathbf{H}_{2.3}$ – O quinto elo (e_5) é mantido paralelo à superfície de apoio, durante todo o movimento.

Com base em $\mathbf{H}_{2.1}$, $\mathbf{H}_{2.2}$ e $\mathbf{H}_{2.3}$ são determinadas as variáveis associadas às juntas para o robô bípede, conforme Shih (1993) cujo método permite simplificar bastante a obtenção da cinemática inversa.

Sejam as Equações (2.5) e (2.6) que expressam as matrizes de rotações em torno dos eixos x e y, bem como a Equação (2.7), que define a nomenclatura utilizada nessa seção.

$$\mathbf{R}_x(\zeta)=\begin{bmatrix}1 & 0 & 0\\ 0 & c_\zeta & -s_\zeta\\ 0 & s_\zeta & c_\zeta\end{bmatrix} \tag{2.5}$$

$$\mathbf{R}_y(\zeta)=\begin{bmatrix}c_\zeta & 0 & s_\zeta\\ 0 & 1 & 0\\ -s_\zeta & 0 & c_\zeta\end{bmatrix} \tag{2.6}$$

$$\mathbf{R}_{\wp j}=\mathbf{R}_\wp(\zeta_j) \tag{2.7}$$

As Equações (2.8) e (2.9) são escritas considerando-se o sistema global de coordenadas (SG), utilizando-se as Equações (2.5) e (2.6) e a Definição (2.7).

$$\mathbf{R}_i=\mathbf{R}_o\mathbf{R}_{x1}\mathbf{R}_{y2}\mathbf{R}_{y3}\mathbf{R}_{y4}\mathbf{R}_{x5} \tag{2.8}$$

$$\mathbf{x}_i=\mathbf{x}_o+\mathbf{R}_o\begin{Bmatrix}0\\ w_i\\ 0\end{Bmatrix}-\mathbf{R}_o\mathbf{R}_{x1}\begin{Bmatrix}0\\ 0\\ a_4\end{Bmatrix}-\mathbf{R}_o\mathbf{R}_{x1}\mathbf{R}_{y2}\begin{Bmatrix}0\\ 0\\ a_3\end{Bmatrix}-$$

$$\mathbf{R}_o\mathbf{R}_{x1}\mathbf{R}_{y2}\mathbf{R}_{y3}\begin{Bmatrix}0\\ 0\\ a_2\end{Bmatrix}-\mathbf{R}_o\mathbf{R}_{x1}\mathbf{R}_{y2}\mathbf{R}_{y3}\mathbf{R}_{y4}\begin{Bmatrix}0\\ 0\\ a_1\end{Bmatrix}-\mathbf{R}_i\begin{Bmatrix}0\\ 0\\ a_o\end{Bmatrix} \tag{2.9}$$

Onde:

$\mathbf{R}_i$ Matriz de rotação do i-ésimo ponto

$\mathbf{x}_i$ Vetor posição do i-ésimo ponto

$\mathbf{R}_o$ Matriz de rotação do centro corpo

$\mathbf{x}_o$ Vetor posição do centro do corpo

$w_i=\begin{cases}-a_5/2 & ,i=1\\ a_5/2 & ,i=2\end{cases}$ Comprimento da metade do quinto elo

As Equações (2.10) e (2.11) descrevem as localizações espaciais de $SC3^i$ e $SC0^i$, respectivamente.

$$\mathbf{p}_3^i = \mathbf{x}_o + \mathbf{R}_o \begin{Bmatrix} 0 \\ w_i \\ 0 \end{Bmatrix} - \mathbf{R}_o \mathbf{R}_{x1} \begin{Bmatrix} 0 \\ 0 \\ a_4 \end{Bmatrix} \tag{2.10}$$

$$\mathbf{p}_0^i = \mathbf{x}_o + \mathbf{R}_o \begin{Bmatrix} 0 \\ w_i \\ 0 \end{Bmatrix} \tag{2.11}$$

A diferença vetorial entre as Equações (2.10) e (2.11) é computada conforme (2.12).

$$\mathbf{p}_3^i - \mathbf{p}_0^i = \mathbf{R}_o \mathbf{R}_{x1} \mathbf{R}_{y2} \begin{Bmatrix} 0 \\ 0 \\ a_3 \end{Bmatrix} + \mathbf{R}_o \mathbf{R}_{x1} \mathbf{R}_{y2} \mathbf{R}_{y3} \begin{Bmatrix} 0 \\ 0 \\ a_2 \end{Bmatrix} + \mathbf{R}_o \mathbf{R}_{x1} \mathbf{R}_{y2} \mathbf{R}_{y3} \mathbf{R}_{y4} \begin{Bmatrix} 0 \\ 0 \\ a_1 \end{Bmatrix}$$

$$\mathbf{p}_3^i - \mathbf{p}_0^i = \mathbf{R}_i \mathbf{R}_{x5}^{-1} \mathbf{R}_{y4}^{-1} \mathbf{R}_{y3}^{-1} \begin{Bmatrix} 0 \\ 0 \\ a_3 \end{Bmatrix} + \mathbf{R}_i \mathbf{R}_{x5}^{-1} \mathbf{R}_{y4}^{-1} \begin{Bmatrix} 0 \\ 0 \\ a_2 \end{Bmatrix} + \mathbf{R}_i \mathbf{R}_{x5}^{-1} \begin{Bmatrix} 0 \\ 0 \\ a_1 \end{Bmatrix} \tag{2.12}$$

A Equação (2.13) é obtida por multiplicação à esquerda e, por posterior, manipulação algébrica da Equação (2.12).

$$\mathbf{R}_{x5} \{v\} = \mathbf{R}_{y4}^{-1} \mathbf{R}_{y3}^{-1} \begin{Bmatrix} 0 \\ 0 \\ a_3 \end{Bmatrix} + \mathbf{R}_{y4}^{-1} \begin{Bmatrix} 0 \\ 0 \\ a_2 \end{Bmatrix} \tag{2.13}$$

Onde:

$$\{\mathbf{v}\} = \mathbf{R}_i^{-1}\left(\mathbf{p}_3^i - \mathbf{p}_0^i\right) - \begin{Bmatrix} 0 \\ 0 \\ a_1 \end{Bmatrix} \tag{2.14}$$

Assim, a variável relacionada à primeira junta é obtida pela manipulação algébrica do elemento pertencente à segunda linha de (2.13).

$$\theta_5^i = \mathrm{tg}^{-1}\left(\frac{v_2}{v_3}\right) \tag{2.15}$$

A Equação (2.16) descreve a localização espacial de SC1,

$$\mathbf{p}_1^i = \mathbf{x}_i + \mathbf{R}_i \begin{Bmatrix} 0 \\ 0 \\ a_o \end{Bmatrix} + \mathbf{R}_0\mathbf{R}_{x1}\mathbf{R}_{y2}\mathbf{R}_{y3}\mathbf{R}_{y4} \begin{Bmatrix} 0 \\ 0 \\ a_1 \end{Bmatrix} \tag{2.16}$$

Por um procedimento similar, é obtida a diferença vetorial entre as Equações (2.10) e (2.16). Em seguida são realizadas as operações de multiplicação à esquerda e de manipulação algébrica da equação resultante.

$$\mathbf{R}_{y4}\{\gamma\} = \mathbf{R}_{y3}^{-1} \begin{Bmatrix} 0 \\ 0 \\ a_3 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 0 \\ a_2 \end{Bmatrix} \tag{2.17}$$

Onde:

$$\{\gamma\} = \mathbf{R}_{x6}\mathbf{R}_i^{-1}\left(\mathbf{p}_5^i - \mathbf{p}_1^i\right) \tag{2.18}$$

A soma dos quadrados dos elementos pertencentes à primeira e à terceira linha de (2.17) permite obter a variável relacionada à terceira junta, conforme em (2.19).

$$\theta_3^i = \cos^{-1}\left(\frac{\gamma_1^2 + \gamma_3^2 - a_2^2 - a_3^2}{2a_2a_3}\right) \tag{2.19}$$

A variável da segunda junta pode ser obtida a partir de manipulações trigonométricas aplicadas à primeira linha da Equação (2.17), cujo resultado é expresso em (2.20).

$$\theta_4^i = \sin^{-1}\left(\frac{-a_3\sin(\theta_3^i)}{\sqrt{\gamma_1^2 + \gamma_3^2}}\right) - \mathrm{tg}^{-1}\left(\frac{\gamma_1}{\gamma_3}\right) \tag{2.20}$$

A variável relacionada à quinta e à quarta junta podem ser consideradas como as responsáveis pela orientação do quinto elo (e_5). Assim, utilizando-se a representação de orientação pelas operações[2] de rolagem (*roll*), balanço (*pitch*) e guinada (*yaw*) a Equação (2.7) pode ser reescrita. Para este problema, considera-se que a rotação em torno do eixo z é dado por $\phi = 0$ grau dado que o robô bípede proposto não possui esse grau de liberdade.

$$\mathbf{R}_{z\phi}\mathbf{R}_{y2}^{-1}\mathbf{R}_{x1}^{-1} = \mathbf{R}_{x5}\mathbf{R}_{y4}\mathbf{R}_{y3}\mathbf{R}_i^{-1}\mathbf{R}_o \equiv \begin{bmatrix}\mathbf{r} & \mathbf{s} & \mathbf{o}\end{bmatrix} \tag{2.21}$$

A solução do problema inverso é expresso pelas Equações (2.22) e (2.23).

$$\theta_2^i = -\mathrm{tg}^{-1}\left(\frac{-r_3}{\sqrt{r_1^2 + r_2^2}}\right) \tag{2.22}$$

$$\theta_1^i = -\mathrm{tg}^{-1}\left(\frac{s_3}{o_3}\right) \tag{2.23}$$

[2] Notação utilizada em dinâmica de corpos rígidos, em particular em dinâmica veicular.

As Equações (2.15), (2.19), (2.20), (2.22) e (2.23) descrevem a cinemática inversa para o robô bípede em questão. As variáveis, segundo o formalismo de *Denavit-Hartemberg* são:

$$\left[\theta_1 \quad \cdots \quad \theta_5 \quad \theta_6 \quad \cdots \quad \theta_{10}\right]^T = \left[\theta_5^1 \quad \theta_4^1 \quad \theta_3^1 \quad \theta_2^1 \quad \frac{\pi}{2}+\theta_1^1 \quad \frac{\pi}{2}-\theta_1^2 \quad \theta_2^2 \quad \theta_3^2 \quad \theta_4^2 \quad \theta_5^2\right]^T \tag{2.24}$$

De modo análogo, as variáveis para as juntas associadas ao pêndulo invertido (tronco) podem ser determinadas. A partir do sistema móvel de coordenadas, a extremidade do tronco pode ser expressa pelas seguintes expressões:

$$\mathbf{R}_i = \mathbf{R}_o \mathbf{R}_{x11} \mathbf{R}_{y12} \tag{2.25}$$

$$\mathbf{x}_i = \mathbf{R}_o \mathbf{R}_{x11} \begin{Bmatrix} 0 \\ 0 \\ a_{11} \end{Bmatrix} + \mathbf{R}_o \mathbf{R}_{x11} \mathbf{R}_{y12} \begin{Bmatrix} 0 \\ 0 \\ a_{12} \end{Bmatrix} \tag{2.26}$$

Onde:

$\mathbf{R}_i$ Matriz de rotação que descreve a orientação do centro de massa do tronco;

$\mathbf{x}_i$ Vetor posição do centro de massa do tronco;

$\mathbf{R}_o$ Matriz de rotação do centro de massa do corpo;

x_5 Distância do centro de massa do corpo ao SC11, medido na direção z.

Pré-estabelecidos o vetor $\mathbf{x}_i$ e a matriz de rotação $\mathbf{R}_o$, a Equação (2.26) pode ser manipulada algebricamente e por multiplicações à esquerda, resultando em:

$$\mathbf{R}_{X11}^{-1} \mathbf{R}_0^{-1} \left(\mathbf{x}_i - \mathbf{x}_0\right) = \begin{Bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{a_{11}} \end{Bmatrix} + \mathbf{R}_{X12} \begin{Bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{a_{12}} \end{Bmatrix} \equiv \mathbf{R}_{X12}^{-1} \{\chi\} \tag{2.27}$$

Da primeira e da segunda linha de (2.27) resultam as Equações (2.28) e (2.29), respectivamente:

$$\theta_{12} = \sin^{-1}\left(\frac{\chi_1}{a_{12}}\right) \tag{2.28}$$

$$\theta_{11} = -\mathrm{tg}^{-1}\left(\frac{\chi_2}{\chi_3}\right) \tag{2.29}$$

No capítulo terceiro é apresentado um formalismo baseado em algoritmos genéticos que permite obter o modelo cinemático para o robô bípede, fornecendo as informações necessárias para validar as hipóteses $\mathbf{H_{2.1}}$, $\mathbf{H_{2.2}}$ e $\mathbf{H_{2.3}}$ (que constam à pág. 24).

2.3. *Formulação das Equações de Movimento*

Historicamente, os formalismos de *Lagrange-Euler* (Schilling, 1990) e de *Newton-Euler* (Craig, 1995) são os algoritmos aplicados à modelagem de sistemas dinâmicos, permitindo tratar problemas de cunho geral e automatizar o processo de modelagem em computador. A formulação por *Lagrange-Euler* emprega o balanço das energias cinéticas e potenciais para formular as equações dinâmicas de sistemas, utilizando conceitos de forças e coordenadas generalizadas. O modelo matricial resultante é composto por termos que possuem interpretações físicas: matriz de inércia, vetor de carregamento das cargas gravitacionais, matriz de forças de *Coriolis* e centrífugas e o vetor de dissipação de energia.

A formulação por *Newton-Euler* é um algoritmo iterativo baseado no balanço de forças, analisando forças e momentos atuantes em cada elo. *A priori*, a trajetória no espaço das juntas é conhecida. Então, velocidade e aceleração experimentadas em cada elo são computadas recursivamente, partindo da base ao órgão terminal. Em seguida, a força e o momento provocados em cada elo são computados recursivamente, na direção inversa (partindo do órgão terminal à base). Neste trabalho preferiu-se empregar a formulação via *Newton-Euler*.

2.3.1. Equações de *Newton-Euler*

Considera-se que o robô é composto por elos rígidos, cujas localizações dos centros de massa (CoM, m) e os tensores de inércia ($\aleph$, kg m^2) são conhecidos, *a priori*. O tensor de inércia, descrito na forma matricial, é simétrico e permite computar a distribuição de massa do corpo rígido. Sejam ρ (kg/m^3) e υ (m^3) a massa específica e o volume do corpo rígido, respectivamente. O tensor de inércia relativo ao sistema de coordenadas do CoM, que está alinhado com o sistema de coordenadas do elo, é expresso pela Equação (2.30).

$$\aleph = \begin{bmatrix} \mathbf{I}_{11} & -\mathbf{I}_{12} & -\mathbf{I}_{13} \\ -\mathbf{I}_{12} & \mathbf{I}_{22} & -\mathbf{I}_{23} \\ -\mathbf{I}_{13} & -\mathbf{I}_{23} & \mathbf{I}_{33} \end{bmatrix} \tag{2.30}$$

Onde:

$$I_{11} \equiv \int_{v} \left(y^2 + z^2\right) \rho dv$$

$$I_{22} \equiv \int_{v} \left(x^2 + y^2\right) \rho dv$$

$$I_{33} \equiv \int_{v} \left(x^2 + y^2\right) \rho dv$$

$$I_{12} \equiv \int_{v} xy\rho dv$$

$$I_{13} \equiv \int_{v} xz\rho dv$$

$$I_{23} \equiv \int_{v} yz\rho dv$$

As forças (F, N) e momentos (η, Nm) requeridos para promover um dado movimento desejado são computados pelas equações de *Newton* e de *Euler*, conforme as Equações (2.31) e (2.32) respectivamente.

$$F = m\ddot{p}_c \tag{2.31}$$

$$\eta = \aleph\dot{\omega} + \omega \times \aleph\omega \tag{2.32}$$

Sendo:

$\mathbf{p}_c = \{x_c \quad y_c \quad z_c\}$ Localização do CoM;

$\varpi = \{\omega_x \quad \omega_y \quad \omega_z\}$ Velocidade angular.

Para computar as Equações (2.31) e (2.32) é necessário conhecer a trajetória, a velocidade e a aceleração medidas no centro de massa de cada elo. O procedimento é dividido em duas etapas. Na primeira são computadas recursivamente as velocidades e acelerações, partindo da base ao órgão terminal. Na seqüência, são computados as forças e os momentos provocados em cada elo, partindo do órgão terminal à base.

O robô bípede é a conjunção de elos sucessivos conectados por juntas revolutas, conforme ilustra a Figura 2.1. Seja ${}^{o}\varpi_k$ a velocidade angular do sistema de coordenadas $\{k\}$ em relação ao sistema inercial $\{o\}$. A propagação de ϖ_k é de natureza iterativa, definida pela soma vetorial de ϖ_{k-1} com a rotação de $\{k\}$, provocada pela rotação de j_{k+1}, conforme determina a Equação (2.33).

$${}^{o}\varpi_k = \hbar_1 {}^{o}\mathbf{T}_{k-1}{}^{k-1}\varpi_{k-1} + \varepsilon_k \hbar_1 {}^{o}\mathbf{T}_{k-1}\dot{\theta}_k i^3 \equiv {}^{o}\varpi_{k-1} + \varepsilon_k \hbar_1 {}^{o}\mathbf{T}_{k-1}\dot{\theta}_k i^3 \tag{2.33}$$

Onde:

$\hbar_\sigma = \frac{1}{\sigma}[\mathbf{I}_{3\times3} \quad 0_{1\times1}]$ (σ é o fator de escala);

i^e seleciona a enésima coluna da matriz;

ε_e igual a um, se a enésima junta for revoluta e zero em caso contrário;

θ_e variável da enésima junta;

${}^{0}\mathbf{T}_{k-1}$ Matriz de transformação homogênea entre {0} e {k-1}.

Sejam as seguintes definições (Craig, 1995):

$$\hbar_1 {}^{o}\dot{\mathbf{T}}_{k-1}{}^{k-1}\mathbf{T}_{o} \equiv \begin{bmatrix} 0 & -\Omega_z & \Omega_y \\ \Omega_z & 0 & -\Omega_x \\ -\Omega_y & \Omega_x & 0 \end{bmatrix} \tag{2.34}$$

$$\boldsymbol{\Omega} \equiv \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} = \boldsymbol{\varpi} \mathbf{i}^3 \tag{2.35}$$

A aceleração angular do sistema de coordenadas $\{k\}$ em relação ao sistema inercial $\{o\}$ é obtida como se segue.

$$\frac{d}{dt}\left({}^{o}\boldsymbol{\varpi}_k\right) = \frac{d}{dt}\left({}^{o}\boldsymbol{\varpi}_{k-1} + \varepsilon_k \, \hbar_1 {}^{o}\mathbf{T}_{k-1}\dot{\theta}_k \mathbf{i}^3\right)$$

$${}^{o}\dot{\boldsymbol{\varpi}}_k = {}^{o}\dot{\boldsymbol{\varpi}}_{k-1} + \varepsilon_k \left(\hbar_1 {}^{o}\dot{\mathbf{T}}_{k-1}{}^{k-1}\mathbf{T}_{o}{}^{o}\dot{\theta}_k + \hbar_1 {}^{o}\mathbf{T}_{k-1}\ddot{\theta}_k\right)\mathbf{i}^3$$

$${}^{o}\dot{\boldsymbol{\varpi}}_k = {}^{o}\dot{\boldsymbol{\varpi}}_{k-1} + \varepsilon_k \left({}^{o}\boldsymbol{\varpi}_{k-1} \times \hbar_1 {}^{o}\mathbf{T}_{k-1}\dot{\theta}_k \mathbf{i}^3 + \hbar_1 {}^{o}\mathbf{T}_{k-1}\ddot{\theta}_k \mathbf{i}^3\right) \tag{2.36}$$

Consideram-se ${}^{o}\mathbf{p}_k$ e ${}^{o}\mathbf{p}_{k-1}$ as localizações dos sistemas de coordenadas $\{k\}$ e $\{k-1\}$ em relação ao sistema inercial $\{o\}$, respectivamente. A descrição ${}^{o}\mathbf{p}_k$ pode ser obtida, a partir de ${}^{o}\mathbf{p}_{k-1}$, conforme define a Equação (2.37).

$$^{o}\mathbf{p}_{k} \equiv \hbar_{1}\,{}^{o}\mathbf{T}_{k-1}\mathbf{i}^{4} + \hbar_{1}\,{}^{o}\mathbf{T}_{k}\left({}^{k-1}\mathbf{T}_{k} - {}^{k-2}\mathbf{T}_{k-1}\right)\mathbf{i}^{4} = {}^{o}\mathbf{p}_{k-1} + \hbar_{1}\,{}^{o}\mathbf{T}_{k}\left(\mathbf{p}_{k} - \mathbf{p}_{k-1}\right) \tag{2.37}$$

Definindo-se: $^{k-1}\boldsymbol{\Delta}_{k}(t) \equiv \mathbf{p}_{k}(t) - \mathbf{p}_{k-1}(t)$, a derivada temporal de (2.37) fornece a velocidade tangencial de $\{k\}$ relativa a $\{o\}$, conforme representado a seguir.

$$^{o}\dot{\mathbf{p}}_{k} = {}^{o}\dot{\mathbf{p}}_{k-1} + \hbar_{1}\,{}^{o}\dot{\mathbf{T}}_{k}\,{}^{k-1}\boldsymbol{\Delta}_{k} + \hbar_{1}\,{}^{o}\mathbf{T}_{k}\,{}^{k-1}\dot{\boldsymbol{\Delta}}_{k}$$

$$^{o}\dot{\mathbf{p}}_{k} = {}^{o}\dot{\mathbf{p}}_{k-1} + \hbar_{1}\,{}^{o}\dot{\mathbf{T}}_{k}\,{}^{k}\mathbf{T}_{o}\,\boldsymbol{\Delta}_{o} + \hbar_{1}\,{}^{o}\mathbf{T}_{k}\,{}^{k-1}\dot{\boldsymbol{\Delta}}_{k}$$

$$^{o}\dot{\mathbf{p}}_{k} = {}^{o}\dot{\mathbf{p}}_{k-1} + {}^{o}\boldsymbol{\Omega}_{k} \times \hbar_{1}\,{}^{o}\mathbf{T}_{k}\,{}^{k-1}\boldsymbol{\Delta}_{k} + \hbar_{1}\,{}^{o}\mathbf{T}_{k}\,{}^{k-1}\dot{\boldsymbol{\Delta}}_{k}$$

$$^{o}\dot{\mathbf{p}}_{k} = {}^{o}\dot{\mathbf{p}}_{k-1} + {}^{o}\boldsymbol{\varpi}_{k} \times \hbar_{1}\,{}^{o}\mathbf{T}_{k-1}\,{}^{k-1}\boldsymbol{\Delta}_{k} + (1-\varepsilon_{k})\hbar_{1}\,{}^{o}\mathbf{T}_{k-1}\,{}^{k-1}\dot{\boldsymbol{\Delta}}_{k} \tag{2.38}$$

A aceleração tangencial de $\{k\}$ relativa a $\{o\}$ é obtida derivando-se a Equação (2.38), cujo resultado é descrito pela Equação (2.39).

$$\begin{aligned} ^{o}\ddot{\mathbf{p}}_{k} = {}^{o}\ddot{\mathbf{p}}_{k-1} &+ {}^{o}\dot{\boldsymbol{\varpi}}_{k} \times \hbar_{1}\,{}^{o}\mathbf{T}_{k-1}\,{}^{k-1}\boldsymbol{\Delta}_{k} + {}^{o}\boldsymbol{\varpi}_{k} \times \hbar_{1}\,{}^{o}\mathbf{T}_{k-1}\,{}^{k-1}\boldsymbol{\Delta}_{k} + \\ &(1-\varepsilon_{k})\left[\hbar_{1}\,{}^{o}\mathbf{T}_{k-1}\,{}^{k-1}\ddot{\boldsymbol{\Delta}}_{k} + 2\,{}^{o}\boldsymbol{\varpi}_{k} \times \hbar_{1}\,{}^{o}\mathbf{T}_{k-1}\,{}^{k-1}\dot{\boldsymbol{\Delta}}_{k}\right] \end{aligned} \tag{2.39}$$

As Equações (2.33), (2.36), (2.38) e (2.39) constituem a primeira etapa do algoritmo. Adicionalmente, são providas as seguintes condições iniciais:

$$\boldsymbol{\varpi}_{0} = \{0\} \tag{2.40}$$

$$\dot{\boldsymbol{\varpi}}_{0} = \{0\} \tag{2.41}$$

$$^{o}\dot{\mathbf{p}}_{0} = \{0\} \tag{2.42}$$

$$^{\circ}\ddot{\mathbf{p}}_0 = -\{\mathbf{g}\} \tag{2.43}$$

Tal que a velocidade e a aceleração angulares da base são consideradas nulas, dadas pelas Equações (2.40) e (2.41), respectivamente. A velocidade tangencial da base é nula (Eq. 2.42) e o carregamento gravitacional é incluso no vetor aceleração da base (Eq. 2.43) que será propagado para os demais elos, pelo algoritmo a seguir.

Partindo do órgão terminal à base, é possível computar recursivamente as forças e momentos atuantes em cada elo, utilizando-se as Equações (2.33), (2.36), (2.38) e (2.39).

A Equação (2.44) permite computar recursivamente a força externa atuante sobre o e_k, descrito em $\{\mathrm{k}\}$, devido a e_{k-1}.

$$\mathbf{f}_k = \mathbf{f}_{k+1} + \mathbf{m}_k \left\{ {}^{\circ}\ddot{\mathbf{p}}_k + {}^{\circ}\dot{\boldsymbol{\varpi}}_k \times \left({}^{\circ}\mathbf{c}_k - {}^{\circ}\mathbf{p}_k \right) + {}^{\circ}\boldsymbol{\varpi}_k \times {}^{\circ} \left[\boldsymbol{\varpi}_k \times \left({}^{\circ}\mathbf{c}_k - {}^{\circ}\mathbf{p}_k \right) \right] \right\} \tag{2.44}$$

Onde:

m_k massa de e_k;

$^{\circ}\mathbf{c}_k$ localização relativa a $\{\mathrm{o}\}$ do centro de massa de e_k

O momento externo aplicado a e_k, descrito em $\{\mathrm{k}\}$, devido a e_{k-1} é obtido recursivamente, de acordo com a Equação (2.45).

$$\begin{aligned} \boldsymbol{\eta}_k = \boldsymbol{\eta}_{k+1} + \hbar_1 \left({}^{\circ}\mathbf{T}_k \mathbf{c}_k - {}^{\circ}\mathbf{T}_{k-1} \mathbf{i}^4 \right) \times \mathbf{f}_k - \hbar_1 \left({}^{\circ}\mathbf{T}_k \mathbf{c}_k - {}^{\circ}\mathbf{T}_k \mathbf{i}^4 \right) \times \mathbf{f}_{k-1} + \\ \aleph_k \, {}^{\circ}\dot{\boldsymbol{\varpi}}_k + {}^{\circ}\boldsymbol{\varpi}_k \times \left(\aleph_k \, {}^{\circ}\boldsymbol{\varpi}_k \right) \end{aligned} \tag{2.45}$$

Para computar as Equações (2.44) e (2.45) consideram-se as forças e momentos provocados pelo contato do órgão terminal com a superfície de apoio, gerando-se as condições iniciais:

$$f_{n+1} = -f_a \tag{2.46}$$

$$\eta_{n+1} = -\eta_a \tag{2.47}$$

As Equações (2.44) e (2.45) são utilizadas para obter o torque generalizado necessário para a realização do movimento, conforme (2.48).

$$\tau_k = \varepsilon_k \eta_k^T \hbar_1^{k-1} \mathbf{T}_o i^3 + (1-\varepsilon_k) \mathbf{f}_k^T \hbar_1^{k-1} \mathbf{T}_o i^3 + \mathbf{b}_k(\mathbf{q}_k, \dot{\mathbf{q}}_k) \tag{2.48}$$

Escrevendo a Equação (2.48) na forma de espaço de estados, considerando um vetor de variáveis generalizadas q, resulta a Equação (2.49) que fornece um conjunto de n equações diferenciais, expressa na forma matricial.

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{B}(\mathbf{q},\dot{\mathbf{q}}) = \tau \tag{2.49}$$

Onde:

$\mathbf{M}(\mathbf{q})$	Matriz de Inércia;
$\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$	Matriz de forças de *Coriolis* e centrifugas;
$\mathbf{G}(\mathbf{q})$	Vetor de carregamento gravitacional;
$\mathbf{B}(\mathbf{q},\dot{\mathbf{q}})$	Vetor de acoplamento dissipativos;
τ	Vetor de torque externo.

O *software Maple® V R4* (Geddes, et. al., 1997) foi utilizado para implementar o formalismo de *Newton-Euler*, automatizando o processo de modelagem. No **Anexo A** são apresentadas as principais características do referido *software*, a implementação computacional da Equação (2.48) e as discussões das principais partes do programa desenvolvido, nomeado *NEROBOT*. A modelagem simbólica de alguns sistemas robóticos, objetivando validar e exemplificar a utilização do programa *NEROBOT*, é apresentada no **Anexo B**.

2.3.2. Modelagem Dinâmica do Robô Bípede

A dinâmica do robô bípede foi obtida subdividindo-se a estrutura mecânica em dois subsistemas: os *membros inferiores* e o *pêndulo invertido* (tronco). O quinto elo (e_5) conecta esses subsistemas. A interação ocorre pelas forças generalizadas de reação no vínculo de e_5 com o décimo primeiro elo (e_{11}), ao qual está acoplado o pêndulo invertido (e_{12}). Nesse contexto, à medida que a marcha se realiza, e_5 é acelerado na direção e sentido do movimento. As forças generalizadas de reação no vínculo são geradas pelo movimento de e_{12}, quando excitado pelo movimento de e_5. Assim, admite-se que a dinâmica do pêndulo invertido montado sobre um carro represente as características principais de acoplamento entre os subsistemas *membros inferiores* e *tronco* (Drummond *et al.*, 1999).

Considere-se por primeiro a modelagem do tronco. Para a adequação do modelo físico proposto aos procedimentos sistemáticos apresentados na Sub-capítulo (2.3), a Figura 2.4 ilustra uma junta virtual de translação e um elo virtual ao qual está acoplado um sistema inercial.

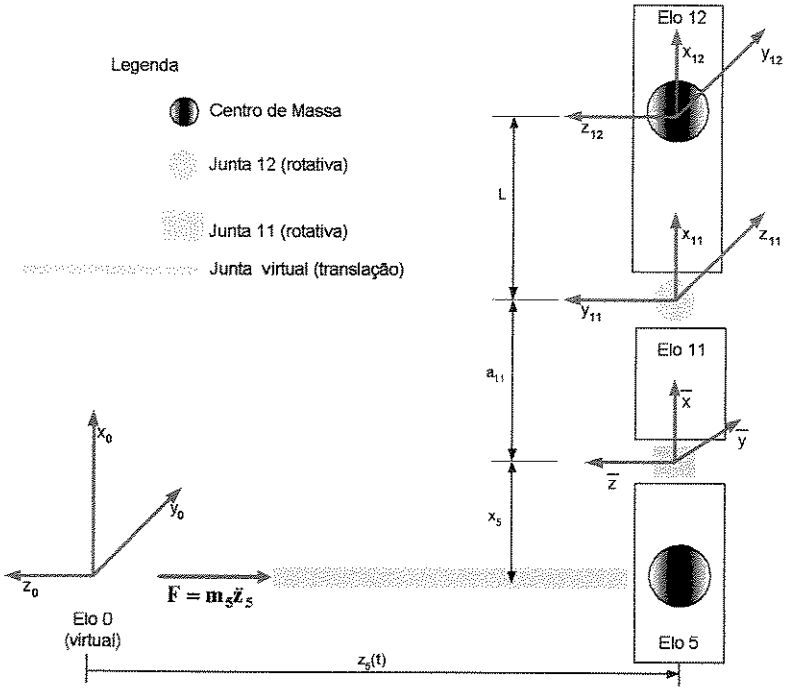


Figura 2.4 – Modelo do Pendulo Invertido sobre um Carro.

As Tabelas 2.4 e 2.5 fornecem os valores para os parâmetros geométricos e os parâmetros físicos para este modelo, respectivamente. As variáveis generalizadas de junta são: $z_5(t)$, $\theta_{11}(t)$ e $\theta_{12}(t)$.

Tabela 2.4 – Parâmetros geométricos relativos à Figura 2.8.

Elo	θ	α	a	d	Variáveis de Junta
1	0	0	0	q_1	$z_5(t)$
2	q_2	$-\pi/2$	a_{11}	0	$\theta_{11}(t)$
3	q_3	0	a_{12}	0	$\theta_{12}(t)$

Tabela 2.5 – Parâmetros físicos relativos à Figura 2.2.

Elos	Massa	I_{xx}	I_{yy}	I_{zz}	I_{xy}	I_{xz}	I_{yz}	x_c	y_c	z_c
5	m_5	0	0	0	0	0	0	0	0	0
11	m_{11}	0	0	0	0	0	0	0	0	0
12	m_{12}	0	Iy	Iz	0	0	0	L	0	0

Às informações contidas nas Tabelas 2.4 e 2.5 são adicionadas as informações que seguem:

N_Elos := 3	número de elos;
GRAV := matrix(3,1,[0,0,-g[o]])	vetor gravitacional;
TJ := matrix(N_Elos,1,1)	vetor identificador de juntas revolutas;
Ftoos := matrix(3,1,0)	forças de contato do órgão terminal com o meio;
Ttoos := matrix(3,1,0)	momentos de contato do órgão terminal e o meio.

A partir destes dados de entrada foi obtido o modelo dinâmico mostrado em (2.50), com a utilização do programa *NEROBOT*.

$$\begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{12} & M_{22} & M_{23} \\ M_{13} & M_{23} & M_{33} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} + \begin{bmatrix} c_1\dot{q}_3^2 \\ c_2\dot{q}_2\dot{q}_3 \\ c_3\dot{q}_2^2 \end{bmatrix} + \begin{bmatrix} 0 \\ h_2(q_2,q_3) \\ h_3(q_2,q_3) \end{bmatrix} + \begin{bmatrix} b_1(\dot{q}_1) \\ b_2(\dot{q}_2) \\ b_3(\dot{q}_3) \end{bmatrix} = \begin{bmatrix} 0 \\ \tau_2 \\ \tau_3 \end{bmatrix} \tag{2.50}$$

Onde:

$$M_{11} = m_5 + m_{11} + m_{12} \qquad M_{12} = 0 \qquad M_{13} = -m_{12}(a_{12} + L)\cos(q_3)$$

$$M_{22} = m_{12}\left\{\frac{1}{2}\left[\cos(2q_3)\left(a_{12}^2 + La_{12} + L^2 + I_y\right) + a_{12}^2 + L^2 + I_y\right] + 2a_{11}\cos(q_3)(L + a_{12}) + La_{12}\right\}$$

$$+ a_{11}^2(m_{11} + m_{12})$$

$$M_{23} = 0$$

$$M_{33} = m_{12}\left\{a_{12}(2L + a_{12}) + L^2\right\} + I_z$$

$$c_1 = m_{12}\cos(q_3)(a_{12} + L)\dot{q}_3^2$$

$$c_2 = -m_{12}\left\{\sin(2q_3)\left(L^2 + a_{12}^2 + 2La_{12} + I_y\right) + 2a_{11}\sin(q_3)(L + a_{12})\right\}\dot{q}_2\dot{q}_3$$

$$c_3 = m_{12}\left[\sin s(2q_3)\left(La_{12} + \frac{I_y}{2} + \frac{a_{12}^2}{2} + \frac{L^2}{2}\right) + \sin(q_3)a_{11}(L + +a_{12})\right]\dot{q}_3^2$$

$$h_2 = -\left\{a_{11}\sin(q_2)(m_{11} + m_{12}) + m_{12}\cos(q_2)\cos(q_3)(a_{12} + L)\right\}g_o$$

$$h_3 = -m_{12}\sin(q_2)\sin(q_3)(a_{12} + L)g_o$$

A primeira linha de (2.50) expressa a equação de movimento de e_5 em relação ao sistema inercial e as demais expressam os movimentos de j_{11} e j_{12} relativas aos graus de liberdade do

pêndulo invertido. Reescrevendo-as, as variáveis do problema tornam-se, cuja implementação computacional está definida no **Anexo C.**

$$M_{11}\,{}^{0}\ddot{z}_5 + b_1\left({}^{0}\dot{z}_5\right) = -M_{13}\ddot{\theta}_{12} - c_1\left(\theta_{11},\dot{\theta}_{12}\right) \tag{2.51}$$

$$\begin{bmatrix} M_{22} & 0 \\ 0 & M_{33} \end{bmatrix}\begin{Bmatrix} \ddot{\theta}_{11} \\ \ddot{\theta}_{12} \end{Bmatrix} + \begin{Bmatrix} c_2\left(\dot{\theta}_{11},\dot{\theta}_{12}\right) \\ c_3\left(\dot{\theta}_{11},\dot{\theta}_{12}\right) \end{Bmatrix} + \begin{Bmatrix} h_2\left(\theta_{11},\theta_{12}\right) \\ h_3\left(\theta_{11},\theta_{12}\right) \end{Bmatrix} + \begin{Bmatrix} b_2\left(\dot{\theta}_{11}\right) \\ b_3\left(\dot{\theta}_{12}\right) \end{Bmatrix} = \begin{Bmatrix} \tau_2 \\ \tau_3 \end{Bmatrix} - \begin{Bmatrix} 0 \\ d_{11}\,{}^{o}\ddot{z}_5 \end{Bmatrix} \tag{2.52}$$

Considerando o robô bípede, e_5 faz a conexão entre o pêndulo invertido e os membros inferiores. Assim, os termos do lado esquerdo de (2.51) é parte do modelo dinâmico referente aos membros inferiores do robô bípede; os termos do lado direito são as forças generalizadas de vínculo e podem ser consideradas como perturbações ao movimento dos membros inferiores do robô bípede. De forma análoga, o segundo termo do lado direito de (2.52) é considerado como perturbação ao movimento do pêndulo invertido. Desse modo, essas expressões mostram as influências que os movimentos dos membros inferiores causam ao movimento do pêndulo invertido e vice-versa. Como o pêndulo invertido é utilizado para fornecer estabilidade ao modo de andar, tais influências (perturbações) devem ser controladas.

De maneira análoga à modelagem do tronco, a modelagem para os membros inferiores foi realizada empregando-se o programa *NEROBOT*, utilizando-se os parâmetros descritos nas Tabelas 2.1 e 2.2, e as seguintes informações adicionais:

N_Elos := 10	número de elos;
GRAV := matrix(3,1,[0,0,-g[o]])	vetor gravitacional;
TJ := matrix(N_Elos,1,1)	vetor identificador de juntas revolutas;
Ftoos := $\{f_x,f_y,f_z\}$	forças de contato do órgão terminal com o meio;
Ttoos := $\{T_x,T_y,T_z\}$	momentos no contato do órgão terminal e o meio.

Em virtude da complexidade e da grande extensão do modelo obtido, ele é apresentado na forma literal, o **Anexo D** mostra sua implementação computacional. Para as simulações foram considerados nulos os vetores χ e η.

$$\mathbf{M}(\theta)\ddot{\theta} + \mathbf{C}(\dot{\theta})\dot{\theta} + \mathbf{G}(\theta) + \mathbf{B}(\dot{\theta}) = \tau - \chi - \eta - \Delta \tag{2.53}$$

Onde:

χ Vetor devido às forças externas aplicadas ao órgão terminal;

η Vetor devido aos momentos externos aplicados ao órgão terminal;

Δ Vetor de perturbações (forças de vínculo entre e_5 e o tronco).

Capítulo 3

Geradores Automáticos de Marcha e de Trajetória para o Tronco

O robô bípede pode ser classificado de acordo com a marcha a realizar. Durante o processo de andar há uma troca periódica entre as fases bi-apoiada e mono-apoiada. A fase bi-apoiada é caracterizada quando ambos os pés estão em contato com a superfície de apoio (solo). Na fase mono-apoiada apenas um dos pés serve de apoio. Neste trabalho, considera-se uma mudança instantânea de uma fase à outra.

A estabilidade postural é avaliada durante a fase mono-apoiada. A interação entre o pé de apoio e o solo é descrita em termos de forças generalizadas distribuídas sobre a superfície de contato (polígono suporte), cujas resultantes estão aplicadas ao centro de pressão[1] (CoP) cuja localização é função da dinâmica do sistema (Goswami, 1999). O robô bípede realiza a marcha estática se a projeção de seu *centro de massa* estiver contido no polígono suporte. Do contrário, o robô bípede realiza marcha dinâmica.

O **g**erador **a**utomático de **m**archa (*GAM*) aqui desenvolvido permite variar a andadura alterando-se os parâmetros de entrada: *velocidade de trajeto* (v), *comprimento do passo* (λ), *máxima altura vertical do pé em balanço* (d) e *ângulo de inclinação do pé em relação ao solo* (q_o). A partir dessas informações, foi proposto um problema de otimização, baseado em algoritmo genético, que busca determinar um conjunto de pontos-via que permita unir posições

[1] Centro de pressão é o ponto onde agem as resultantes das forças dinâmicas.

básicas previamente definidas. Para suavizar a trajetória entre pontos-via é empregado um conjunto de *Splines* de quinta-ordem. Transições suaves são garantidas se velocidades e acelerações nos pontos-via são contínuas entre trechos subseqüentes. Assim, foi proposto um segundo problema de otimização, também resolvido por algoritmo genético, que visa determinar as velocidades e as acelerações associadas aos pontos-via para garantir suavidade ao movimento. Assim, implementamos um gerador automático de marcha utilizando algoritmos genéticos hierárquicos para resolver dois problemas distintos de otimização. O *GAM* trabalha no espaço cartesiano do robô bípede e, após aplicar cinemática inversa, fornece trajetórias angulares de referência.

O gerador automático de trajetórias para o tronco (*GAT*) emprega uma rede neural recorrente do tipo MLP que, a partir da marcha previamente planejada e da trajetória do tronco em instantes anteriores, atualiza a trajetória para o tronco para garantir o balanço dinâmico (estabilidade postural) por análise do ZMP (*Zero Moment Point*).

Antes de tratar da implementação do *GAM* e do *GAT*, são introduzidos alguns conceitos sobre algoritmos genéticos e redes neurais artificiais, utilizados neste trabalho.

3.1. Redes Neurais e Técnicas Evolucionárias

Uma nova geração de robôs multifuncionais está surgindo, visando auxiliar o ser humano em suas diversas tarefas cotidianas. Andróides deixam o mundo da ficção e começam a fazer parte do nosso cotidiano. A sofisticação dessa nova geração de máquinas está apoiada no extensivo emprego de técnicas evolucionárias: *algoritmos genéticos* e *programação evolutiva*, como exemplos. Emprega-se evolução artificial a sistemas robóticos para otimizar controladores neurais artificiais ou parâmetros de estruturas de controle pré-definidas, para permitir ao robô circular em ambientes desestruturados, por exemplo.

3.1.1. Redes Neurais Artificiais

Redes neurais artificiais – ou apenas redes neurais – visam sintetizar o modo cerebral de processar informações, de maneira bastante simplificada, implementando apenas suas características essenciais, mantendo-se, todavia, o poder do processamento não-linear, paralelo e distribuído. As redes neurais são formadas por coleções de unidades básicas de processamentos (neurônios) interconectados por elos ponderados (sinapses), objetivando propagar informações entre neurônios. A Figura 3.1 ilustra as partes principais de um neurônio artificial.

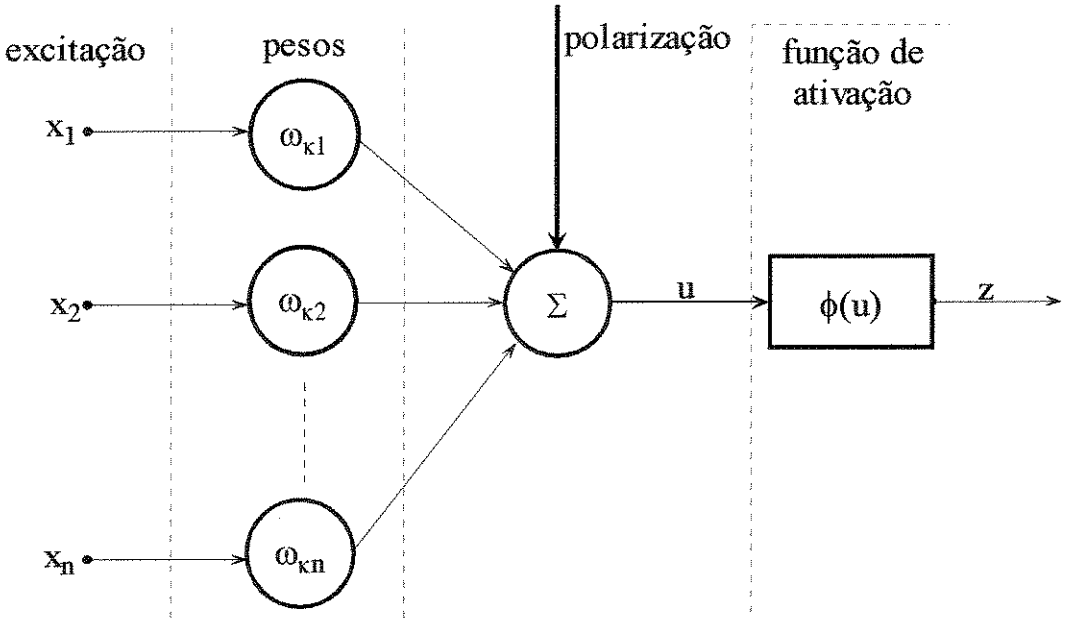


Figura 3.1 – Ilustração de um neurônio artificial.

As redes neurais são projetadas para realizar tarefas específicas, moldando-se aos estímulos externos (entradas da rede) para fornecer saídas desejadas (soluções às tarefas propostas). Para tanto, empregam-se algoritmos de aprendizagem que procuram adaptar os pesos associados aos neurônios, sob a supervisão de um funcional a ser otimizado (Haykin, 2001).

A arquitetura da rede neural é definida pelo número de neurônios e pela sua interconexão. Os neurônios são organizados em camadas de entrada, intermediárias e de saída. O número de camadas intermediárias pode variar, porém é comum utilizar-se até duas camadas. A Figura 3.2

ilustra duas grandes famílias de arquiteturas de redes neurais: (a) *multi-layer perceptron* (MLP) e (b) recorrentes (RNR). Admite-se apenas a recorrência da saída à entrada.

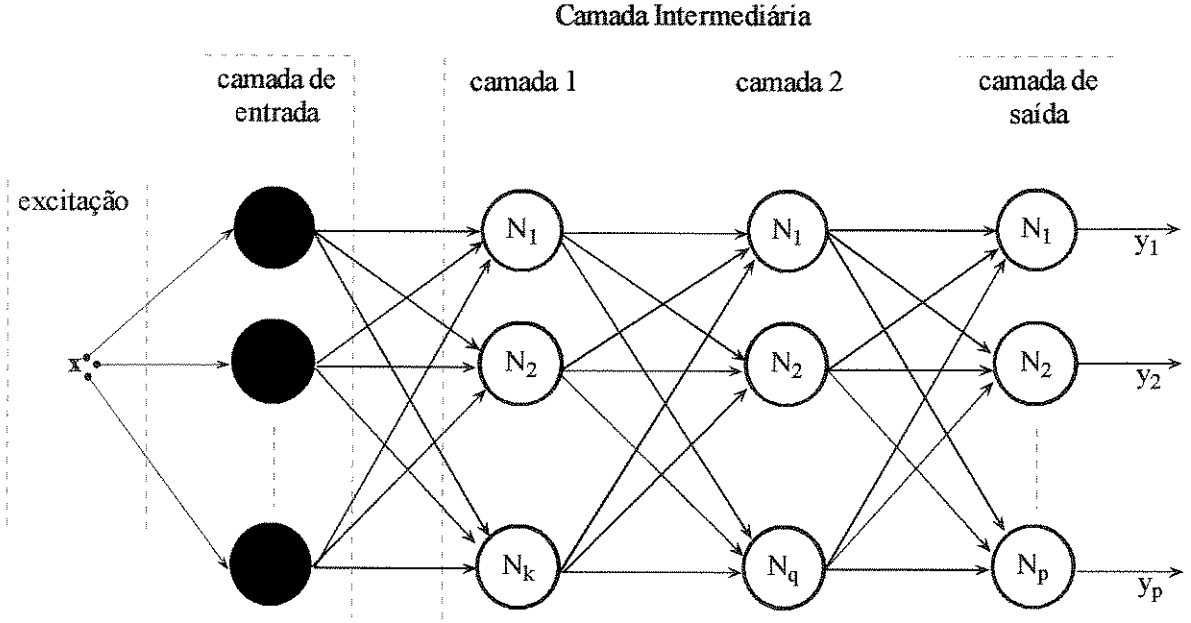


Figura 3.2 (a) – Arquiteturas de redes neurais artificiais (MLP).

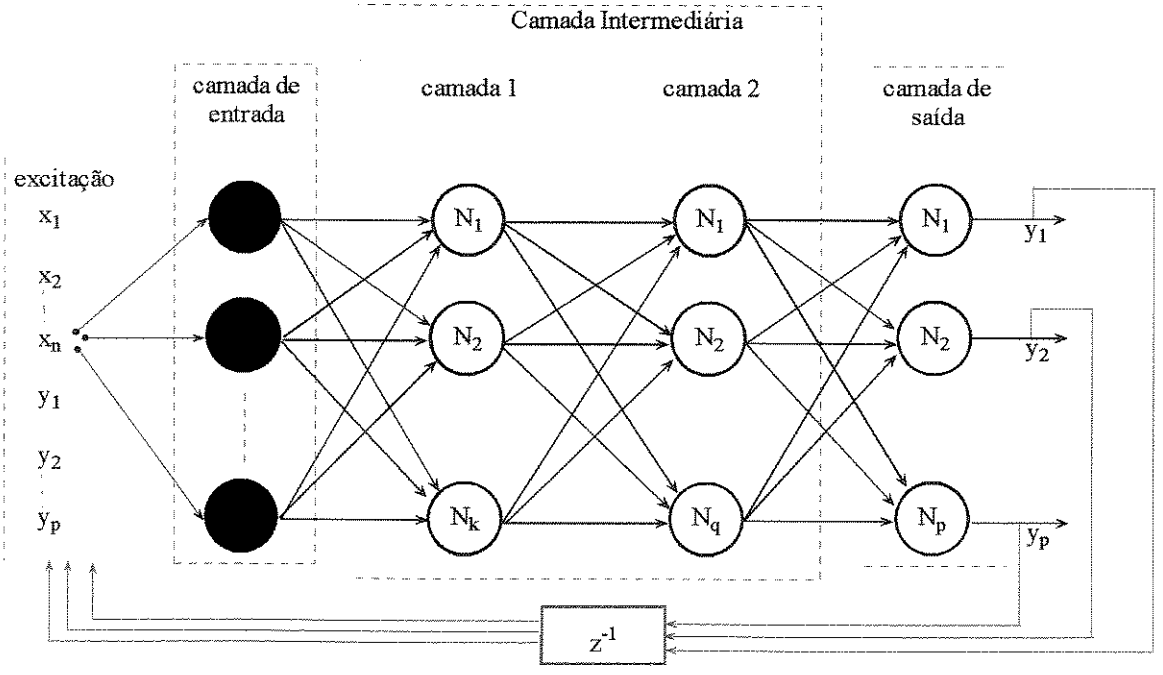


Figura 3.2 (b) – Arquiteturas de redes neurais artificiais (RNR).

Com base na Figura 3.2 (a) e considerando apenas uma camada intermediária, a k-ésima saída da rede é computada pela Equação (3.1).

$$y_k = g_k^s\left(\sum_{j=1}^{m} v_{kj} u_j + s_k\right) \tag{3.1}$$

$$u_j = g_j^o\left(\sum_{\ell=1}^{n} \omega_{j\ell} x_\ell + b_j\right) \tag{3.2}$$

Onde:

$g^s(\bullet)$	função de ativação da camada de saída;
v	pesos da camada de saída;
s	polarização da camada de saída;
u	saída da camada intermediária.
$g^o(\bullet)$	função de ativação da camada intermediária;
ω	pesos da camada intermediária;
b	polarização da camada de intermediária
x	padrões de entrada.

Com base na Figura 3.2 (b) e considerando-se apenas uma camada intermediária, a Equação (3.1) permanece válida, incluindo-se apenas a dependência temporal. A Equação (3.2) pode ser modificada, resultando na Equação (3.3).

$$u_j(t) = \varphi_j\left(\sum_{\ell=1}^{n} \omega_{j\ell} x_\ell(t) + \sum_{k=1}^{p} \omega_{jk} y_k(t-\tau) + b_j\right) \tag{3.3}$$

O termo adicional é devido à retroação da saída da rede neural, atrasada por τ unidades de tempo.

As Equações (3.1), (3.2) e (3.3) admitem funções de ativação diferentes para os neurônios e para as camadas. As funções de ativação mais empregadas são: sigmóide, tangentes hiperbólicas e lineares (Haykin, 2001).

Durante o processo de aprendizagem supervisionada, os pesos da rede neural são ajustados para minimizar o erro quadrático médio, descrito em (3.4).

$$J(\Omega) = \frac{1}{2}\sum_{\ell=1}^{n}\sum_{k=1}^{p}(\hat{y}_{\ell k} - y_{\ell k})^2 \tag{3.4}$$

Onde:

$\hat{y}_{\ell k}$ k-ésima saída estimada pela rede neural associada a ℓ-ésima amostra;

$y_{\ell k}$ Padrão de comparação (valor verdadeiro).

As redes neurais MLP e RNR admitem ajustar os seus pesos de conformidade com a regra do gradiente associado a um termo de *momentum* (β), descrito em (3.5).

$$\Omega_{q+1} = \Omega_q - \alpha \nabla J(\Omega_q) + \beta \Delta\Omega_{q-1} \tag{3.5}$$

Onde:

α Taxa de aprendizagem ($0<\alpha\leq 1$);

β *Momentum*;

q q-ésima iteração;

$\Delta\Omega_{q-1} = \Omega_q - \Omega_{q-1}$ Diferença entre os pesos, em interações subseqüentes;

Neste trabalho a taxa de aprendizagem é determinada pelo algoritmo *gold*, conforme apresentado por Von Zuben *et al.* (1998). Os gradientes são computados pelo algoritmo *back-propagation* que toma a derivada parcial do Funcional (3.4) em relação aos pesos da rede neural.

Os gradientes da camada de saída e da camada intermediária são calculados por meio da regra da cadeia, cujos resultados estão descritos pelas Equações (3.6) e (3.7), respectivamente.

$$\frac{\partial J(v)}{\partial v_{kj}} = \sum_{\ell=1}^{L} (\hat{y}_{\ell k} - y_{\ell k}) z_{kj} \qquad k = 1,...,p \text{ e } j = 1,...,m \tag{3.6}$$

$$\frac{\partial J(\omega)}{\partial \omega_{ji}} = \sum_{\ell=1}^{L} \sum_{k=1}^{p} (\hat{y}_{\ell k} - y_{\ell k}) v_{ji} \frac{dg(u_{\ell j})}{du_{\ell j}} x_{ji} \qquad j = 1,...,m \text{ e } i = 1,...,n \tag{3.7}$$

A Figura 3.3 ilustra uma rede neural de base radial (rede RBF) considerando três camadas: *entrada, intermediária* e *saída.* Adicionalmente, possuindo pesos (**v**) apenas na camada de saída, e funções de base radial (ρ) na camada intermediária.

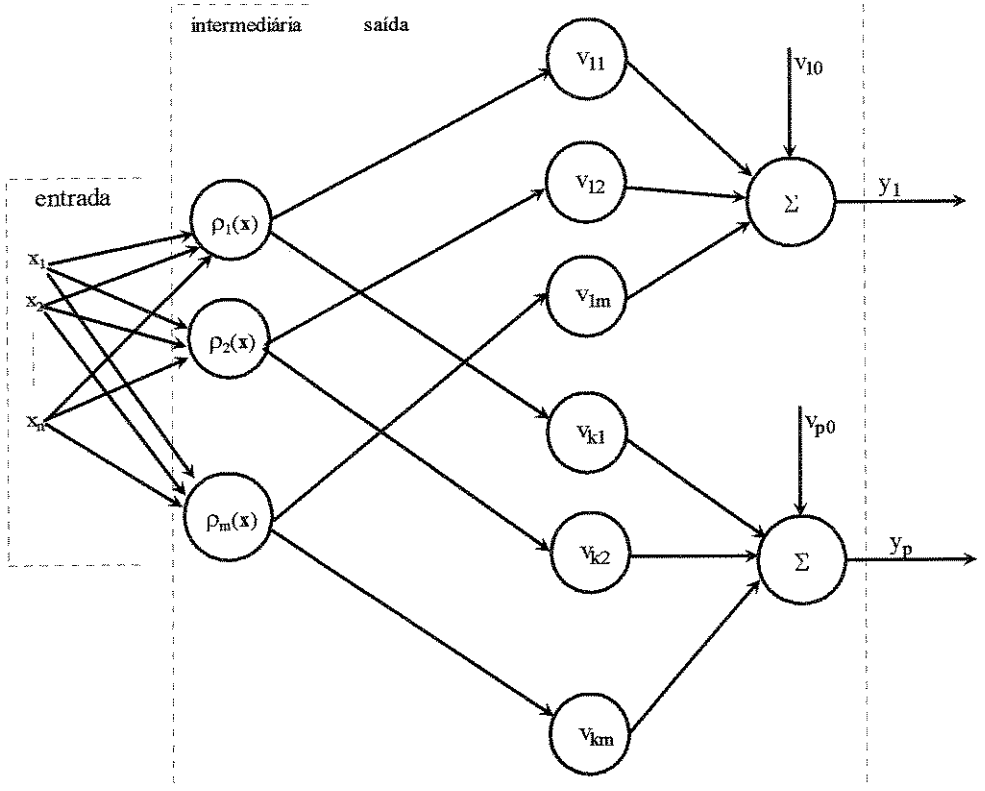


Figura 3.3 – Rede neural de base radial (rede RBF).

A k-ésima saída da rede RBF é definida pela Equação (3.8).

$$y_k(t) = \sum_{j=1}^{m} v_{kj} \rho_j\left(\left\|\mathbf{x}(t) - \boldsymbol{\mu}_j\right\|\right) + s_k \tag{3.8}$$

Onde:

$\mathbf{v}$	Pesos da camada de saída;
$\rho(\bullet)$	Função de ativação de base radial;
s_k	Polarização associado a k-ésima saída da rede, y_k;
$\mathbf{x}$	excitação $\in \Re^n$;
μ_j	centros (ou médias) das funções de base radial, $\in \Re^n$.

Como exemplos, as funções de base radial comumente utilizadas são as *gaussianas*, a multiquadrática de *Hardy* e a inversa de *Hardy*, que utilizam médias (μ) e variâncias (σ), descritas pelas Equações 3.9, 3.10 e 3.11, respectivamente.

$$\rho_j(\mathbf{x}) = \exp\left[\frac{-(\mathbf{x}-\mu_j)^T(\mathbf{x}-\mu_j)}{\sigma_j^2}\right] \tag{3.9}$$

$$\rho_j(\mathbf{x}) = \sqrt{\sigma_j^2 + (\mathbf{x}-\mu_j)^T(\mathbf{x}-\mu_j)} \tag{3.10}$$

$$\rho_j(\mathbf{x}) = \frac{1}{\sqrt{\sigma_j^2 + (\mathbf{x}-\mu_j)^T(\mathbf{x}-\mu_j)}} \tag{3.11}$$

Diferentemente do ajuste de pesos associado às redes MLP e RNR, para a rede RBF o ajuste dos pesos é bastante simplificado, sendo seu principal problema associado à determinação dos centros μ. A formulação apresentada em Chen *et. al.* (1991) para uma escolha dos centros foi estendida em Munzir *et. al.* (2000) para contemplar uma rede neural com *p* saídas.

Primeiramente, reescrevendo a Equação (3.8) como sendo um modelo de regressão linear, na forma matricial, resulta:

$$\begin{bmatrix} \mathbf{d}^1 \\ \mathbf{d}^2 \\ \vdots \\ \mathbf{d}^p \end{bmatrix}^T = \begin{bmatrix} \mathbf{\Phi}^1 \\ \mathbf{\Phi}^2 \\ \vdots \\ \mathbf{\Phi}^p \end{bmatrix} \begin{bmatrix} \mathbf{v}^1 & \mathbf{v}^2 & \cdots & \mathbf{v}^p \end{bmatrix} + \begin{bmatrix} \varepsilon^1 \\ \varepsilon^2 \\ \vdots \\ \varepsilon^p \end{bmatrix} \tag{3.12}$$

Onde:

$$\mathbf{d}^j = \begin{bmatrix} y_j(0) & y_j(T) & y_j(2T) & \cdots & y_j(kT) & \cdots & y_j(t_f) \end{bmatrix}^T$$

$$\mathbf{v}^j = \begin{bmatrix} v_{j1} & \mathbf{v}_{j2} & \cdots & \mathbf{v}_{jm} \end{bmatrix}^T$$

$$\mathbf{\Phi}^j = \begin{bmatrix} \rho_1(0) & \cdots & \rho_m(0) \\ \vdots & \vdots & \vdots \\ \rho_1(t_f) & \cdots & \rho_m(t_f) \end{bmatrix}$$

Assim, o vetor de pesos pode ser obtido pela Equação (3.13):

$$v = \left(\Phi^T \Phi\right)^{-1} \Phi^T d \tag{3.13}$$

A questão relevante para a aplicação da Equação (3.13) é a escolha adequada dos centros da rede RBF. Uma escolha arbitrária poderá não garantir desempenho adequado à respectiva rede neural, além da questão de alta dimensionalidade e mau-condicionamento da matriz regressão. Esse problema foi tratado em (Chen *et. al*, 1991) que empregou o OLS (*orthogonal least squares*) cujo algoritmo estendido (Munzir *et. al.*, 2000) é apresentado no *Algoritmo 3.1: Determinação dos Centros Significativos para a Rede RBF.* Nos algoritmos 3.2 (*Determina o Primeiro Centro Significativo)* e 3.3 (*Determina os Demais Centros Significativos*) são apresentados os algoritmos associados.

Algoritmo 3.1: Determinação dos Centros Significativos para a Rede RBF

OLS_MIMO(X,Y,ρ)

```
Início
    ok ← 1;                              // Variável lógica
    k ← 1;                               // Número de centros escolhidos.
    P ← rbf(X);                          // Computa uma função de base radial escolhida.
    [err(k),ik(k)] ← first_step(P,Y);    // Escolhe o primeiro centro.
    centro(:,k) ← X(:,ik(k));            // Armazena o primeiro centro escolhido.
    Faça{
        k ← k +1;
        [centro(:,k),ik(k),ok] ← second_step(X,P,Y,ik,k,ρ);  // Demais centros.
    }Até que (ok = 0 ou k = m)
Fim
```

Algoritmo 3.2: Determina o Primeiro Centro Significativo

FIRST_STEP(P,Y)

```
Início
    Para i ← 1 até m faça                // m centros candidatos
        g ← 0;
        Para r ← 1 até p faça            // p saídas da rede neural
            g ← g + P(:,i)^T×Y(:,r)/traço(Y^T×Y);
        Fim-para
        err(i) ← (g^2)×P(:,i)^T×P(:,i)/ traço(Y^T×Y);
    Fim-para
    [max,pos] = valor_máximo(err);       // Acha o valor máximo e a sua posição;
    Retornar(max,pos);
Fim
```

Algoritmo 3.3: Determina os Demais Centros Significativos

SECOND_STEP(X,P,Y,ik,k,ρ)

```
Início
    soma_err ← 0;
    ok ← 0;
    Para i ←1 até m faça                    // m centros candidatos
        ok ← 1;
        Para z ← 1 até k faça
            Se (i = ik(z)),
                ok ← 0;
            Fim-se
        Fim-para
        Se (ok = 1),
            a_soma ← zeros(N,1);            // Monta vetor nulo de N linhas e 1 coluna.
            Para j←1 até (k-1) faça
                a_soma ← a_soma + P(:,j)×(P(:,ik(j))^T×P(:,i)/ (P(:,ik(j))^T×P(:,ik(j))));
            Fim-para
            wk(:,i) ← P(:,i) – a_soma;
            g ← 0;
            Para r ← 1 até p faça           // p saídas da rede neural
                g ← g + wk(:,i)^T×Y(:,r)/traço(wk(:,i)^T×wk(:,i));
            Fim-para
            err(i) ← (g^2)×wk(:,i)^T×wk(:,i)/ traço(Y^T×Y);
            soma_err ← soma_err + err(i);
    Fim-para
    [max,pos] = valor_máximo(err);          // Acha o valor máximo e a sua posição;
    Se (1-soma_err < ρ)                     // Avalia o critério de parada
        ok ← 0;
    Fim-se
    Retornar(X(:,pos),pos,ok);
Fim
```

3.1.2. Algoritmos Genéticos

Algoritmos genéticos utilizam conceitos da reprodução biológica objetivando fornecer um conjunto de operadores para resolver problemas de otimização. Diferentemente de técnicas clássicas de otimização, os algoritmos genéticos operam sobre uma população de cromossomos artificiais (indivíduos) que encapsulam possíveis soluções para um problema sob análise. Os indivíduos são avaliados segundo algum critério (função de adequação) e operadores de reprodução artificial são aplicados aos indivíduos escalonados (pela probabilidade acumulativa) a partir do resultado da função de adequação, gerando uma nova população (geração). O processo de avaliação é aplicado à nova geração. Esse processo é realizado sucessivamente (por gerações) até que um critério de parada seja atendido. Um cromossomo artificial (genótipo) é uma cadeia de caracteres (genes) que codifica as características de um indivíduo (fenótipo). A Figura 3.4 ilustra a aplicação de um algoritmo genético clássico.

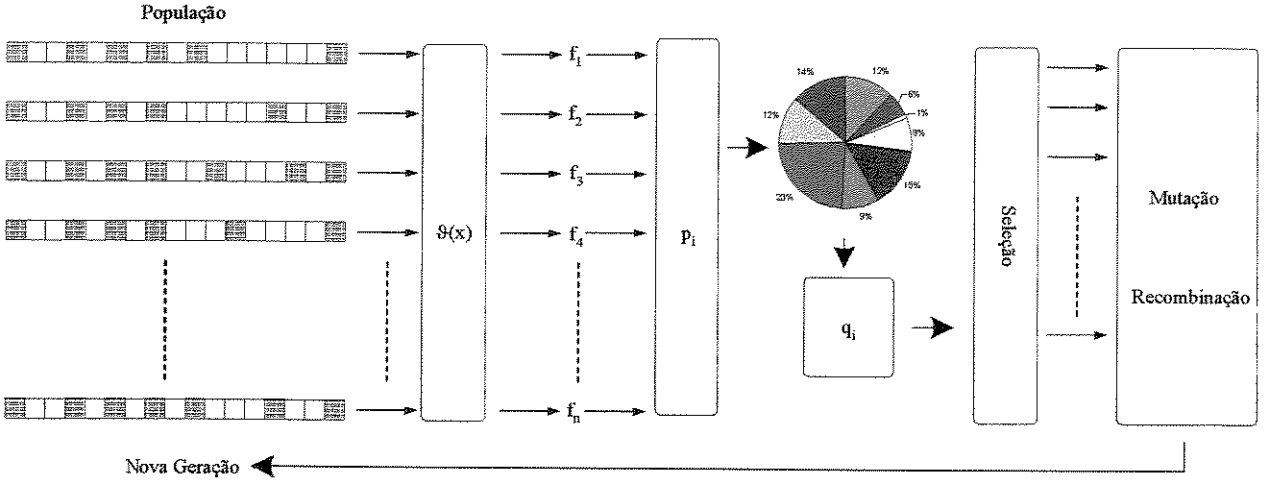


Figura 3.4 – Ilustração do algoritmo genético clássico.

A função de adequação ϑ(x) é um critério de desempenho característico do problema a ser otimizado, portanto é um funcional representativo do mesmo. Cada valor de adequação é avaliado pela Equação (3.15), fornecendo uma medida percentual do "indivíduo representar uma solução para o problema em análise". A Equação (3.16) computa a probabilidade acumulativa.

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i} \tag{3.15}$$

$$q_i = \sum_{j=1}^{i} p_j \tag{3.16}$$

Nesse contexto, o valor p_i é a probabilidade do i-ésimo indivíduo de participar na geração de uma nova população de indivíduos. Por efeito, o critério de seleção pode utilizar os resultados de (3.16) elegendo os indivíduos com maiores probabilidades de serem soluções possíveis para o problema, para participarem no processo de reprodução artificial. Basicamente, há dois modos de gerar novos indivíduos: por *recombinação*, que consiste em escolher arbitrariamente dois indivíduos selecionados e intercambiar seus genes; e por *mutação*, que objetiva alterar aleatoriamente os genes de um indivíduo selecionado por algum critério. Um algoritmo genético clássico é exemplificado no *Algoritmo 3.4: Algoritmo Genético Clássico*, dado a quantidade (M) de genes do individuo:

Algoritmo 3.4: Algoritmo Genético Clássico

```
ALGORITMO_GENÉTICO(M)
Início
    N ← pop_size;                  // Arbitre a quantidade de indivíduos.
    pop ← Pop_inicial(N,M);        // Arbitre a população inicial.
    k ← 0;                         // Geração inicial.
    [p,q] ← Aval_adequação(pop,N); // Aplica a função de adequação pertinente ao problema.
    Repita {
        k ← k+1;                   // Cria novas gerações.
        pop_s ← Seleção(pop,q);    // Aplica o critério de seleção.
        pop ← Recombinação(pop_s); // Aplica a reprodução por recombinação.
        pop ← Mutação(pop);        // Aplica alteração genética.
        f ← Aval_adequação(pop);
    } até que (condição = ok).
Fim
```

O *Algoritmo 3.5: Cômputo da Probabilidade de Seleção* apresenta o cômputo da função de adequação que é avaliada para cada indivíduo pertencente à população inicialmente gerada. Os argumentos para a função *AVAL_ADEQUAÇÂO* são: a população (pop) e o tamanho da população (N). Retornando a probabilidade acumulativa (q), como resultado do processamento.

Algoritmo 3.5: Cômputo da Probabilidade de Seleção

```
AVAL_ADEQUAÇÃO(pop,N)
Início
    k ← 1;
    F ← 0;
    Repita {
        f[k] ← ϑ(pop[k]);          // Computa a função de adequação.
        F ← F + f[k];              // Computa a função de adequação total.
        q[k] ← 0;
        k ← k+1;
    } até que ( k=N).
    k ← 1;
    j ← 1;
    Repita {
        p[k] ← f[k]/F;             // Computa a probabilidade de seleção.
        Repita {
            q[k] ← q[k] + p[k];    // Computa a probabilidade acumulativa.
            j ← j+1;
        }até que (j=k).
        k ← k+1;
    } até que ( k=N).
    Retorna(q);
Fim
```

O *Algoritmo 3.6: Seleção Aleatória de Indivíduos* ilustra o processo de seleção dos indivíduos que gerarão nova população, possivelmente, melhor qualificada para resolver o problema em questão. A função *Seleção* possui os seguintes argumentos: a população (pop), a

probabilidade acumulativa (q) e o tamanho da população (N). Esse procedimento retorna uma população escalonada segundo a probabilidade acumulativa, como resultado do processamento.

Algoritmo 3.6: Seleção Aleatória de Indivíduos

```
SELEÇÃO(pop,q,N)
Início
    k ← 1;
    Repita {
        r ← randômico(0,1);              // Valor aleatório ∈ (0,1).
        Se ( r < q[1] )
            pop_s[k] ← pop[1];
        Senão
        j ← 2;
        Repita {
            Se (r > q[j-1]  .e.  r ≤ q[j])
                pop_s[k] ← pop[j];
            j ← j+1;
        }até que (j=N).
        k ← k+1;
    } até que ( k=N).
    Retorna(pop_s);
```

O *Algoritmo 3.7: Intercambiar Genes entre Indivíduos Selecionados* apresenta um procedimento para o intercâmbio de genes entre indivíduos selecionados. A função *Recombinação* recebe os seguintes argumentos: a população (pop), a probabilidade de intercambio (p_c) e o tamanho da população (N). A função retorna uma nova população gerada pela recombinação de material genético, como resultado do processamento.

Algoritmo *3.7: Intercambiar Genes entre Indivíduos Selecionados*

```
RECOMBINAÇÃO(pop_s,pc,N)
Início
    k ← 1;
    nind ← 0;
    Repita {
        r ← randômico(0,1);              // Valor aleatório ∈ (0,1).
        Se (r < pc){
            nind ← nind + 1;
            sel[nind] ← k;
        }
        k ← k+1;
    }até que (k=N).
    k ← 1;
    Repita {
        pos ← randômico(0,M);            // Valor inteiro aleatório ∈ [0,M].
        Repita {
            temp ← pop_s[sel[k],pos];
            pop_s[sel[k],pos] ← pop_s([sel[k+1],pos];
            pop_s([sel[k+1],pos] ← temp;
            pos ← pos+1;
        }até que (pos < M).
        k ← k+1;
    } até que ( k=nind).
    Retorna(pop_s);
Fim
```

O Algoritmo *3.8: Alteração dos Genes de Indivíduos Selecionados* ilustra um procedimento para a alteração de genes de indivíduos selecionados, realizada pela função *Mutação* cujos argumentos são: a população (pop), a probabilidade de mutação (p_m) e o tamanho da população (N). A função retorna uma nova população gerada pela mutação de material genético, como resultado do processamento.

Algoritmo ***3.8: Alteração dos Genes de Indivíduos Selecionados***

```
MUTAÇÃO(pop_s,pm,N)
Início
    k ← 1;
    Repita {
        j ← 1;
        Repita{
            r ← randômico(0,1);          // Valor aleatório ∈ (0,1).
            Se (r < pm){
            Se (pop_s[k,j] = 1)
            pop_s[k,j] ← 0;
            Senão
            pop_s[k,j] ← 0;
            }
            j ← j+1;
        }até que (j=M).
        k ← k+1;
    }até que (k=N).
    Retorna(pop_s);
Fim
```

O *Algoritmo 3.4* pode ser modificado para melhor solucionar certos problemas. Por exemplo, o indivíduo mais apto da geração anterior pode ser incluso na próxima geração.

As informações necessárias para projetar algoritmos genéticos são: *tamanho da população* (pop_size), *a taxa de recombinação* (p_c), *a taxa de mutação* (p_m), *a função de adequação* (ϑ) e *o critério de parada.*

As simulações apresentadas neste trabalho empregam implementações em comandos do *Matlab*® V5.3 R12 para o algoritmo clássico, modificado para incluir o indivíduo mais apto de gerações anteriores.

3.3. Movimento Pendular

Para assegurar marcha dinâmica e gerar trajetórias desejadas (quaisquer) para os membros inferiores, o robô bípede é dotado de um tronco (pêndulo invertido) visando compensar as forças inerciais e gravitacionais intrínsecas a andadura dinâmica. Contudo, esse acréscimo origina problemas inerentes à estabilidade. Primeiramente, há o problema de mantê-lo sob controle na posição vertical. Dotá-lo de juntas motoras e empregar sistema de controle com retroação é uma alternativa para resolver essa questão. Outro problema está relacionado à geração de trajetória do próprio pêndulo invertido. Podendo ser resolvido considerando-se a dinâmica de contato entre o pé de apoio e o solo que produz forças generalizadas expressas no CoP. No ponto de contato, quando os momentos nas direções do plano de apoio são nulos denominado-se o CoP por ZMP (*Zero Moment Point*). Os momentos nas direções do plano de apoio é que contribuem para a perda do balanço dinâmico do robô bípede. Assim, controlar a ação desses momentos permite manter a estabilidade postural do robô bípede ao caminhar. O emprego do critério ZMP para garantir a estabilidade postural fornece um sistema de equações diferenciais não-lineares que pode ser resolvida para produzir ou a localização espacial do pêndulo invertido ou a localização espacial do ZMP.

3.3.1. Modelagem do ZMP

A Figura 3.11 ilustra um robô bípede no plano sagital. A massa (m_i) de cada elo está concentrada em seu respectivo centro de massa (CoM_i). O peso total do robô e o centro de massa global são designados por P e CoM, respectivamente. Designa-se por *polígono suporte* a área de contato entre o pé de apoio e o solo. O ponto onde atuam as forças e momentos resultantes do contato entre o pé e o solo é designado por E (circunscrito ao polígono suporte, área de contato entre o pé e o solo). O vetor $\vec{r}_i$ fornece a distância dos CoM_i em relação a origem O-XYZ. A aceleração linear é computada por $a_i \equiv d^2 r_i / dt^2$. Um sistema móvel de coordenadas é alocado ao quinto elo (e_5), com origem em $\overline{O}$, conforme ilustra a Figura 3.5.

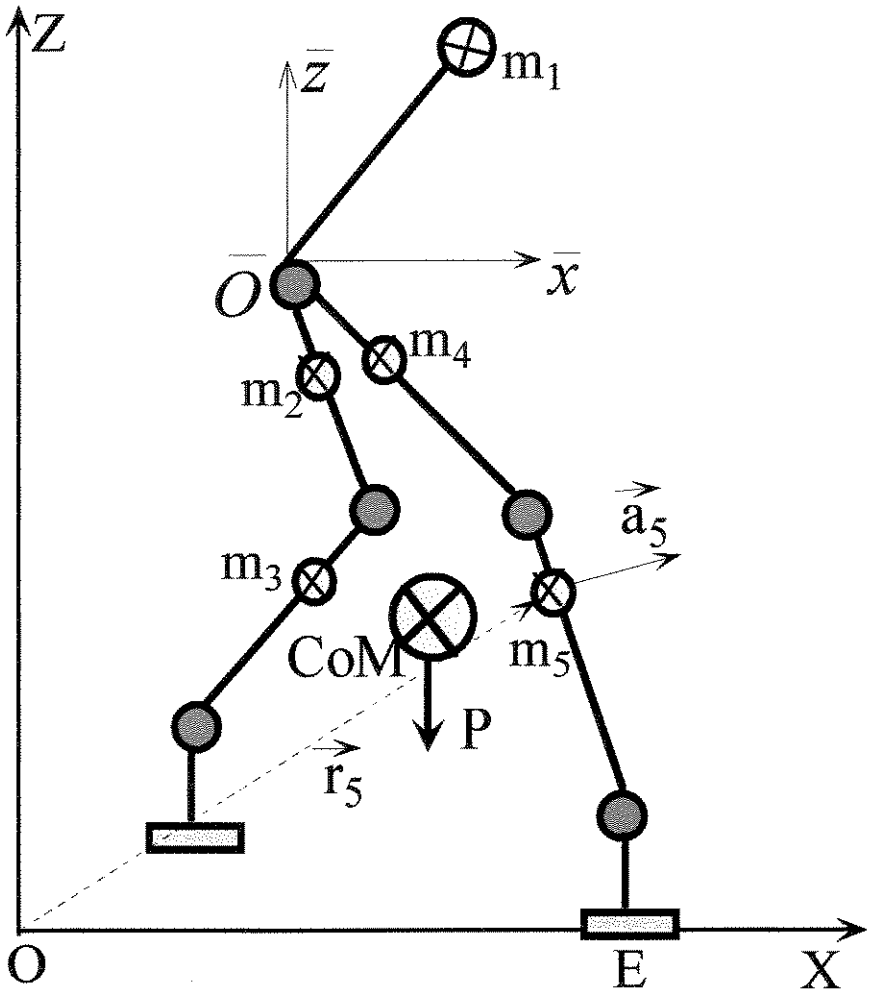


Figura 3.5 – Robô Bípede no Plano Sagital

A Figura 3.6 ilustra uma massa concentrada (m_i) cuja localização no espaço tridimensional é descrita em termos do vetor $\vec{r}_i$. Um vetor $\vec{s}$ descreve a localização do ponto E, no plano XY; $\vec{r} - \vec{s}$ define a altura da massa em relação ao solo, medida ao longo do eixo Z. Também mostra a força R e o momento tangencial T, resultantes da aplicação da força peso e forças inerciais, definidas pelas Equações (3.17) e (3.18), respectivamente.

$$\mathbf{R} = -\mathbf{F}_z \tag{3.17}$$

$$\mathbf{T} = \mathbf{F} \times (\vec{\mathbf{r}}_i - \vec{\mathbf{s}}) \tag{3.18}$$

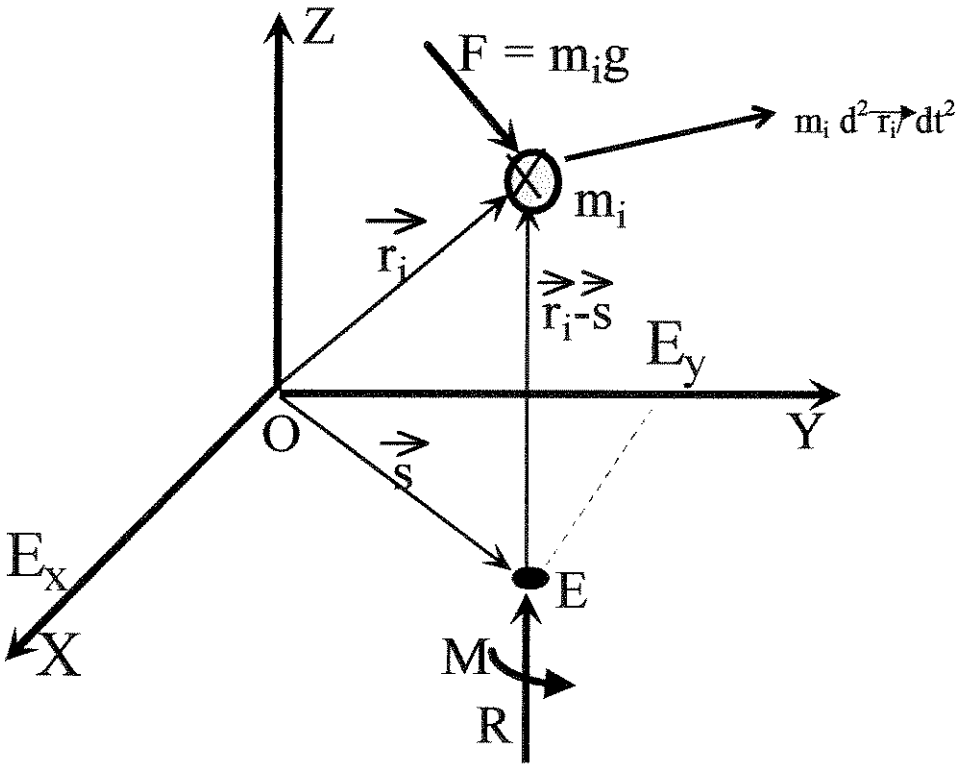


Figura 3.6 – Descrição tridimensional de uma massa concentrada.

Com base na Figura 3.6, é derivada uma equação de movimento para o modelo do robô bípede, não levando em consideração as forças tangenciais (Goswami, 1999).

$$\mathbf{T} - \sum_{i=1}^{n} (\mathbf{r}_i - \mathbf{s}) \times \mathbf{m}_i (\mathbf{g} + \mathbf{a}_i) = \mathbf{0} \tag{3.19}$$

Onde:

$$r_i = [x_i \;\; y_i \;\; z_i]^T \tag{3.20}$$

$$s = [E_x \;\; E_y \;\; 0]^T \tag{3.21}$$

$$g = [0 \;\; 0 \;\; g_z]^T \tag{3.22}$$

$$a_i = [\ddot{x}_i \ \ddot{y}_i \ \ddot{z}_i]^T \tag{3.23}$$

$$T = [T_x \ T_y \ T_z]^T \tag{3.24}$$

Sejam as seguintes definições:

$$\mathbf{r_i - s} \overset{\Delta}{=} [\mathbf{X^i \ Y^i \ Z^i}]^T \tag{3.25}$$

$$\mathbf{m_i(g + a_i)} = [\mathbf{F_x^i \ F_y^i \ F_z^i}]^T \tag{3.26}$$

Utilizando-se as Equações (3.25) e (3.26) para realizar o produto vetorial em (3.19), resulta:

$$\begin{vmatrix} \vec{\mathbf{i}} & \vec{\mathbf{j}} & \vec{\mathbf{k}} \\ \mathbf{X^i} & \mathbf{Y^i} & \mathbf{Z^i} \\ \mathbf{F_x^i} & \mathbf{F_y^i} & \mathbf{F_z^i} \end{vmatrix} = \left(\mathbf{Y^i F_z^i - Z^i F_y^i}\right)\vec{\mathbf{i}} + \left(\mathbf{Z^i F_x^i - X^i F_z^i}\right)\vec{\mathbf{j}} + \left(\mathbf{X^i F_y^i - Y^i F_x^i}\right)\vec{\mathbf{k}} \tag{3.27}$$

Utilizando a Equação (3.27) em (3.19), resulta[2]:

$$\begin{Bmatrix} T_x \\ T_y \\ T_z \end{Bmatrix} = \sum_{i=1}^{n} m_i \begin{Bmatrix} (y_i - E_y)(\ddot{z}_i + g_z) - (z_i - E_z)(\ddot{y}_i + 0) \\ (z_i - E_z)(\ddot{x}_i + 0) - (x_i - E_x)(\ddot{z}_i + g_z) \\ (x_i - E_x)(\ddot{y}_i + 0) - (y_i - E_y)(\ddot{x}_i + 0) \end{Bmatrix} \equiv \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \tag{3.28}$$

Considera-se que não há deslizamento entre o pé e o solo (pé em equilíbrio estático). Para garantir a estabilidade postural são de interesse as análises dos torques nas direções X e Y, que devem ser nulos, como indicado em (3.28). Assim, resolvendo as respectivas equações para as coordenadas do ponto E no plano XY, resultam:

[2] T_Z deve ser nulo para manter o pé de apoio em equilíbrio estático. Entretanto, neste trabalho, considera-se que não há escorregamento entre o pé de apoio e o solo.

$$E_x = \frac{\sum_{i=1}^{n} m_i \left[x_i (g_z + \ddot{z}_i) - z_i (0 + \ddot{x}_i) \right]_{\Delta}}{\sum_{i=1}^{n} m_i (g_z + \ddot{z}_i)} = x_{zmp} \tag{3.29}$$

$$E_y = \frac{\sum_{i=1}^{n} m_i \left[y_i (g_z + \ddot{z}_i) - z_i (0 + \ddot{y}_i) \right]_{\Delta}}{\sum_{i=1}^{n} m_i (g_z + \ddot{z}_i)} = y_{zmp} \tag{3.30}$$

As equações (3.29) e (3.30) descrevem as coordenadas cartesianas do ponto E no plano XY, cuja somatória de momentos gerados por forças gravitacionais e inércias é nula (definição do ZMP).

3.3.2. Solução para o Deslocamento do Tronco

A Figura 3.7 ilustra o deslocamento do ponto E (que localiza as coordenadas planejadas para o ZMP), sendo **L** o comprimento do quinto elo e **λ** o comprimento do passo, cuja trajetória pode ser descrita por uma equação de reta (Sugihata *et al.*, 2002). Conforme a Figura 3.7, o caminhar é iniciado com o pé esquerdo (PE) percorrendo uma distância de λ metros (passo); o pé direito (PD) completa a passada (percorrendo o dobro do passo). A localização de E é planejada de tal modo que siga os pontos E_1, E_2, etc, caracterizando o modo de andar dinâmico.

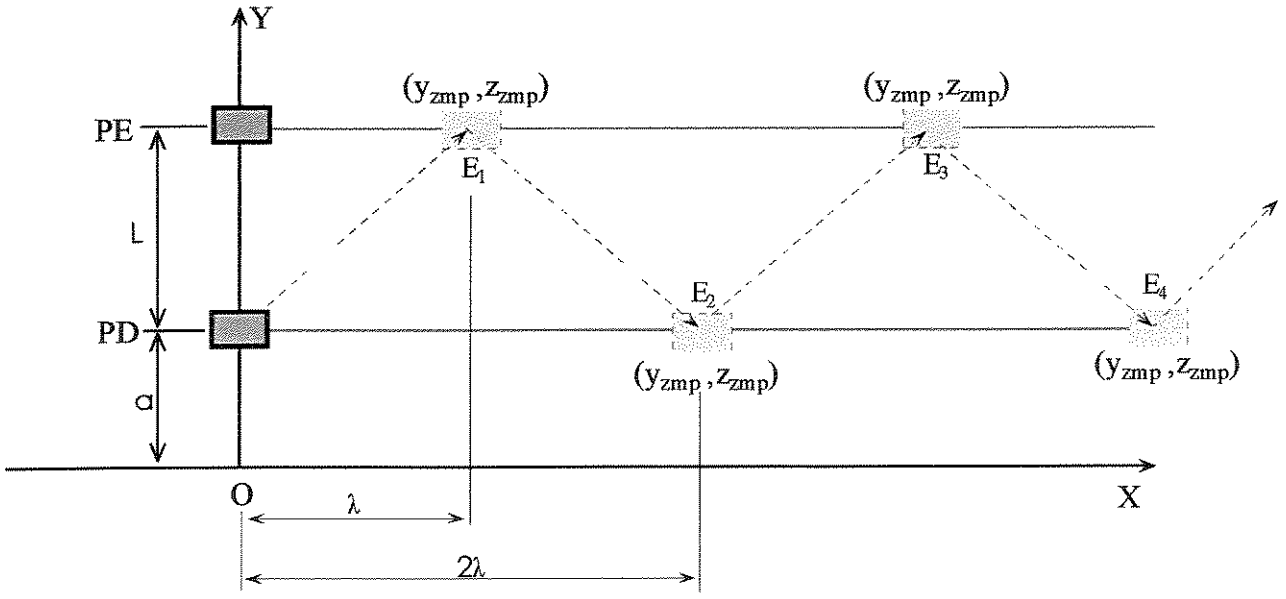


Figura 3.7 – Vista superior do robô bípede, ilustrando o deslocamento do ZMP planejado.

O objetivo é descrever a trajetória do tronco (pêndulo invertido) de modo a fornecer estabilidade postural à marcha imposta aos membros inferiores. O sistema móvel de coordenadas (SCT), alocado à base do pêndulo invertido e cuja relação com o sistema global é definida em (3.31), é ilustrada na Figura 3.8.

$$\vec{r}_i = \vec{q} + \vec{p}_i = \begin{bmatrix} x_q & y_q & z_q \end{bmatrix}^T + \begin{bmatrix} \bar{x}_i & \bar{y}_i & \bar{z}_i \end{bmatrix}^T \tag{3.31}$$

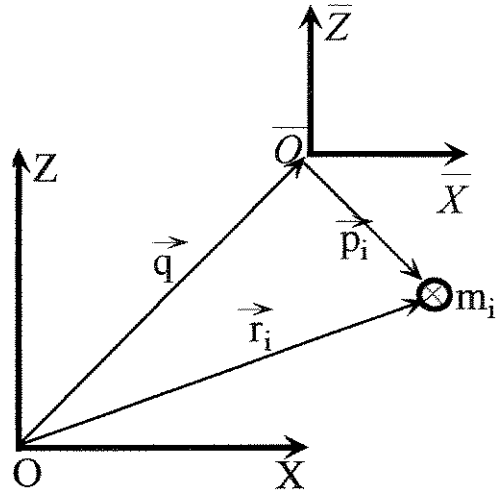


Figura 3.8 – Relação entre os sistemas inercial e móvel, vista no plano sagital.

Expandindo as Equações (3.29) e (3.30), as variáveis do tronco são explicitadas, sendo descritas no sistema coordenado móvel. Após as manipulações algébricas que isolam no lado esquerdo da igualdade as variáveis do tronco, resulta um sistema de equações diferenciais de segunda ordem, não-lineares e acopladas.

$$\left(\ddot{\bar{z}}_{11}+\ddot{z}_q\right)\left(\bar{x}_{11}+x_q-x_{zmp}\right)-\left(\ddot{\bar{x}}_{11}+\ddot{x}_q\right)\left(\bar{z}_{11}+z_q\right)+\left(\bar{x}_{11}+x_q\right)g_z-\left(\bar{z}_{11}+z_q\right)g_x=\kappa_1 \qquad (3.31)$$

$$\left(\ddot{\bar{z}}_{11}+\ddot{z}_q\right)\left(\bar{y}_{11}+y_q-y_{zmp}\right)-\left(\ddot{\bar{y}}_{11}+\ddot{y}_q\right)\left(\bar{z}_{11}+z_q\right)+\left(\bar{y}_{11}+y_q\right)g_z-\left(\bar{z}_{11}+z_q\right)g_y=\kappa_2 \qquad (3.32)$$

Onde:

$$\kappa_1=x_{zmp}g_z+\sum_{i=1}^{10}\frac{m_i}{m_{11}}\left[\left(x_{zmp}-x_i\right)\left(\ddot{z}_i+g_z\right)+z_i\left(\ddot{x}_i+g_x\right)\right] \qquad (3.33)$$

$$\kappa_2=y_{zmp}g_z+\sum_{i=1}^{10}\frac{m_i}{m_{11}}\left[\left(y_{zmp}-y_i\right)\left(\ddot{z}_i+g_z\right)+z_i\left(\ddot{y}_i+g_y\right)\right] \qquad (3.34)$$

Na abordagem clássica, o conjunto de Equações (3.31) e (3.32) é desacoplado considerando-se pequenos deslocamentos verticais do tronco e movimento uniformemente variável na vertical, resultando nas seguintes simplificações:

$\bar{z}_{11}\approx c^{te}$ Altura do tronco relativo ao sistema coordenado móvel;

$z_q\approx c^{te}$ Altura do quadril relativo ao sistema coordenado global;

$\ddot{\bar{z}}_{11}=0$ Aceleração vertical do tronco;

$\ddot{z}_q=0$ Aceleração vertical do quadril.

Há diversos autores que projetam marchas periódicas e utilizam a FFT para obter o posicionamento do pêndulo invertido, resolvendo o modelo simplificado cujo resultado é utilizado para obter, de forma iterativa, uma solução para o modelo original (Yamaguchi *et al.*, 1993; Li *et al.*, 1992, entre outros).

Este trabalho procurou uma abordagem inovadora (Gonçalves *et al.*, 2003), empregando diretamente o modelo original (Equações 3.31 e 3.32) e utilizando uma rede neural artificial (RN) para resolver esse problema clássico de robôs bípedes. A partir do planejamento da marcha e da localização do ZMP, descritos no espaço de trabalho do robô bípede (plano cartesiano), os padrões da RN foram determinados. Um processo iterativo por lote foi empregado para ajustar os pesos da RN. Ajustados os pesos para a RN, a solução passa a ser praticamente instantânea. A RN projetada é inserida na malha de controle possibilitando dotar o robô bípede de "reflexos" (implementação baseada em sensores que indiquem as condições do terreno e as forças que atuam no CoP, por exemplo), ou seja, permitindo a obtenção de posicionamentos instantâneos para o tronco.

3.3.3. Gerador Automático de Trajetórias do Tronco

Para melhor esclarecer o procedimento desenvolvido, sejam as Equações (3.35) e (3.36) que explicitam as variáveis estimadas (^) e as variáveis computadas numericamente (~) a partir das quais as Equações 3.29 e 3.30 são aproximadas pelas Equações (3.35 e (3.36). As variáveis estimadas são obtidas por um processo de otimização, as computadas numericamente são obtidas pela Equação (3.37).

$$\tilde{x}_{zmp} \cong \beta\left\{\left(\tilde{\tilde{z}}_{11} + \ddot{z}_q + g_z\right)\left(\hat{\tilde{x}}_{11} + x_q\right) - \left(\hat{\tilde{z}}_{11} + z_q\right)\left(g_x + \tilde{\tilde{x}}_{11} + \ddot{x}_q\right) + \sum_{i=1}^{10} \frac{m_i}{m_{11}}\left[x_i\left(g_x + \ddot{z}_i\right) - z_i\left(g_x + \ddot{x}_i\right)\right]\right\} \tag{3.35}$$

$$\breve{y}_{zmp} \cong \beta\left\{\left(\breve{\ddot{z}}_{11} + \ddot{z}_q + g_z\right)\left(\hat{y}_{11} + y_q\right) - \left(\hat{z}_{11} + z_q\right)\left(\breve{\ddot{y}}_{11} + \ddot{y}_q + g_y\right) + \sum_{i=1}^{10} \frac{m_i}{m_{11}}\left[y_i\left(g_z + \ddot{z}_i\right) - z_i\left(g_y + \ddot{y}_i\right)\right]\right\} \tag{3.36}$$

Onde:

$$\beta = \frac{1}{\breve{\ddot{z}}_{11} + \ddot{z}_q + g_z + \sum_{i=1}^{n} \frac{m_i}{m_{11}}\left(g_z + \ddot{z}_i\right)}$$

Para computar as Equações (3.35) e (3.36), foi necessário planejar um padrão de marcha e o deslocamento para o ZMP. Em seguida, as coordenadas cartesianas do tronco foram estimadas (^) por meio de técnicas de otimização e a diferenciação numérica central (Equação 3.37) foi empregada para computar os termos de acelerações (~) associados ao tronco.

$$\ddot{\eta}(k) \approx \frac{\eta(k+2) - 2(k) + \eta(k-1)}{\Delta T^2} \tag{3.37}$$

O problema de otimização pode assim ser descrito: Para o padrão de marcha (incluindo o deslocamento do ZMP) planejado para o robô bípede, determinar o vetor de parâmetros Θ (ângulos de juntas associados ao tronco) tal que minimize a Equação (3.38).

$$\begin{cases} \min\limits_{\Theta \in \Re} \left\| \begin{bmatrix} x_{zmp} \\ y_{zmp} \end{bmatrix} - \begin{bmatrix} \breve{x}_{zmp} \\ \breve{y}_{zmp} \end{bmatrix} \right\| \leq \gamma \\ \text{sujeito a : } \quad \Theta_{min} \leq \Theta \leq \Theta_{max} \end{cases} \tag{3.38}$$

Onde:

$\begin{bmatrix} x_{zmp} & y_{zmp} \end{bmatrix}^T$ ZMP planejado;

$\begin{bmatrix} \breve{x}_{zmp} & \breve{y}_{zmp} \end{bmatrix}^T$ ZMP aproximado pelas Equações (3.35) e (3.36);

$\Phi = \{\theta_{11}\ \theta_{12}\}^T$ Ângulos associados às juntas rotativas do tronco;

γ Parâmetro de projeto.

A solução de (3.38) permite gerar padrões de treinamento, possibilitando que o processo de identificação dos pesos de uma rede neural artificial seja efetivado. Diversas simulações foram executadas com diferentes arquiteturas de redes neurais artificiais. A que apresentou melhor resultado (menor erro quadrático médio *vs* poder de generalização) foi uma rede neural recorrente (RNR) com duas camadas intermediárias, apresentada na seqüência. A Figura 3.9 ilustra a estrutura projetada para o treinamento da RNR, que generaliza a solução para as Equações (3.35) e (3.36). A entrada da RNR emprega o padrão de marcha planejado para os membros inferiores, κ_1 e κ_2 conforme as Equações (3.33) e (3.34).

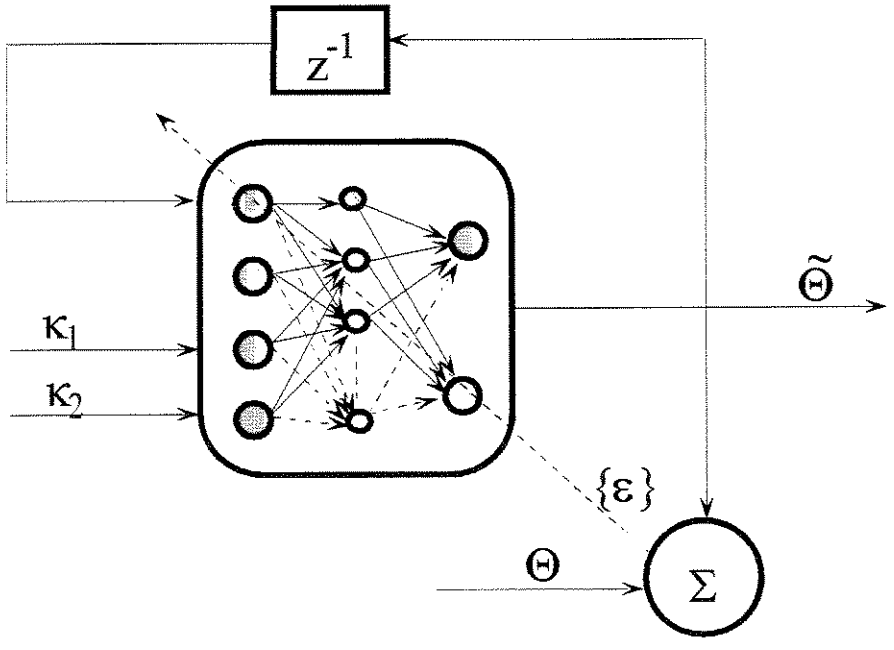


Figura 3.9 – Estrutura do processo de identificação (treinamento da RNR).

O treinamento da RNR (busca de uma solução) é via um processo iterativo por lote (Von Zuben, 2000), sendo finalizado, por exemplo, se o erro quadrático médio for menor que 10^{-3} ou após 1000 épocas.

$$J(\varpi) = \frac{1}{2}\sum_{L=1}^{N}\sum_{i=1}^{M}\left(\Theta_{Li} - \tilde{\Theta}_{Li}\right)^2 < \delta \tag{3.39}$$

Onde:

δ	Parâmetro de projeto;
N	Número total de amostras;
M	Número total de saídas na RNR.

O modelo matemático que descreve as saídas da rede neural é dado pela Equação (3.40), tendo a polarização inclusa.

$$\hat{\Theta}_i = \sum_{j=1}^{m+1} v_{ij} z_{ij} = \sum_{j=1}^{m+1} v_{ij} g_e\left(u_{ij}\right) = \sum_{j=1}^{m+1} v_{ij} g_e\left(\sum_{k=1}^{n+1} \omega_{jk} X_k\right) \tag{3.40}$$

Onde:

v_{ij}	j-ésimo peso relativo a i-ésima saída na RNR;
ω_{jk}	j-ésimo peso relativo a k-ésima entrada na RNR ;
g_e	Função de ativação do neurônio na camada intermediária, g_e=a tanh(bu);
m	Número de neurônios na camada intermediária na RNR;
n	Número de padrões;
X_k	k-ésimo padrão na entrada da RNR.

A atualização dos pesos da RNR segue a seguinte regra (Von Zuben *et al.*, 1998):

$$\omega_{n+1} = \omega_n - \alpha\left[\nabla^2 J(\omega_n)\right]^{-1} \nabla J(\omega_n) \tag{3.41}$$

Onde:

n	enésima iteração no processo de aprendizagem;

α Taxa de aprendizagem ($0 < \alpha \leq 1$), cujos valores seguem o algoritmo *gold*;

$\nabla J(\bullet)$ Gradiente

Para execução do processo de treinamento da rede neural, foram pré-definidas uma trajetória para o ZMP, conforme a Figura 3.9 e, pelo emprego do *GAM*, uma marcha foi gerada com os seguintes requisitos adotados:

Comprimento do passo, λ = **0,15** m;

Velocidade horizontal, vs = **0,55** m/s;

Ângulo entre pé o solo, q_0 = 0 rad;

Altura máxima do pé em translado, d = **0,03** m

Taxa de amostragem, DT = **0,001** s.

Nessa fase, optou-se por restringir os movimentos do robô bípede no plano frontal. O padrão de marcha obtido está ilustrado nas figuras que seguem, com amostras a cada **0,01** s para facilitar a classificação das curvas por meio de legendas. De conformidade com o procedimento para obtenção da cinemática inversa, primeiramente, foram especificados os deslocamentos no plano cartesiano para os quadris e pés. Os ângulos relativos obtidos por meio da cinemática inversa, estão ilustrados na Figura 3.14. A cinemática direta foi empregada para computar os movimentos do robô bípede no plano cartesiano, ilustrados nas Figuras 3.11 e 3.12. As acelerações tangenciais foram computadas numericamente (diferenciação numérica central), apresentadas na Figura 3.13.

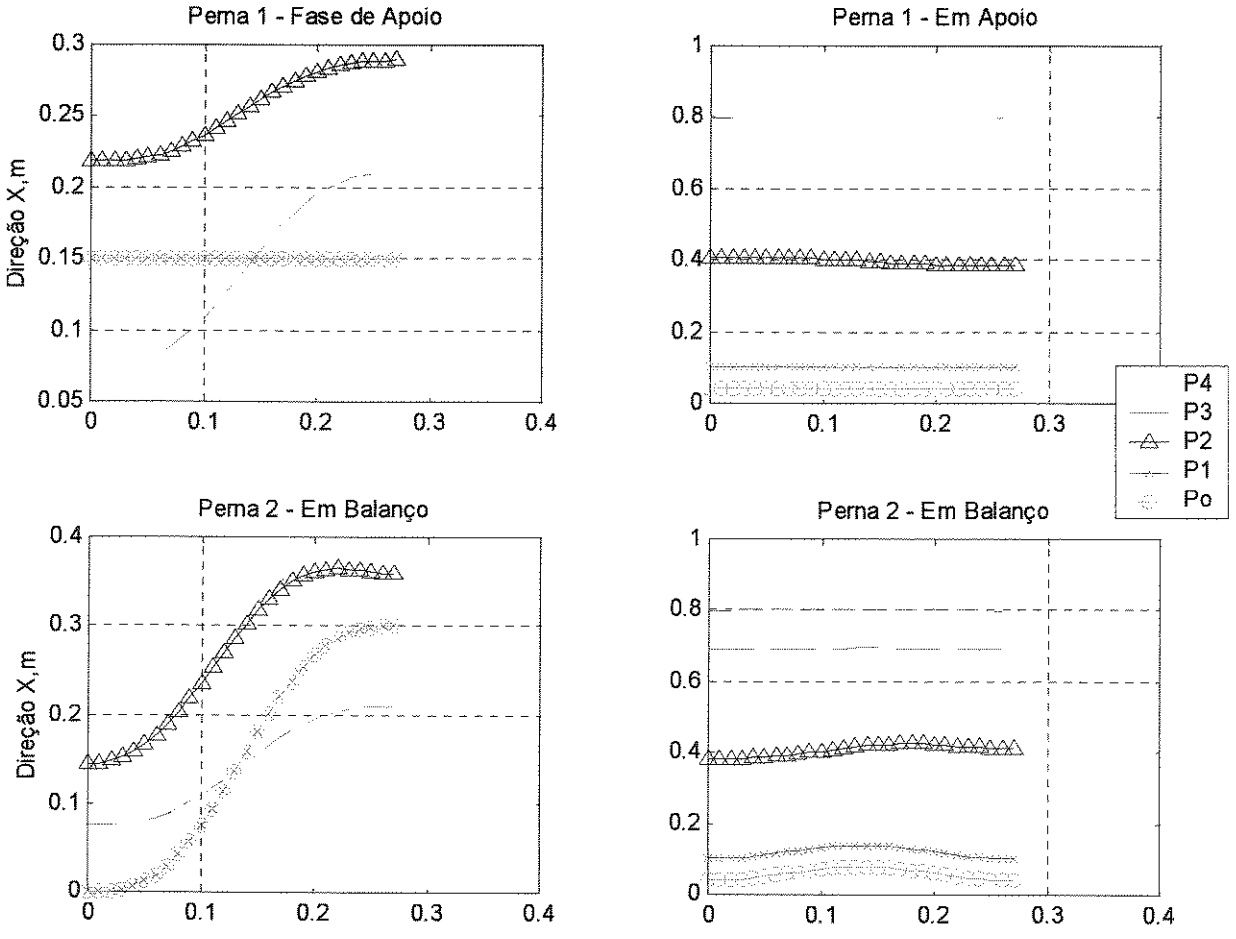


Figura 3.11 – Movimentos das pernas no plano cartesiano, medidos em relação ao SCG.

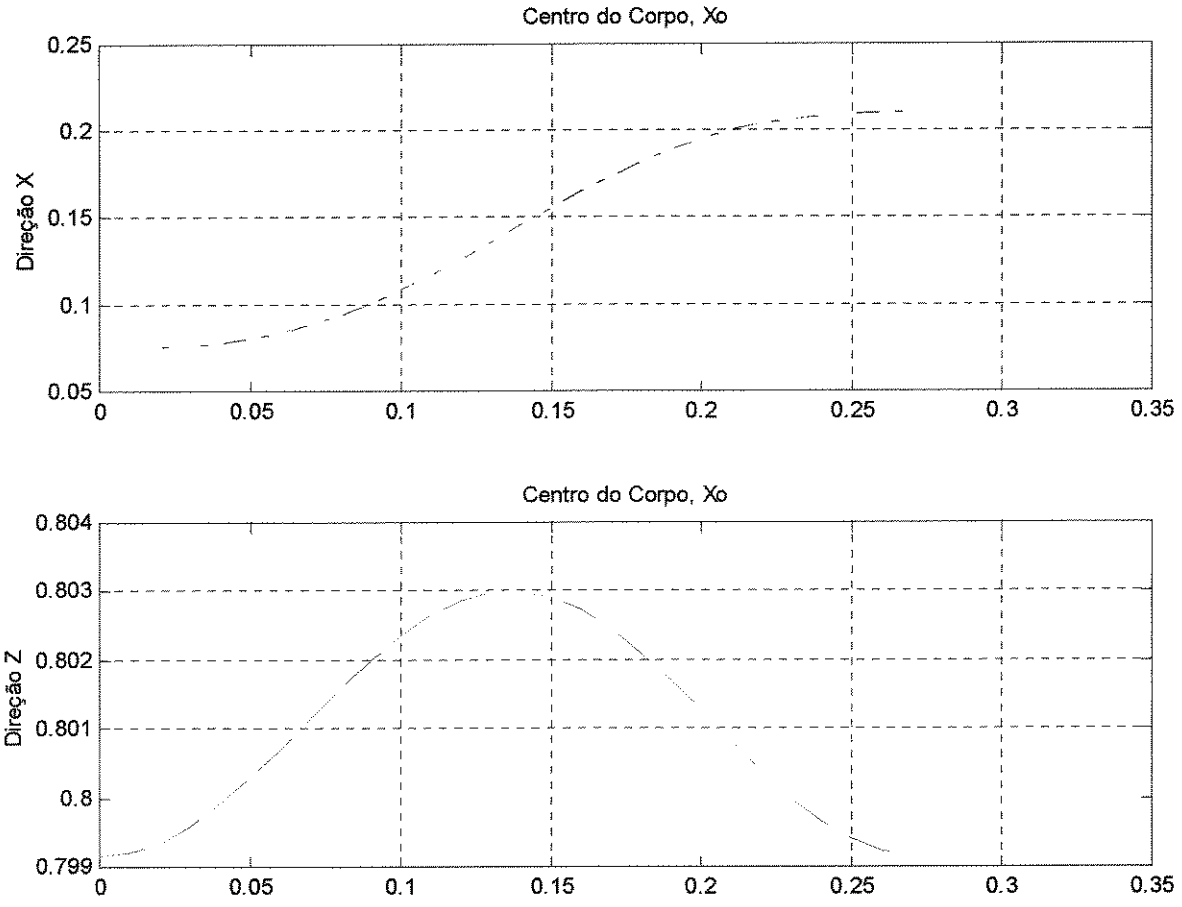


Figura 3.12 – Posições do Centro do Corpo, medidos em relação ao SCG.

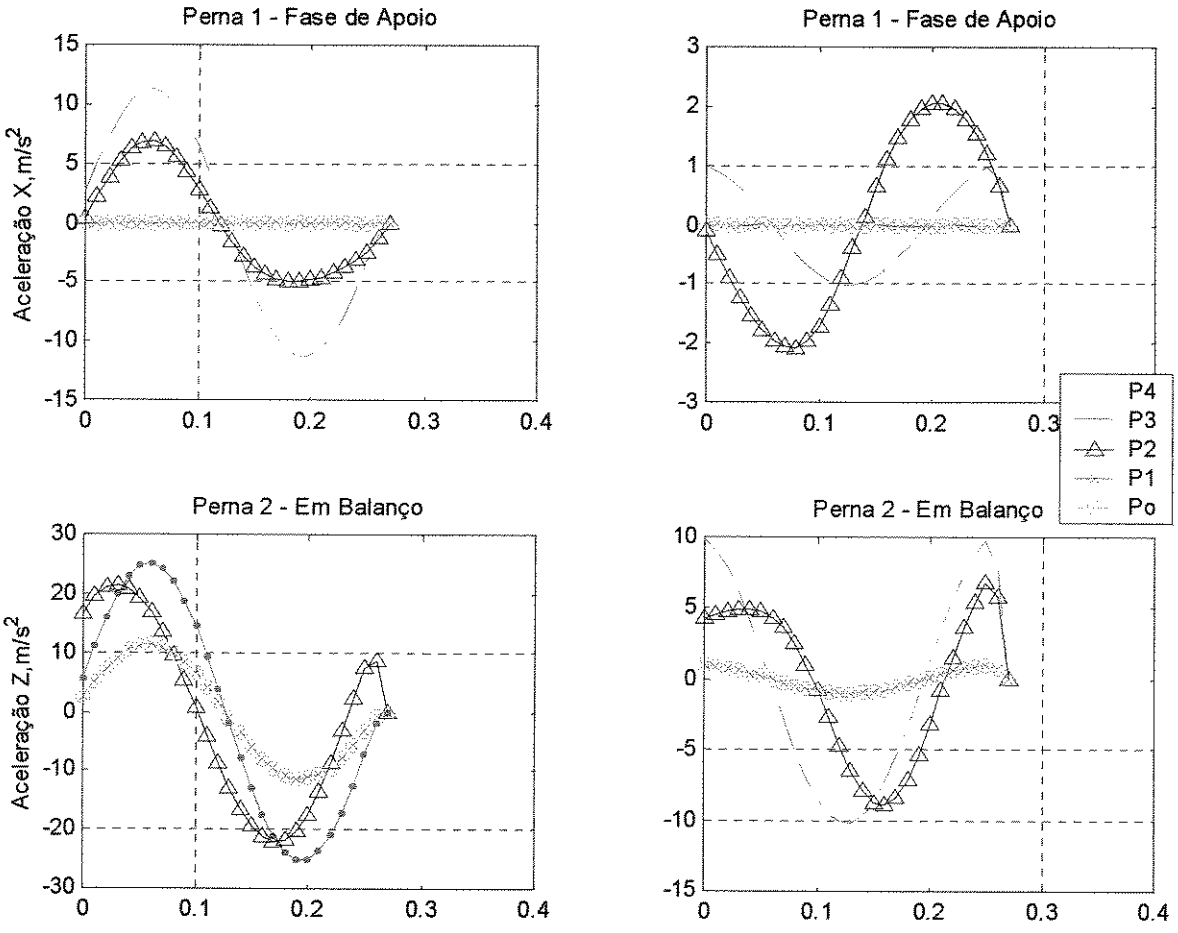


Figura 3.13 – Acelerações Tangenciais no plano cartesiano, medidas em relação ao SCG.

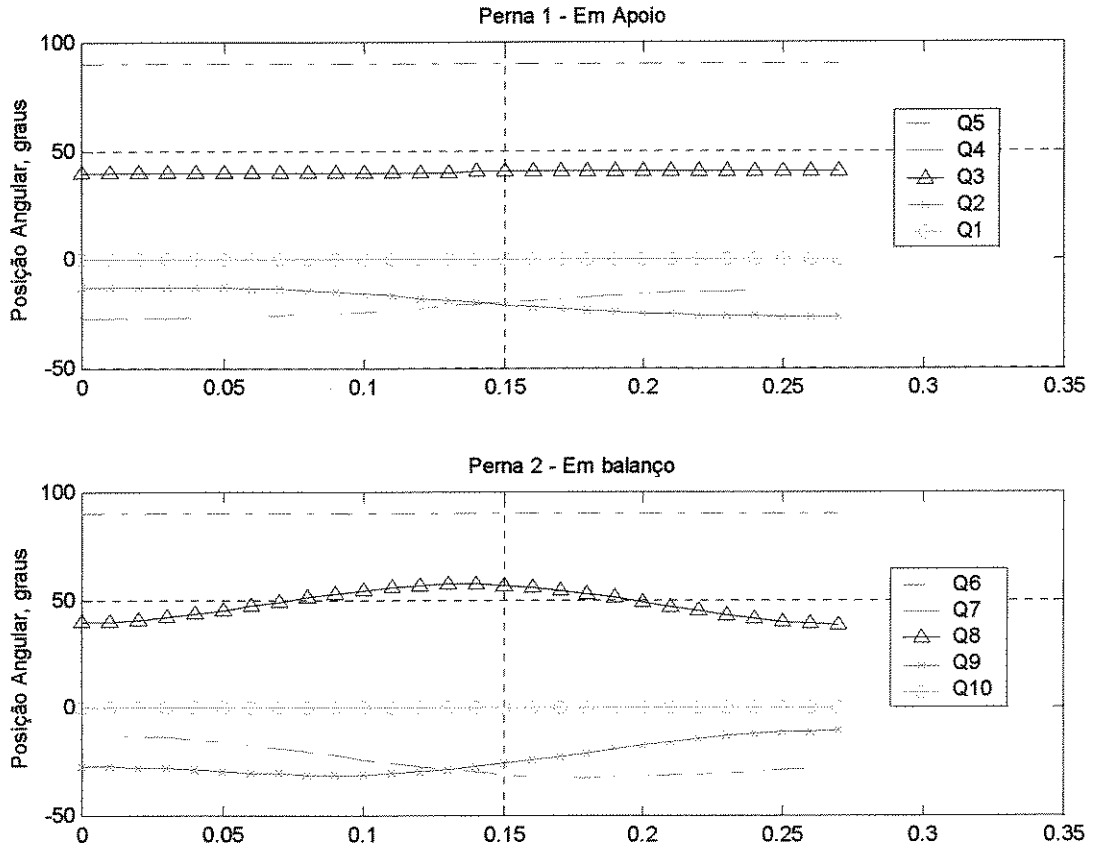


Figura 3.14 – Posições angulares associadas aos membros inferiores do robô bípede.

Em seguida, este padrão de marcha foi empregado para resolver o problema de otimização proposto em (3.38). A função *fmincon* da *Tool Box Otimization* (Matlab®) foi empregada, com a seguinte forma (**Anexo E** – Solução para o Tronco):

```
x = fmincon('fotimo',XO,[],[],[],[],lb,ub,[],options)
```

Os resultados obtidos estão mostrados na Figura 3.15. Os deslocamentos no plano cartesiano para o tronco estão definidos no canto superior esquerdo e as velocidades e acelerações (dispostas à direita) foram computadas numericamente. Os correspondentes ângulos associados ao tronco foram computados via cinemática inversa. No canto inferior direito está o deslocamento para o ZMP estimado (curva em preto) e para o planejado (curva em azul).

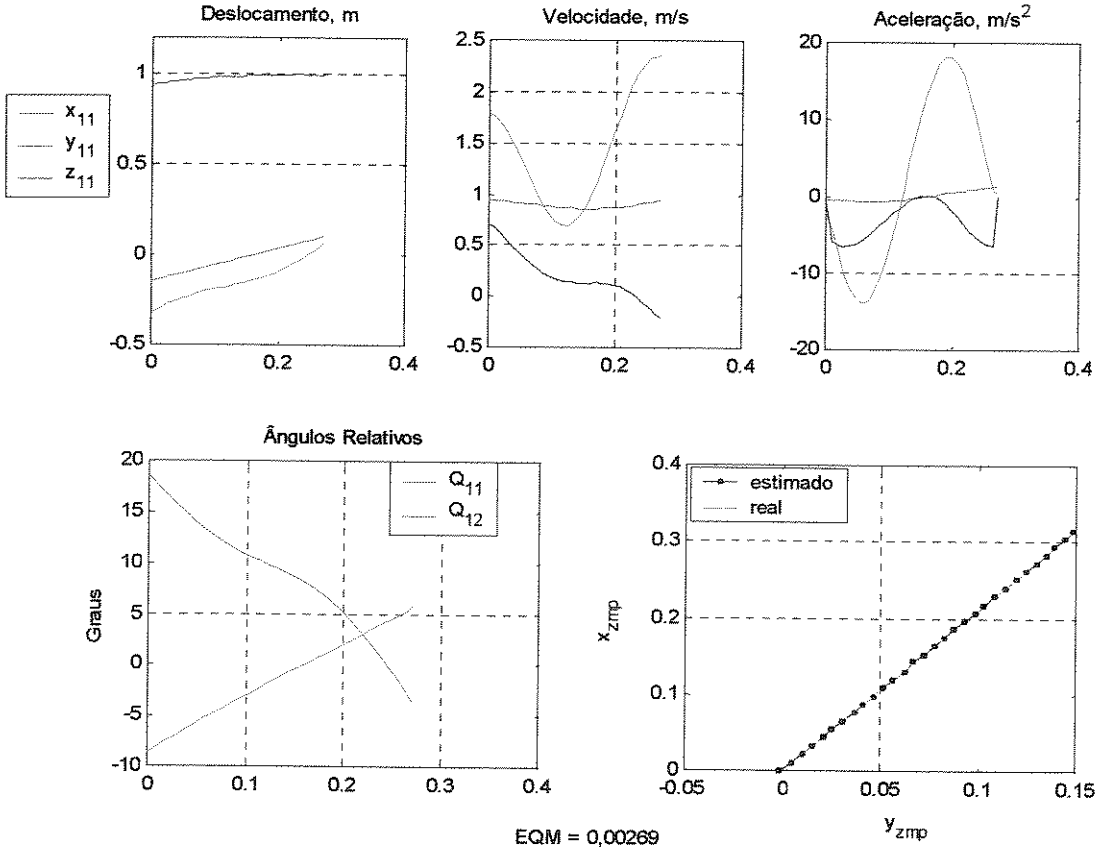


Figura 3.15 – Solução para o problema proposto em (3.38).

Os ângulos foram utilizados como padrões (apresentados a RNR de forma aleatória) para o treinamento da rede neural recorrente. O *modo de treinamento por lote* foi empregado e os pesos, associados à camada intermediária e à camada de saída, foram identificados. A estrutura da RNR e os parâmetros de projeto que apresentaram melhores resultados foram (**Anexo F**):

Número de neurônios na primeira camada intermediária, $m_1 = 20$;

Número de neurônios na segunda camada intermediária, $m_2 = 20$;

Número de neurônios na camada de saída, $n = 2$;

Critério de parada, $\delta =$ **0,0001** rad.

A Figura 3.16 mostra algumas informações relevantes obtidas do treinamento da rede neural. A figura de cima apresenta as normas dos pesos da camada intermediária (w) e de saída (v). A figura de baixo mostra as normas dos gradientes computados a cada época.

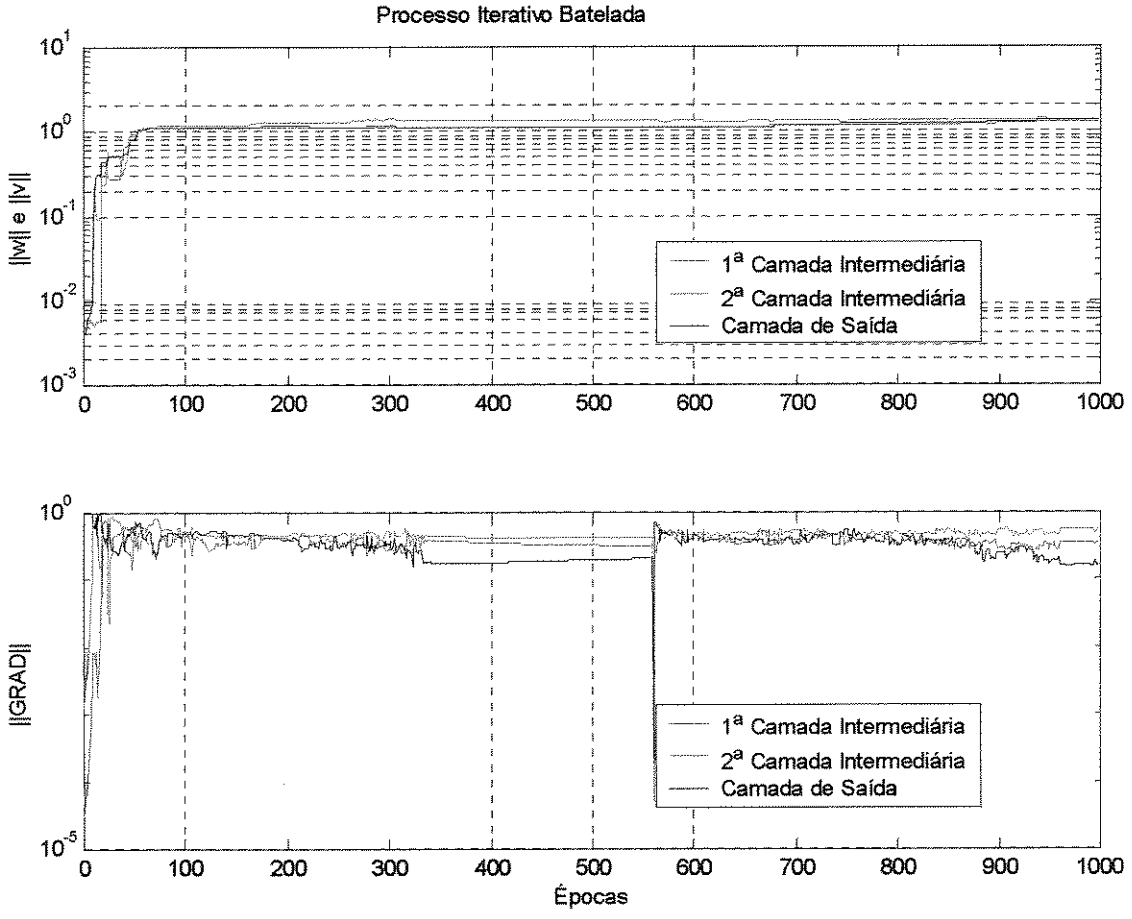
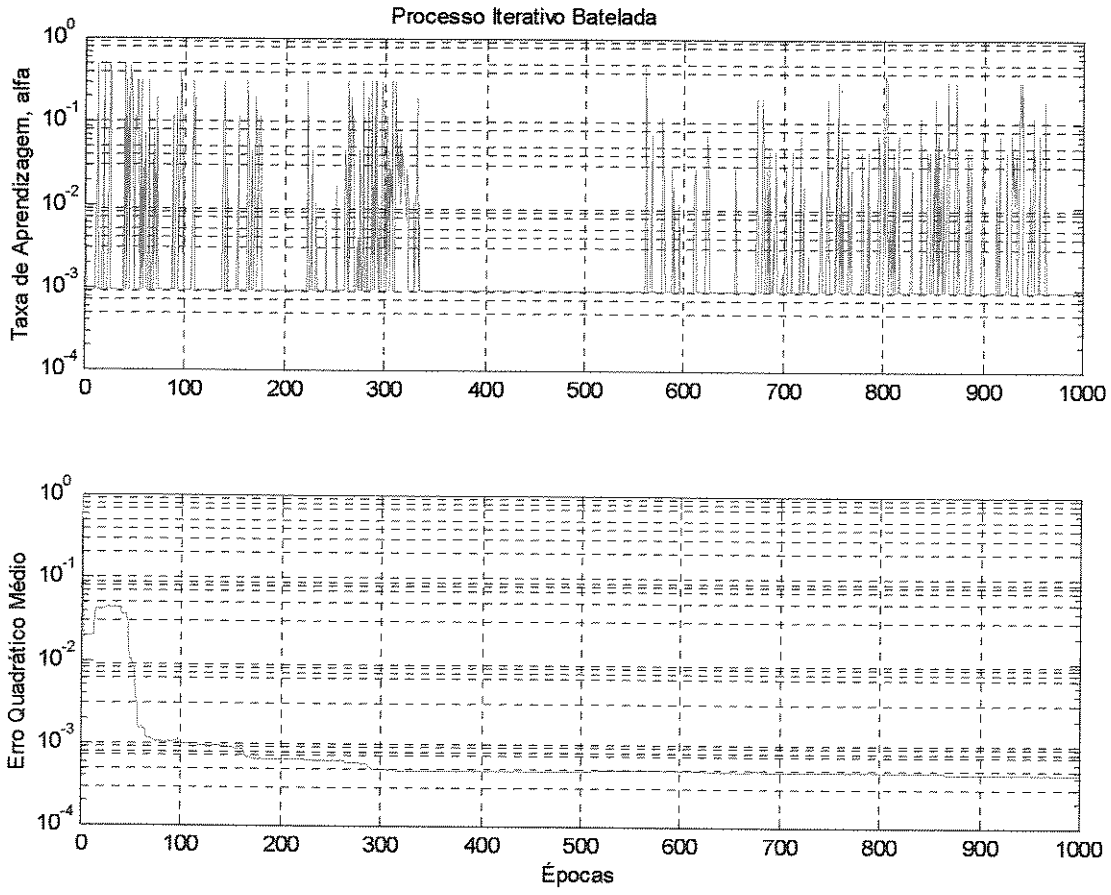


Figura 3.16 – Informações decorrentes do treinamento da rede neural proposta.

A Figura 3.17 mostra as evoluções da taxa de aprendizagem (figura de cima), cujos valores foram determinados pelo algoritmo *gold* (Von Zuben *et al.*, 1998) e do erro quadrático médio (figura de baixo) computado a cada época, cujo valor final é de **0,00045** rad. O tempo de processamento (*cpu time*) foi de 20 minutos (foi utilizado um PC com 512 Mb de memória e processador Pentium 4).



Figuras 3.17 – Variáveis temporais relativas ao movimento do tronco, obtidas no treinamento da rede neural.

Para o teste de generalização da RNR (acima projetada), os padrões de testes (separados a partir dos padrões de treinamento) foram utilizados para excitar a RNR (apresentados seqüencialmente). A Figura 3.18 ilustra o resultado obtido, sendo as curvas em azul e vermelho as trajetórias angulares determinadas pela RNR e as curvas em rosa e preto os padrões de teste.

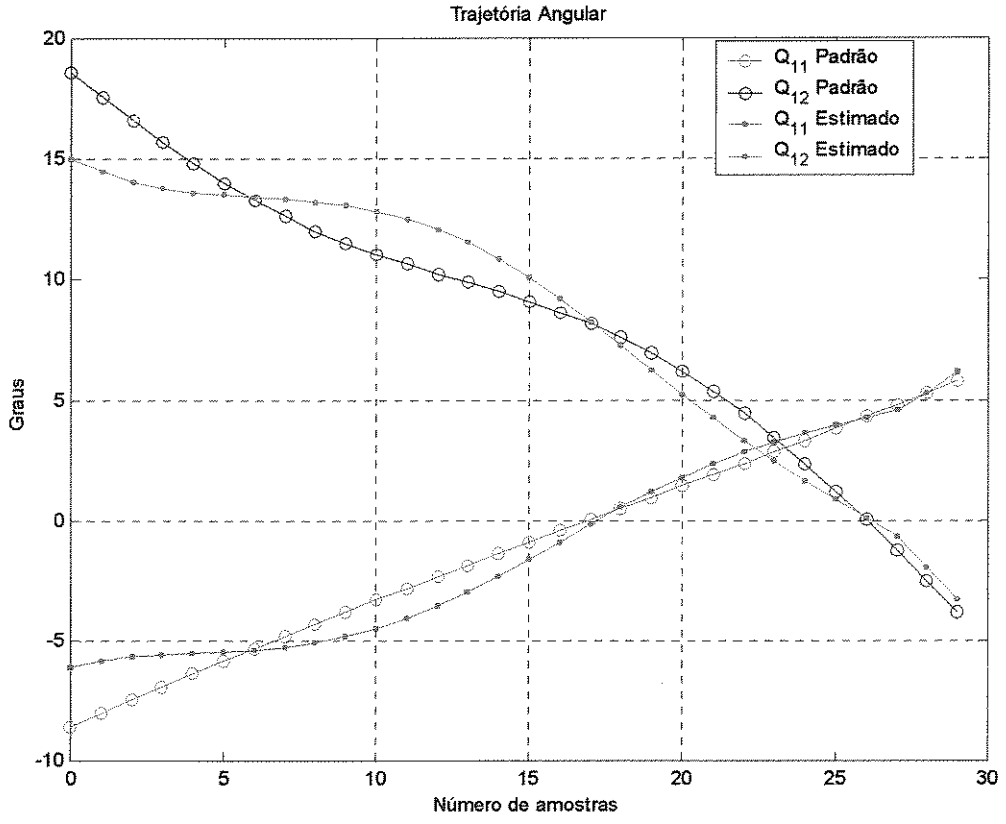


Figura 3.18 – Apresentações das trajetórias planejadas e computadas do par ordenado (x_{zmp},y_{zmp}).

3.4. Gerador Automático de Marcha

O planejador de marchas deverá executar duas tarefas principais: determinar o conjunto de pontos-via e fornecer trajetórias factíveis entre pontos-via, permitindo ao robô bípede caminhar.

O conjunto de pontos-via é determinado por meio da cinemática inversa, fornecendo-se os pontos-via relativos ao pé em balanço e à quinta junta (j_4). Por simplificação, os pontos-via do pé em balanço são determinados por uma função conhecida. Foram propostas duas soluções para os pontos-via da quinta junta. Entretanto, nesse trabalho optou-se por implementar a segunda proposta.

1. São pesquisados em um espaço de soluções parâmetros (ver Equação 3.102) que descrevem a localização da pelve no sentido do movimento e na direção horizontal. A cinemática inversa é computada e o gerador automático de trajetórias (*GAT*) é utilizado para fornecer trajetórias suaves entre pontos-via adjacentes. Em seguida, é verificado o critério de estabilidade postural.

2. Admitem-se funções conhecidas para as trajetórias da pelve nas direções x, y e z. São pesquisados três parâmetros (ver Equação 3.103) associados às funções conhecidas. A trajetória do pé em balanço é função da trajetória da pelve. A cinemática inversa é aplicada para computar as trajetórias angulares associadas às juntas. As velocidades angulares são calculadas utilizando-se matrizes Jacobianos e por meio do *GAT* as acelerações são computadas. E, por fim, o critério de estabilidade postural.

O *GAT* emprega funções *Splines* objetivando fornecer trajetórias com transições suaves entre pontos-via subseqüentes, respeitando restrições de velocidades e acelerações nos pontos-via, e nos pontos inicial e final. As velocidades e as acelerações nos pontos-via intermediários foram especificadas por otimização, utilizando algoritmo genético. O *GAT* foi implementado como mostrado a seguir.

3.4.1. Gerador Automático de Trajetórias

Polinômios de quinta-ordem são empregados para interpolar pontos-via adjacentes. Os coeficientes são calculados impondo-se restrições de continuidades às velocidades e às acelerações entre segmentos subseqüentes. As velocidades e as acelerações iniciais e finais devem ser pré-estabelecidas e aquelas associadas aos pontos-via são computadas pelo seguinte algoritmo (Adade Fº, 1992).

A Equação (3.67) é um polinômio de quinta-ordem associado ao enésimo segmento e ao i-ésimo ponto-via, sendo $\Delta t = t - t_e$.

$$q_j^e(t) = a_0^e + a_1^e \Delta t + a_2^e \Delta t^2 + a_3^e \Delta t^3 + a_4^e \Delta t^4 + a_5^e \Delta t^5 \qquad (3.67)$$

Onde:

e = 1, 1, 2, ... N segmentos;

j j-ésima junta.

A transição entre segmentos subseqüentes ocorre no instante t_e, portanto $t \in (t_e, t_{e+1})$. Para manter passagens suaves pelos pontos-via, as seguintes restrições devem ser avaliadas:

$$q_j^{e-1}(t_i) \equiv q_j^e(t_e) \tag{3.68}$$

$$\dot{q}_j^{e-1}(t_e) \equiv \dot{q}_j^e(t_e) \tag{3.69}$$

$$\ddot{q}_{j1}^e(t_e) \equiv \ddot{q}_j^e(t_e) \tag{3.70}$$

$$\dddot{q}_j^e(t_i) \equiv \dddot{q}_j^e(t_e) \tag{3.71}$$

Levando-se em consideração as Equações (3.68) a (3.71) e tomando-se as derivadas temporais de primeira e segunda-ordem da Equação (3.67), resultam:

$$q_j^e(t_e) \equiv a_0^e \tag{3.72}$$

$$\dot{q}_j^e(t_e) \equiv a_1^e \tag{3.73}$$

$$\ddot{q}_j^e(t_e) \equiv 2a_2^e \tag{3.74}$$

Utilizando-se as Equações (3.72) a (3.74) e a Equação (3.67), seguem:

$$q_j^e(t_{e+1}) - q_j^e(t_e) - \dot{q}_j^e(t_e)\Delta t - \tfrac{1}{2}\ddot{q}_j^e(t_e)\Delta t^2 = a_3^e\Delta t^3 + a_4^e\Delta t^4 + a_5^e\Delta t^5 \tag{3.75}$$

$$\dot{q}_j^e(t_{e+1}) - \dot{q}_j^e(t_e) - \ddot{q}_j^e(t_e)\Delta t - 3a_3^e\Delta t^2 = 4a_4^e\Delta t^3 + 5a_5^e\Delta t^4 \tag{3.76}$$

$$\ddot{q}_j^e(t_{e+1}) - \ddot{q}_j^e(t_e) - 6a_3^e\Delta t = 12a_4^e\Delta t^2 + 20a_5^e\Delta t^3 \tag{3.77}$$

Onde:

$$\Delta t = t_{e+1} - t_e \tag{3.78}$$

Os demais coeficientes associados às Equações (3.75) a (3.77) são determinados pela Equação (3.79), como segue. Conhecendo-se as quantidades das Equações (3.72) a (3.74) e a esquerda das igualdades (3.75) a (3.77), obtém-se:

$$\begin{bmatrix} a_3^e \\ a_4^e \\ a_5^e \end{bmatrix} = \begin{bmatrix} \Delta t^3 & \Delta t^4 & \Delta t^5 \\ 3\Delta t^2 & 4\Delta t^3 & 5\Delta t^4 \\ 6\Delta t & 12\Delta t^2 & 20\Delta t^3 \end{bmatrix}^{-1} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \tag{3.79}$$

Onde:

$$\alpha_1 \equiv q_j^e(t_{e+1}) - q_j^e(t_e) - \dot{q}_j^e(t_e)\Delta t - \tfrac{1}{2}\ddot{q}_j^e(t_e)\Delta t^2 \tag{3.80}$$

$$\alpha_2 \equiv \dot{q}_j^e(t_{e+1}) - \dot{q}_j^e(t_e) - \ddot{q}_j^e(t_e)\Delta t \tag{3.81}$$

$$\alpha_3 \equiv \ddot{q}_j^e(t_{e+1}) - \ddot{q}_j^e(t_e) \tag{3.82}$$

As Equações (3.72) a (3.74) e (3.79) permitem computar os coeficientes do polinômio expresso em (3.75), a partir do conhecimento prévio de posições, velocidades e acelerações em cada ponto-via, inclusos os pontos inicial e meta. Conhecidos os pontos-via (posições), as velocidades e as acelerações podem ser obtidas impondo-se restrições de continuidade em cada ponto-via, resultando um sistema linear de equações (Fu, 1988). Entretanto, objetivando-se

explorar técnicas de otimização, esse problema foi descrito com o formalismo baseado em algoritmos genéticos, apresentado a seguir.

A abordagem aqui apresentada trabalha com os pontos-via associados a cada junta, por vez. Consideram-se M pontos-via e cada indivíduo (cromossomo) contém informações de velocidade e de aceleração associadas a cada ponto-via, como ilustra a Figura 3.24.

$\dot{q}_1$	$\dot{q}_2$	$\dot{q}_3$	-------	$\dot{q}_M$	$\ddot{q}_1$	$\ddot{q}_2$	$\ddot{q}_3$	-------	$\ddot{q}_M$

Figura 3.19 – Informações contidas no indivíduo associado a M pontos-via.

Para avaliar a adequação de cada indivíduo, a Equação (3.67) é avaliada para $t \in (0,t_f)$ e com um tempo de amostragem T. O melhor indivíduo é selecionado com base no funcional (3.83), que deve ser maximizado.

$$\Im_4(x) = \frac{\upsilon}{\delta + \Im_3(x)} \tag{3.83}$$

Onde:

$$\Im_1(x) = \sum_{k=0}^{t_f-1} \left\{ \left[\frac{(\dot{q}_{kT+1} - \dot{q}_{kT})}{\dot{q}_{kT}} \right]^2 + \left[\frac{(\ddot{q}_{kT+1} - \ddot{q}_{kT})}{\ddot{q}_{kT}} \right]^2 \right\} \tag{3.84}$$

$$\Im_2(x) = \sum_{k=0}^{t_f-1} \left[\frac{(\dddot{q}_{kT+1} - \dddot{q}_{kT})}{\dddot{q}_{kT}} \right]^2 \tag{3.85}$$

$$\Im_3(x) = \mu \Im_1(x) + (1-\mu)\Im_2(x) \tag{3.86}$$

$$\mu \in (0,1) \tag{3.87}$$

$$x = f(\dot{q} \quad \ddot{q} \quad \dddot{q}) \tag{3.88}$$

A Equação (3.86) representa as medidas ponderadas de distância entre pontos adjacentes, permitindo detectar descontinuidades. Na Equação (3.83), as escolhas das constantes υ e δ servem para realçar as diferenças entre indivíduos e para evitar divisão por zero, respectivamente.

O *Algoritmo 3.9: Gerador Automático de Trajetória entre Pontos Adjacentes* apresenta as principais etapas pertinentes ao gerador automático de trajetórias (*GAT*), cuja implementação encontra-se no **Anexo G**.

Algoritmo 3.9: Gerador Automático de Trajetória entre Pontos Adjacentes

1. Determinar a velocidade e a aceleração iniciais e finais, associadas à cada junta;
2. Determinar a faixa de valores admissível, associada à cada junta;
3. Determinar os instantes de transições (te);
4. Determinar as quantidades de pontos-via (M);
5. Determinar os valores para as constantes μ, υ, δ, p_m e p_c;
6. Determinar as quantidades de gerações (QG) e de indivíduos (QI);
7. Arbitrar valores iniciais para os QI indivíduos; fazer NQG = 1;
8. Computar o funcional (3.83);
9. Escalonar os indivíduos;
10. Escolher os melhores indivíduos que participarão dos processos evolutivos;
11. Aplicar os processos evolutivos aos melhores indivíduos;
12. Manter o melhor indivíduo da geração anterior na geração posterior;
13. Verificar se NQG $\leq$ QG; se positivo fazer NQG = NQG+1 e retornar ao passo 8;
14. Escolher o melhor indivíduo da geração atual.

Para exemplificar e validar a implementação do *GAT*, sejam os seguintes dados de entrada (passos de 1 a 5):

- ✓ Velocidades e acelerações iniciais nulas;
- ✓ Faixa de valores admissível, range = [-180 180 -360 360];

- ✓ Instantes de transições, te = [0 0,5 1 1,5 2 2,5 3];
- ✓ Quantidade de pontos-via, M = 7;
- ✓ $\mu = ½$, $\upsilon = 1$, $\delta = 10^{-17}$, $p_m = 0{,}013$ e $p_c = 0{,}62$;
- ✓ Quantidade de gerações, QG = 10;
- ✓ Quantidade de indivíduos, QI = 50;

A Figura 3.20 mostra as trajetórias originais para as posições, velocidade e acelerações angulares, considerando-se apenas duas juntas.

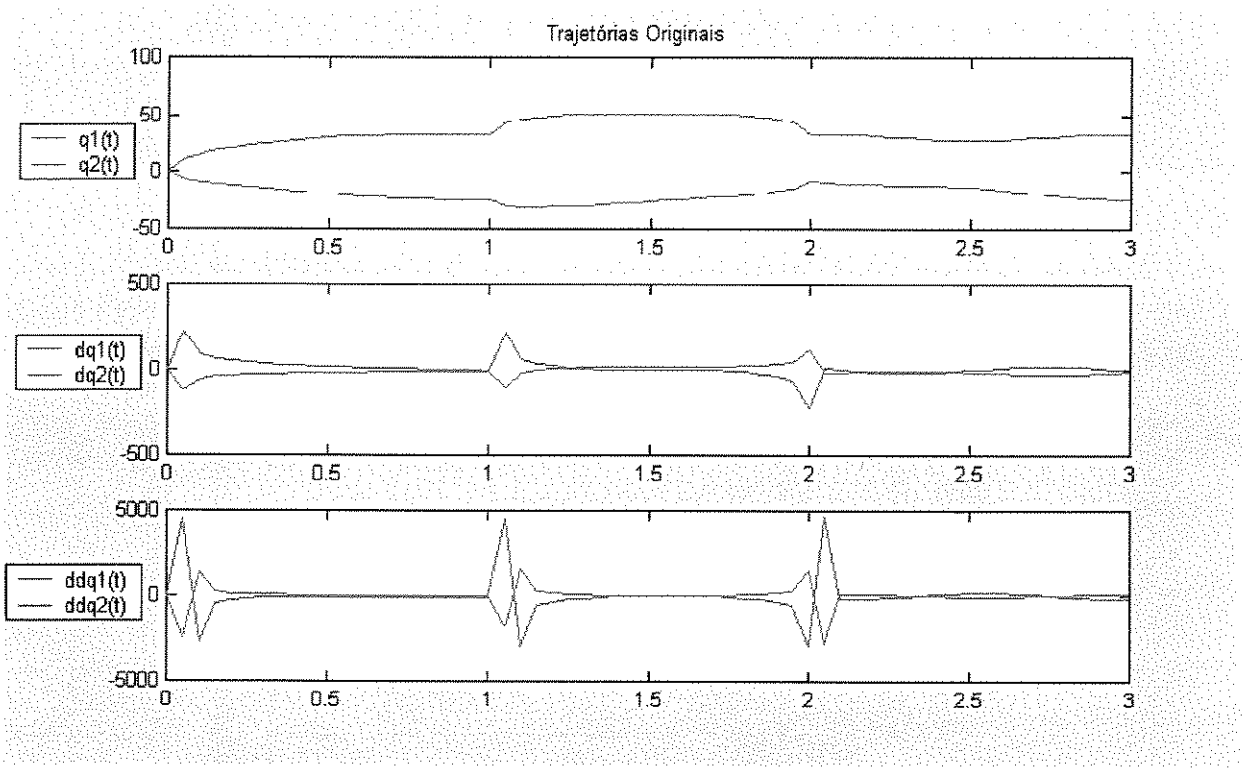


Figura 3.20 – Trajetórias originais para as posições (em cima), velocidades (no meio) e acelerações angulares (em baixo).

Para o emprego do GAT, os pontos-via associados aos instantes de transições foram selecionados das curvas de posições angulares. A Figura 3.21 mostra o valor do funcional (Equação 3.85) relacionado ao melhor indivíduo em cada geração, permitindo verificar que são necessárias poucas gerações (cerca de cinco) para que uma solução seja encontrada. Mas deve-se lembrar que o gerador automático de marcha (*GAM*) utiliza dois módulos: o gerador automático

de trajetórias (*GAT*) e o gerador automático de posturas (descrito mais adiante). O que tornará o custo computacional maior.

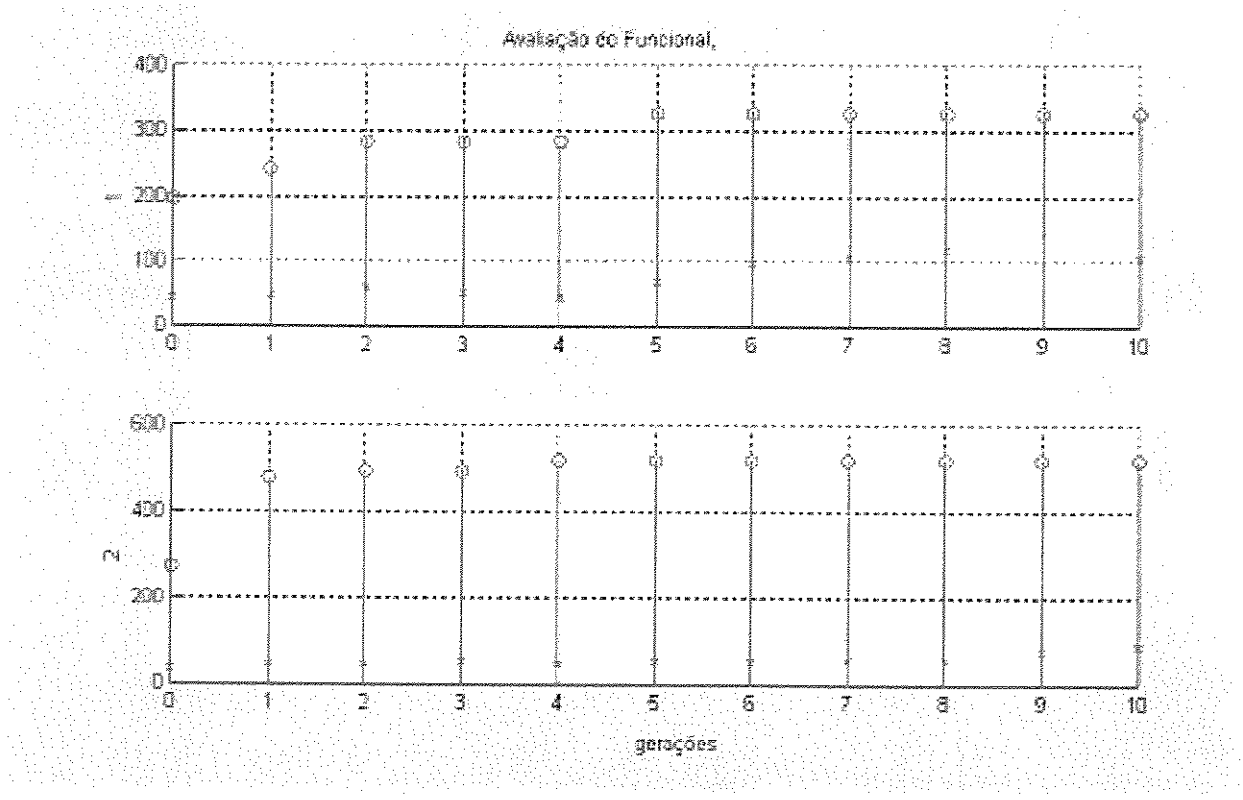


Figura 3.21 – Valor do funcional (3.85) do melhor indivíduo em cada geração.

A Figura 3.22 exemplifica as trajetórias de posições, velocidades e acelerações angulares obtidas a partir do emprego do *GAT*. Os aspectos de suavidades nos instantes de transições ficaram perfeitamente caracterizados.

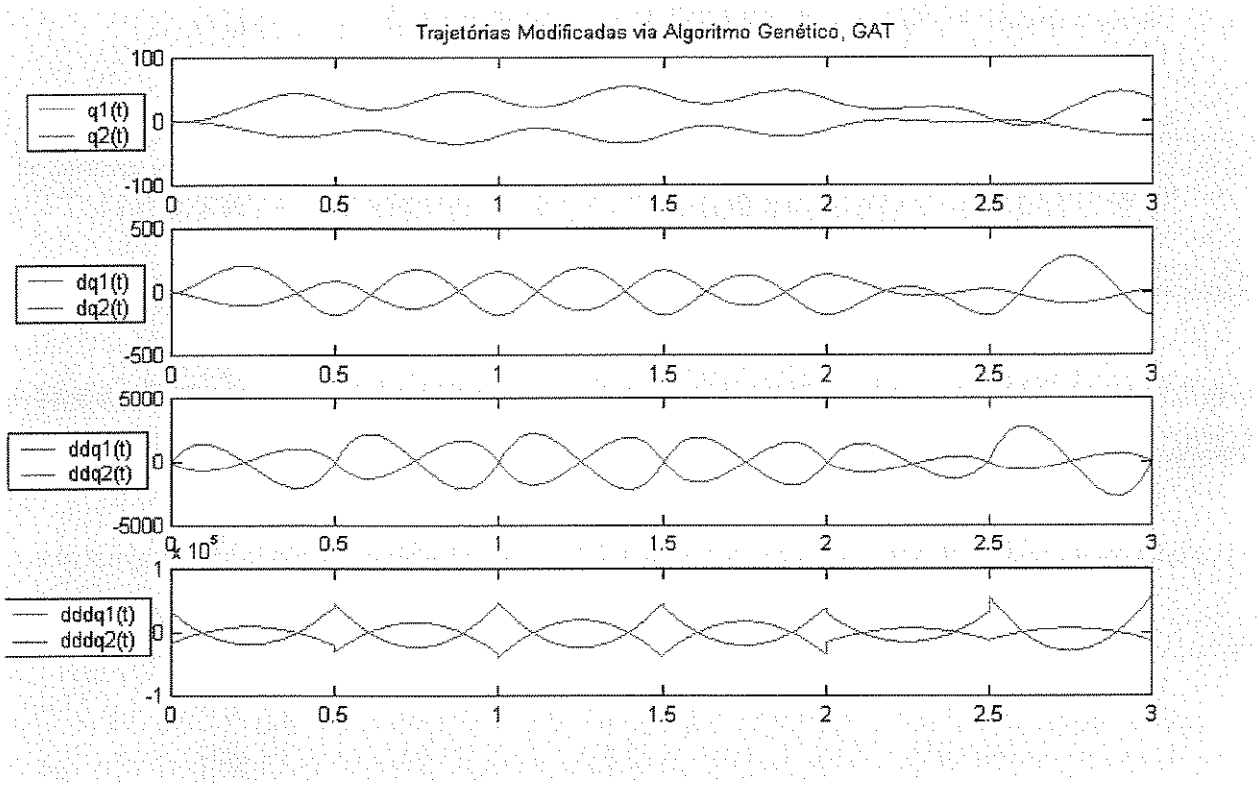


Figura 3.22 – Trajetórias modificadas por meio do GAT.

Além das informações sobre os coeficientes relativos à Equação (3.69), as informações de velocidades e acelerações definidas aos pontos-via podem também ser obtidas conforme ilustram a Tabelas 3.1 (velocidades nos pontos-via) e Tabela 3.2 (acelerações no ponto-via).

Tabela 3.1 – Velocidades associadas a cada ponto-via (graus/s)

j_1	0	85,368	159,017	169,464	134,455	21,8497	16,1588
j_2	0	-180,000	-180,000	-180,000	-179,296	-179,994	-180,000

Tabela 3.2 – Acelerações associadas a cada ponto-via (graus/s^2)

j_1	0	59,9348	44,7069	100,5271	-36,1446	-98,6263	46,6539
j_2	0	-49,7563	-23,2700	-113,7552	-105,1206	132,2096	-57,6078

3.4.2. Planejador de Marcha

O objetivo básico do planejador de marcha é de fornecer conjuntos de pontos-via que satisfaçam à condição de momento nulo (*zmp*), conhecidas *a priori* certas posições básicas. Cada conjunto de pontos-via descreve uma postura específica, cujas sucessões permitem ao robô bípede reproduzir um certo modo de andar. O planejador de marcha está implementado no espaço cartesiano. Assim a cinemática inversa é utilizada para obter o conjunto de ângulos para cada conjunto de pontos-via, formando os conjuntos de pontos-via no espaço das juntas. A seguir, serão discutidos o projeto do planejador de marcha e a correspondente solução desenvolvida.

De acordo com o segundo capítulo, a cinemática inversa é computada se forem conhecidas as quantidades (x_i,R_i) e (x_o,R_o). Admite-se que a marcha é iniciada com o pé esquerdo, tendo o pé direito como base de apoio. A Figura 3.23 mostra uma seqüência de *posturas básicas* (Huang *et al.*, 2001), onde estão definidos: o ângulo $\mathbf{q_o}$ (inclinação do pé em relação ao solo), o comprimento do passo λ, o par ordenado que define a altura máxima permitida para $\mathbf{x_i}$ (L,d). Também são apresentados: o sistema coordenado global (SCG), as distâncias do quadril medida em relação ao sistema global na direção do movimento, $\mathbf{x_{ed}}$ e $\mathbf{x_{ef}}$. A partir da postura básica E admite-se que o robô bípede deixe a fase transitória, passando a vigorar a fase permanente (ou cíclica). Nessa fase, o ponto $\mathbf{x_i}$ percorrerá o dobro do comprimento do passo, denominado por passada.

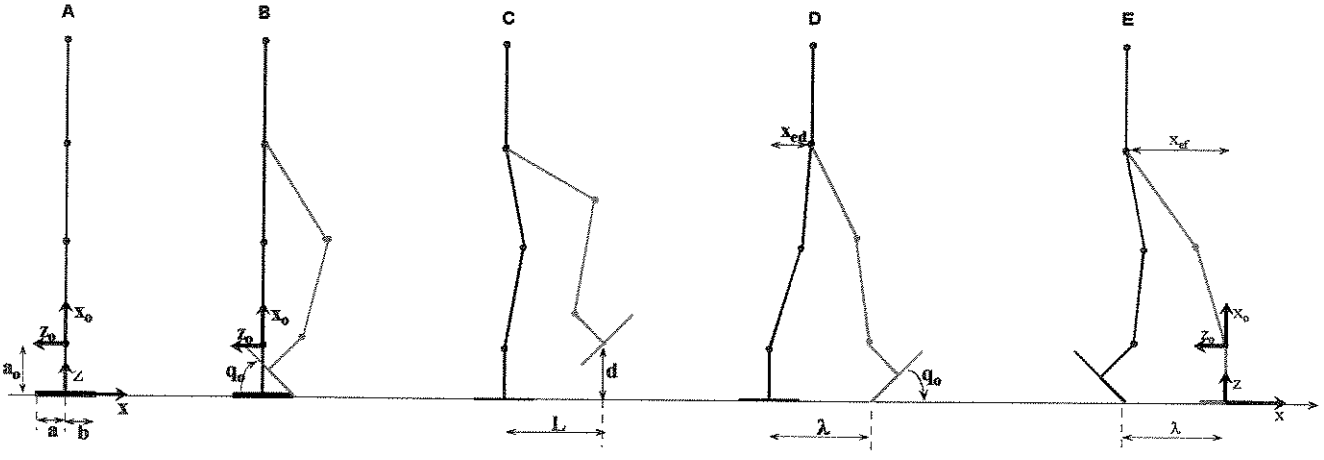


Figura 3.23 – Procedimento da marcha, posturas básicas.

Na fase inicial, o pé esquerdo sofre uma inclinação de $\mathbf{q_o}$ rad em $\mathbf{t_1}$ s. Na seqüência o mesmo se move à frente, cumprindo uma distância de λ m em $\mathbf{t_2}$ s, pousando no solo com um ângulo de $-\mathbf{q_o}$ rad. Então, a inclinação do pé esquerdo em relação ao solo se anula enquanto o pé direito sofre uma inclinação de $\mathbf{q_o}$ rad, durante $\mathbf{t_1}$ s. O pé direito se move à frente cumprindo uma distância de 2λ m, em $\mathbf{t_2}$ s. O ciclo repete-se por k passos. A postura básica C antecede a postura básica D e indica a altura máxima permitida para x_i e o correspondente valor na abscissa. Esse parâmetro pode ser utilizado para evitar a colisão com obstáculos.

Com os rótulos das posturas básicas ilustradas na Figura 3.20, é possível esquematizar o processo de marcha, escolhendo-se a seguinte seqüência, como exemplo:

A – B – C – D –	E – C – D – E – C – D – E –	B – A
Fase transitória	*Fase permanente*	*Fase final*

Admite-se que cada postura básica avante descreve uma *configuração espacial meta* e aquela que a precede descreve uma *configuração espacial inicial*. Para efetivar a marcha é necessário determinar configurações espaciais que permitam a transferência da configuração inicial à meta. Para obter suavidade aos movimentos, é necessário que as posições, velocidades e acelerações entre posturas básicas sucessivas sejam contínuas. Pode-se também impor que a velocidade e aceleração no instante do impacto entre o pé em balanço e o solo (postura básica D) sejam nulas. A seguir são apresentados o equacionamento dessas questões, com as devidas restrições.

A Figura 3.24 ilustra a postura básica D no plano sagital, enfatizando-se as coordenadas de $\mathbf{x_2}$ em relação ao sistema inercial global (Silva *et al.*, 2001). As demais posturas básicas podem ser analisadas por analogia.

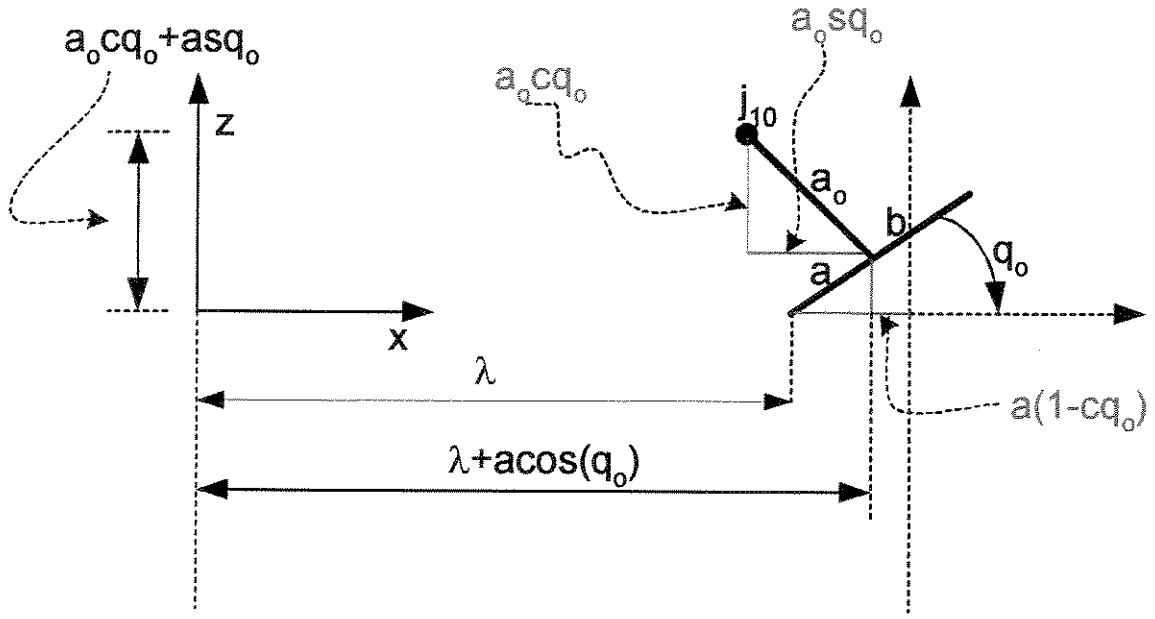


Figura 3.24 – Localização cartesiana da décima junta em relação ao sistema global.

Com base na Figura 3.24, a Equação (3.90) determina a localização espacial de x_2, em relação ao sistema inercial global, para os instantes de tempo correspondentes às posturas básicas.

$$\chi_2 = \{x_2(t) \quad y_2(t) \quad z_2(t)\} \tag{3.90}$$

Onde:

$$x_2(t) = \begin{cases} 0 & ,A \quad ,t = 0 \\ b(1-cq_o) & ,B \quad ,t = t_1 \\ H & ,C \quad ,t = t_2 \\ \lambda + acq_o & ,D \quad ,t = t_3 \\ -(\lambda + bcq_o) & ,E \quad ,t = t_4 \end{cases} \tag{3.91}$$

$$z_2(t)=\begin{cases} 0 & ,A \quad ,t=0 \\ bsq_o & ,B \quad ,t=t_1 \\ L & ,C \quad ,t=t_2 \\ asq_o & ,D \quad ,t=t_3 \\ bsq_o & ,E \quad ,t=t_4 \end{cases} \tag{3.92}$$

$$y_2(t)=a_5 \qquad \forall t \tag{3.93}$$

Em relação às Equações (3.91) e (3.92) e para a postura básica D, as derivadas de primeira e de segunda ordem em relação ao tempo devem ser nulas. Da mesma postura, admitindo-se que o ângulo q_o seja o valor final desejado, resultam[3]:

$$\alpha(t)=\begin{cases} \in(0,q_o) & ,A \rightarrow B \vee E & ,t \in (0,t_1) \\ \in(-q_o,0) & ,B \rightarrow A \vee D \rightarrow E & ,t \in (0,t_1) \end{cases} \tag{3.94}$$

$$\dot{\alpha} \equiv 0 \quad \text{se} \quad D \rightarrow E \tag{3.95}$$

Outro ponto importante para o cômputo da cinemática inversa é a localização de j_4 em relação ao sistema global, descrita pela Equação (3.96).

$$\chi_4 = \{x_4(t) \quad y_4(t) \quad z_4(t)\} \tag{3.96}$$

Considera-se que nas transições entre as posturas, a altura poderá variar em torno de γ_z por cento de sua altura total, conforme Equação (3.97).

$$z_4(t)=\gamma_z \times \sum_{k=0}^{4} a_k \tag{3.97}$$

[3] Os símbolos $\rightarrow$ e $\vee$ designam **implicação** e o operador lógico **ou**.

A coordenada $y_5(t)$ poderá variar conforme a Equação (3.98), sendo γ_y um valor percentual. A coordenada $x_5(t)$ é descrita pela Equação (3.99).

$$y_5(t) = \gamma_y \times a_5 \tag{3.98}$$

$$x_5(t) = \begin{cases} 0 & ,A \vee B \\ L/2 & ,C \\ x_{ed} & ,D \\ x_{ef} & ,E \end{cases} \qquad 0{,}0 < x_{ed} \leq 0{,}5\lambda \tag{3.99}$$

As Equações (3.90) e (3.94) a (3.99) estabelecem os valores para as coordenadas de x_2 e do quadril quando o robô bípede estiver em certas posições básicas. Também é necessário considerar a trajetória para o ponto $x_2(t)$, cuja trajetória no plano sagital pode ser determinada a partir dos pontos-via inicial (x_1,y_1,a_5), intermediário (H,L,a_5) e final (x_3,y_3,a_5), conforme a Figura 3.25, que apresenta também pontos-via selecionados conforme Equação (3.100) e os correspondentes instantes de ocorrência.

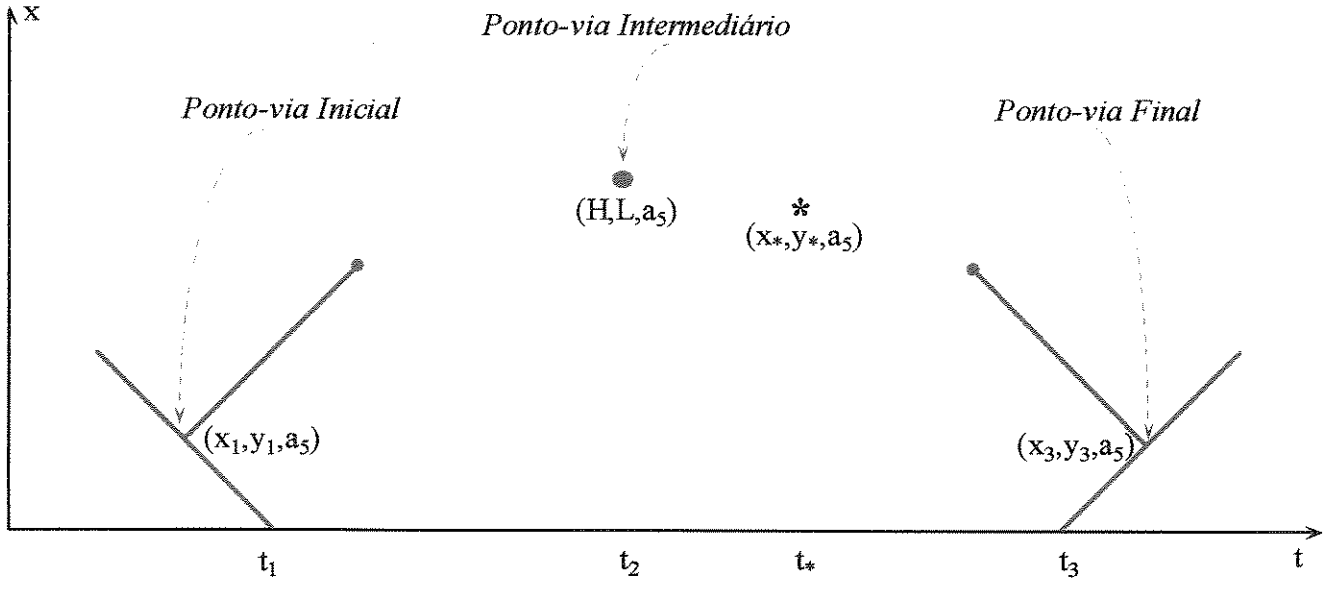


Figura 3.25 – Pontos-via de j_{10} (tornozelo do pé em balanço), no plano sagital.

$$x_* = \begin{cases} \in [x_1, H] & \text{se} \quad t_1 < t < t_2 \\ \in [x_3, H] & \text{se} \quad t_2 < t < t_3 \end{cases} \tag{3.100a}$$

$$y_* = \begin{cases} \in [y_1, L] & \text{se} \quad t_1 < t < t_2 \\ \in [y_3, L] & \text{se} \quad t_2 < t < t_3 \end{cases} \tag{3.100b}$$

$$z_* = a_5 \qquad \forall t \tag{3.100c}$$

As posturas básicas aqui apresentadas permitem computar a cinemática inversa desenvolvida no segundo capítulo. Essa abordagem requer a determinação de um conjunto de parâmetros (definidos pelas Equações 3.102 ou 3.103) que devem ser escolhidos objetivando manter o balanço dinâmico. O que suscita a seguinte questão: *Quais são os valores do conjunto de parâmetros que maximiza o funcional (3.101)?*

$$\Im(\Theta) = \frac{1}{2}\left(e^{-\alpha\sqrt{\frac{1}{N}\sum_{t=0}^{N} T_x(i)^2}} + e^{-\alpha\sqrt{\frac{1}{N}\sum_{t=0}^{N} T_y(i)^2}} \right) \tag{3.101}$$

Onde:

T_x e T_y são definidos pela Equação (3.28);

α: Parâmetro de projeto.

A solução para o problema de otimização proposto em (3.101) admite pelo menos duas soluções. A primeira é o conjunto de parâmetros definido pela Equação (3.102), que permitem caracterizar as posturas básicas ilustradas na Figura 3.20.

$$\Theta = \{L \quad H \quad \lambda \quad X_{ed} \quad X_{ef} \quad \gamma_x \quad \gamma_y \quad \gamma_z\} \tag{3.102}$$

Esses parâmetros podem ser pesquisados em um espaço de soluções por meio de algoritmos genéticos. Após, para cada individuo, a cinemática inversa é computada e o gerador automático

de trajetórias (*GAT*) é utilizado para fornecer trajetórias para as posições, velocidades e acelerações angulares. Na seqüência é verificado o critério de estabilidade postural, avaliando-se a Equação (3.101).

A segunda solução admite um conjunto de parâmetros definido pela Equação (3.103), que está relacionado com as equações que seguem.

$$\Theta = \{\gamma_x \quad \gamma_y \quad \gamma_z\} \tag{3.103}$$

Sejam, primeiramente, as seguintes definições relacionadas à velocidade de trajeto:

ν Velocidade de trajeto na direção horizontal, m/s;

λ Comprimento do passo, m;

$\delta = \frac{\lambda}{\nu}$ Período, p/passo;

$f = \frac{1}{\delta}$ Freqüência, 1/s.

Para a transição da postura básica B para a D, a trajetória do pé em balanço é caracterizada pelas Equações (3.104) a (3.106). A contagem do tempo é sempre reiniciada quando a postura básica B é atingida.

$$x_2(t) = x_2(0) + [\nu + x_2(t_3)]\left[t - \frac{\sin(2\pi ft)}{2\pi f}\right] \tag{3.104}$$

$$y_2(t) = a_5 \tag{3.105}$$

$$z_2(t) = z_2(0) + \tfrac{H}{2}[1 - \cos(2\pi ft)] \tag{3.106}$$

As coordenadas do quadril são descritas como função das Equações (3.104) a (3.106):

$$x_4(t) = \gamma_x x_2(t) + x_4(0) \tag{3.107}$$

$$y_4(t) = \gamma_y x_2(t) \tag{3.108}$$

$$z_4(t) = \gamma_z z_2(t) - H + z_4(0) \tag{3.109}$$

Essas mesmas equações podem também descrever as transições entre as posturas básicas de E para D, quando se inicia o ciclo da marcha.

Utilizando as Equações (3.104) a (3.109), pode-se computar a cinemática inversa, obtendo-se as trajetórias para as posições angulares. As velocidades angulares podem ser computadas utilizando as informações de velocidades do pé em balanço e do quadril, cuja velocidade pode ser computada por diferenciação numérica. A Equação (3.110) computa por cinemática direta a localização de p_{10} ("real") em relação ao sistema global.

$$p_{10} = \left[x(\theta) \quad y(\theta) \quad z(\theta)\right]^T \equiv {}^{SG}T_{10}\,\hat{i}_4 \tag{3.110}$$

A velocidade é determinada, aplicando a derivada de primeira ordem à Equação (3.110).

$$\dot{p}_{10} = \begin{bmatrix} \frac{\partial x(\theta)}{\partial \theta_1} & \cdots & \frac{\partial x(\theta)}{\partial \theta_{10}} \\ \frac{\partial y(\theta)}{\partial \theta_1} & \cdots & \frac{\partial y(\theta)}{\partial \theta_{10}} \\ \frac{\partial z(\theta)}{\partial \theta_1} & \cdots & \frac{\partial z(\theta)}{\partial \theta_{10}} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_{10} \end{bmatrix} \equiv J_1(\theta)\dot{\theta} \tag{3.111}$$

A partir da Equação (3.111), uma solução para o problema inverso é obtida (Noble, 1969).

$$\dot{\theta} = J^+ \dot{p}_{10} + \left(I - J^+ J_1\right)\dot{\theta}_d \tag{3.112}$$

Onde:

$$J^{+} = J_1^T \left(J_1 J_1^T \right)^{-1} \tag{3.113}$$

θ_d Vetor de trajetória arbitrária (3.114)

A trajetória arbitrária pode ser utilizada para permitir ao robô bípede realizar uma segunda tarefa. Sendo conhecidos o vetor posição do quadril e o seu respectivo vetor de velocidades tangenciais, podem ser escritas as seguintes equações:

$$p_4 = \left[x(\theta) \quad y(\theta) \quad z(\theta) \right]^T \equiv {}^{SG}T_4 \, \hat{i}_4 \tag{3.115}$$

$$\dot{p}_5 = J_2(\theta)\dot{\theta} \tag{3.116}$$

A Equação (3.117) permite escolher uma trajetória que minimiza a norma quadrática e evita singularidades.

$$\dot{\theta}_d = \gamma J_2^{+} \dot{p}_5 + (1-\gamma) \frac{\partial \sqrt{\det\left(J_2 J_2{}^T \right)}}{\partial \theta} \qquad 0 \leq \gamma \leq 1 \tag{3.117}$$

Onde:

$$J_2^{+} = J_2^T \left(J_2 J_2^T \right)^{-1} \tag{3.115}$$

$$\gamma = \begin{cases} 0 & \text{se} \quad \det\left(JJ^T \right) = 0 \\ ½ & \text{caso contrário} \end{cases} \tag{3.116}$$

Conhecidas as velocidades angulares, pode-se empregar o *GAT* para obter as correspondentes acelerações angulares e por fim computar a Equação (3.101).

Ambas propostas podem utilizar a busca de parâmetros por algoritmo genético, cujos indivíduos podem ser descritos por valores pertencentes ao conjunto dos números reais. A

primeira solução está descrita na Figura 3.26. A segunda solução está descrita pelo *Algoritmo 3.10: Gerador Automático de Posturas (GAP)* .

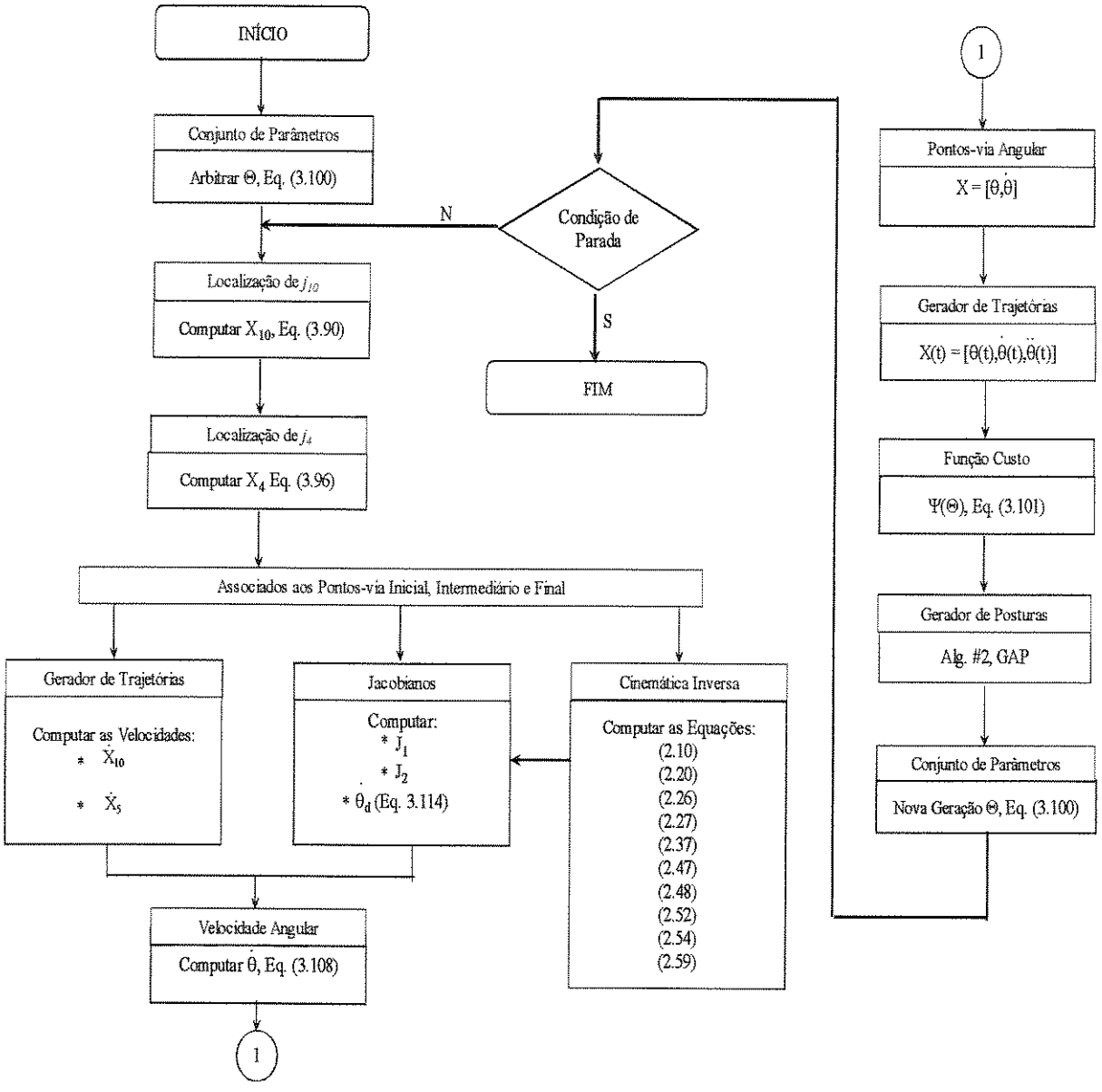


Figura 3.26 – Fluxograma do Gerador Automático de Marcha

Algoritmo 3.10: Gerador Automático de Posturas (GAP-2)

1. Determinar as quantidades de gerações (QG) e de indivíduos (QI);
2. Arbitrar valores iniciais para os QI indivíduos; fazer NQG = 1;
3. Arbitre o tempo total da passada t_3 (segundos) e aplique a divisão áurea para computar os tempos associados às etapas intermediárias, como ilustra a Fig, (3.8);
4. Aplique a divisão áurea aos parâmetros λ e h para determinar a localização dos pontos-via para o pé em balanço, no plano sagital;
5. Aplique a cinemática inversa e determine os pontos-via no espaço das juntas;
6. Empregue o GAT e determine as trajetórias entre pontos-via adjacentes;
7. Compute o funcional (3.21);
8. Aplique a roleta russa para escalonar os indivíduos;
9. Escolha os melhores indivíduos que participarão dos processos evolutivos;
10. Aplique os processos evolutivos aos melhores indivíduos;
11. Fixe o melhor indivíduo da geração anterior à geração posterior;
12. Verifique se NQG $\leq$ QG, se positivo faça NQG = NQG+1 e retorne ao passo #4;
13. Escolher o melhor indivíduo da geração atual.

Neste trabalho optou-se por implementar o *Algoritmo 3.10*, deixando o algoritmo descrito na Figura 3.26 como sugestão para trabalhos futuros. A Figura 3.27 ilustra o resultado obtido pelo emprego do *GAP-2* (**Anexos H** e **I**). À esquerda está esquematizada a marcha no plano sagital; à direita, na parte superior, está ilustrada a marcha no plano frontal e na parte inferior os valores de momentos obtidos ao computar a Equação (3.28).

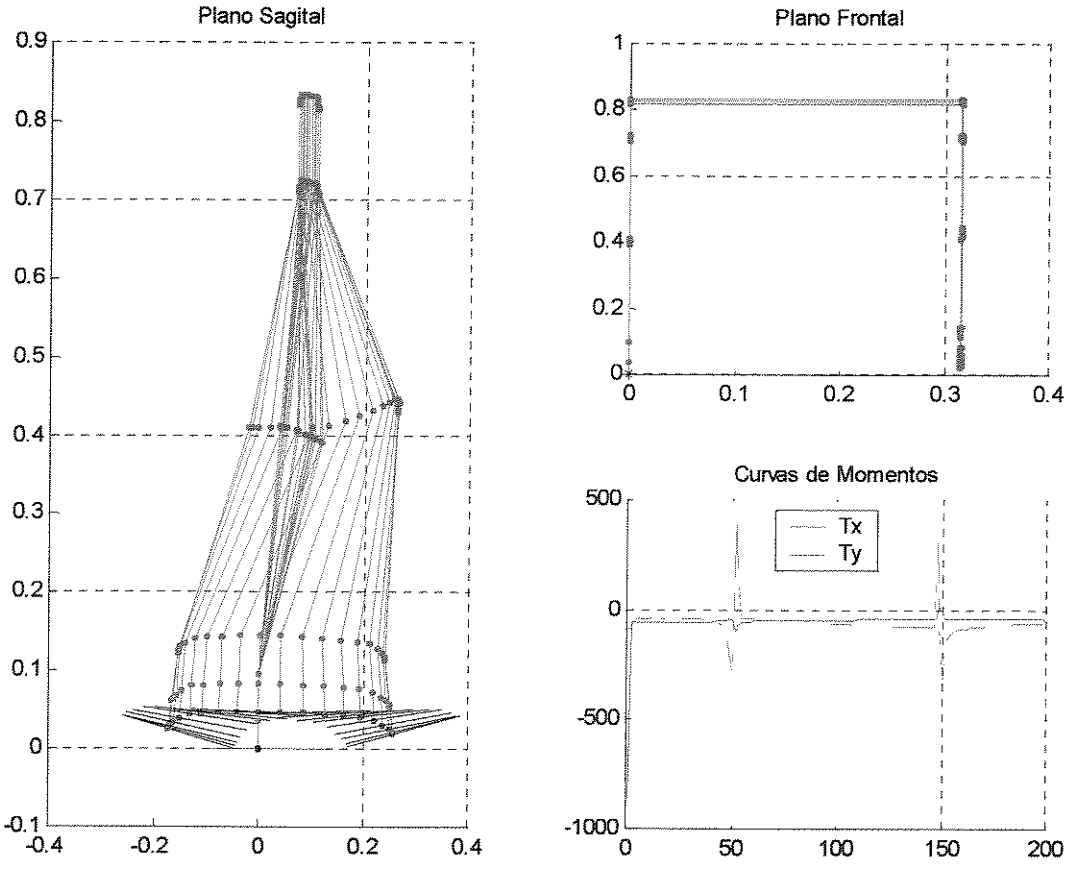


Figura 3.27 – Marcha obtida via GAP-2

Folha em Branco

Capítulo 4

Controle Adaptativo por Modelo de Referência

O modelo dinâmico de um robô bípede é constituído por n equações diferencias ordinárias, não-lineares e acopladas. Aproximações lineares podem resultar em modelos pobres do ponto de vista da caracterização da dinâmica a controlar. Ademais, é necessário considerar aproximações lineares em torno de trajetórias de referência, implicando modelos variantes no tempo (Slotine *et al.*, 1991).

Geralmente durante a modelagem dinâmica alguns fenômenos físicos são desconsiderados (ou aproximados) devido às dificuldades de representá-los adequadamente, comprometendo a robustez do sistema de controle. Assim, é mais conveniente admitir a existência de incertezas no modelo dinâmico, mesmo que aumente a complexidade na fase de projeto.

Essas dificuldades podem ser tratadas empregando-se técnicas de sistema de controle adaptativo baseado em modelos de referência. A lei de controle é constituída por termos baseados no modelo dinâmico da planta, no modelo de referência e em incertezas. Redes neurais de base radial são empregadas para identificação *on-line* de incertezas associadas a fenômenos físicos. Tal escolha assegura a estabilidade em malha fechada no sentido de *Lyapunov* (GE *et al.*, 1998).

4.1. Princípios do Sistema de Controle Adaptativo

4.1.1. Características do Modelo Dinâmico de Robôs

Em geral, modelos dinâmicos de robôs são apresentados conforme a Equação (4.1). As variáveis de posição, velocidade e aceleração angulares são funções do tempo, mas para fins de simplificação da notação não está representada.

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \tau \tag{4.1}$$

Onde:

$\mathbf{M}(\mathbf{q})$	Matriz de inércia associada à planta, $\in \Re^{n\times n}$;
$\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$	Matriz de forças de *Coriolis* e centrífugas, $\in \Re^{n\times n}$;
$\mathbf{F}(\mathbf{q},\dot{\mathbf{q}})$	Vetor de acoplamentos dissipativos, $\in \Re^{n\times n}$;
$\mathbf{G}(\mathbf{q})$	Vetor de carregamento gravitacional, $\in \Re^{n}$;
$\mathbf{q}$	Vetor de posição angular, $\in \Re^{n}$;
τ	Vetor de excitação, $\in \Re^{n}$;
$\dot{z}$	O operador ponto denota a derivada primeira de z(t).

Para adequação aos propósitos do projeto do sistema de controle, uma representação no espaço de estados para a Equação (4.1) é obtida como segue. Seja um vetor de variáveis de estados $\in \Re^{2n}$, dado por:

$$\Omega(t) = \begin{bmatrix} \Omega_1(t) \\ \Omega_2(t) \end{bmatrix} \equiv \begin{bmatrix} \mathbf{q}(t) \\ \dot{\mathbf{q}}(t) \end{bmatrix} \tag{4.2}$$

Reescrevendo a Equação (4.1) utilizando o vetor de estados acima, resultam:

$$\dot{\Omega}_1(t) = \Omega_2(t) \tag{4.3}$$

$$\dot{\Omega}_2(t) = \left[\mathbf{M}(\Omega_1(t))\right]^{-1}\left[\tau(t) - \mathbf{C}(\Omega(t))\dot{\Omega}_1(t) - \mathbf{F}(\Omega(t)) - \mathbf{G}(\Omega_1(t))\right] \tag{4.4}$$

Expressando as Equações (4.3) e (4.4) na forma matricial e escrevendo seus termos de forma mais simples para simplificar a notação, obtém-se:

$$\begin{bmatrix}\dot{\Omega}_1 \\ \dot{\Omega}_2\end{bmatrix} = \begin{bmatrix}\mathbf{0}_{n\times n} & \mathbf{I}_{n\times n} \\ \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n}\end{bmatrix}\begin{bmatrix}\Omega_1 \\ \Omega_2\end{bmatrix} + \begin{bmatrix}\mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} \\ \mathbf{0}_{n\times n} & \mathbf{I}_{n\times n}\end{bmatrix}\begin{bmatrix}\mathbf{0}_{n\times n} \\ \mathbf{M}^{-1}(\tau - \mathbf{C} - \mathbf{F} - \mathbf{G})\end{bmatrix} \tag{4.5}$$

Comparando a Equação (4.5) com o modelo matricial padrão no espaço de estados, pode-se escrever:

$$\dot{\Omega} = \mathbf{A}\Omega + \mathbf{B}\mathbf{M}^{-1}(\tau + \Gamma) \tag{4.6}$$

Onde:

$$\mathbf{A} \triangleq \begin{bmatrix}0_{n\times n} & \mathbf{I}_{n\times n} \\ 0_{n\times n} & 0_{n\times n}\end{bmatrix} \qquad \text{Matriz associada aos estados, } \in \Re^{2n\times 2n}; \tag{4.7}$$

$$\mathbf{B} \triangleq \begin{bmatrix}\mathbf{0}_{n\times n} \\ \mathbf{I}_{n\times n}\end{bmatrix} \equiv \begin{bmatrix}\mathbf{b}_1 & \cdots & \mathbf{b}_n\end{bmatrix} \qquad \text{Matriz associada à excitação, } \in \Re^{2n\times 2n}; \tag{4.8}$$

$$\Gamma \triangleq -(\mathbf{C} + \mathbf{G} + \mathbf{F}) \tag{4.9}$$

4.1.2. Modelo de Referência e Aproximação Linear

Um modelo de referência estável, cujas localizações dos pólos no plano complexo estão de acordo com especificações de projeto, pode ser considerado para que o servomecanismo retro-alimentado tenda a comportamentos análogos a sistemas de segunda ordem padrão (GE *et al.*, 1998). Esse esquema de controle é semelhante à aprendizagem supervisionada, pois o modelo de referência fornece respostas desejadas a excitações (trajetórias de referências) que podem ser

comparadas às respectivas saídas da planta (variáveis medidas). Os desvios (sinais de erros) podem ser utilizados para ajustar os parâmetros do compensador ou para ajustar termos associados a incertezas paramétricas do modelo da planta. A Equação (4.10) descreve um sistema de segunda ordem padrão.

$$\ddot{x} + 2\zeta\omega\dot{x} + \omega^2 x = \omega^2\tau \tag{4.10}$$

A representação da Equação (4.10) na forma de espaço de estados é obtida como segue. Seja um vetor de estados dado por:

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \equiv \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{4.11}$$

Então, a Equação (4.10) é reescrita como segue:

$$\dot{x}_1 = x_2 \tag{4.12}$$

$$\dot{x}_2 = \omega^2\tau - 2\zeta\omega x_2 - \omega^2 x_1 \tag{4.13}$$

Os resultados logo acima são estendidos para representar uma planta com n graus de liberdade, escritos na forma matricial[1].

$$\begin{bmatrix} \dot{\mathbf{x}}_1^1 \\ \dot{\mathbf{x}}_1^2 \\ \dot{\mathbf{x}}_1^3 \\ \vdots \\ \dot{\mathbf{x}}_1^n \\ \dot{\mathbf{x}}_2^1 \\ \dot{\mathbf{x}}_2^2 \\ \vdots \\ \dot{\mathbf{x}}_2^n \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{(n\times n)} & \mathbf{I}_{(n\times n)} \\ \mathbf{A}_{2(n\times n)} & \mathbf{A}_{1(n\times n)} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^1 \\ \mathbf{x}_1^2 \\ \mathbf{x}_1^3 \\ \vdots \\ \mathbf{x}_1^n \\ \mathbf{x}_2^1 \\ \mathbf{x}_2^2 \\ \vdots \\ \mathbf{x}_2^n \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{(n\times n)} & \mathbf{0}_{(n\times n)} \\ \mathbf{0}_{(n\times n)} & -\mathbf{A}_{2(n\times n)} \end{bmatrix} \begin{bmatrix} \mathbf{0}_{(n\times n)} \\ \boldsymbol{\tau}_{(n\times n)} \end{bmatrix} \tag{4.14}$$

[1] O sobrescrito significa o grau de liberdade e o subscrito significa a variável de estado associado. Assim, x_2^1 lê-se segunda variável de estado ($x_2 \equiv \dot{x}$) associada ao primeiro grau de liberdade.

Onde:

$$\mathbf{A}_1 = -\begin{bmatrix} 2\zeta_1\omega_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & 2\zeta_n\omega_n \end{bmatrix}$$ Matriz diagonal com n produtos entre fatores de amortecimentos (ζ) e freqüências naturais (ω); (4.15)

$$\mathbf{A}_2 = -\begin{bmatrix} \omega_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \omega_n^2 \end{bmatrix}$$ Matriz diagonal com n freqüências naturais ω, rad/s; (4.16)

Expressando a Equação (4.14) em uma forma mais compacta, resulta em:

$$\dot{\underline{\Omega}} = \underline{\mathbf{A}}\underline{\Omega} + \underline{\mathbf{B}}\mathbf{u} \tag{4.17}$$

Onde:

$$\underline{\mathbf{A}} = \begin{bmatrix} \mathbf{0}_{(n\times n)} & \mathbf{I}_{(n\times n)} \\ \mathbf{A}_{2(n\times n)} & \mathbf{A}_{1(n\times n)} \end{bmatrix}$$ Matriz associada aos estados, $\in \Re^{2n\times 2n}$; (4.18)

$$\underline{\mathbf{B}} = \begin{bmatrix} \mathbf{0}_{(n\times n)} \\ -\mathbf{A}_{2(n\times n)} \end{bmatrix}$$ Matriz associada à excitação, $\in \Re^{2n\times n}$; (4.19)

$$\underline{\Omega} \equiv \begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} \mathbf{q}_{\text{referência}} \\ \dot{\mathbf{q}}_{\text{referência}} \end{bmatrix}$$ Vetor de estados, $\in \Re^{2nx1}$; (4.20)

$\mathbf{u}_{(nx1)}$ Nova ação de controle, $\in \Re^{nx1}$. (4.21)

Retornando à Equação (4.6), e objetivando realizar uma aproximação linear via realimentação, é considerada uma lei de controle possuindo termos do modelo dinâmico da planta e do modelo de referência:

$$\tau = \mathbf{M}\mathbf{B}^{-1}\underline{\mathbf{B}}\mathbf{u} + \mathbf{M}\begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}\Omega - \Gamma \tag{4.22}$$

Considerando que o modelo dinâmico represente com fidelidade o processo físico (incertezas nulas), a Equação (4.6) pode ser reescrita, utilizando a Equação (4.22).

$$\dot{\Omega} = \left(\mathbf{A} + \mathbf{B}\begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}\right)\Omega + \underline{\mathbf{B}}\mathbf{u}$$

$$\dot{\Omega} = \underline{\mathbf{A}}\,\Omega + \underline{\mathbf{B}}\mathbf{u} \tag{4.23}$$

Definindo-se o erro de rastreamento, obtido pela diferença matemática entre as Equações (4.20) e (4.2), obtém-se:

$$\mathbf{e} = \Omega - \underline{\Omega} \tag{4.24}$$

A partir de (4.24) e utilizando as Equações (4.17) e (4.23), resulta:

$$\dot{\mathbf{e}} = \underline{\mathbf{A}}\,\mathbf{e} \tag{4.25}$$

Portanto, o rastreamento assintótico é garantido por uma escolha adequada da matriz associada ao vetor de estados do modelo de referência, cujos autovalores necessariamente devem possuir parte real negativa. A Equação (4.23) descreve um modelo linear, estável e não-acoplado em malha fechada. Entretanto, há a forte hipótese do pleno conhecimento da dinâmica da planta a controlar, o que possibilita eliminar os efeitos não-lineares do modelo.

4.1.3. Aproximações das Incertezas e Robustez

Alguns fenômenos físicos são difíceis de modelar, principalmente os relacionados aos efeitos de acoplamentos dissipativos. A rigor, a hipótese de conceber um modelo dinâmico que represente os efeitos físicos com a fidelidade exigida pela Equação (4.22), é bastante restritiva à

aplicação prática. Assim, é necessário admitir que o modelo (4.6) seja uma estimativa do processo físico, que não permite o cancelamento exato proposto em (4.23).

Para a Equação (4.6) admitem-se incertezas não paramétricas relacionadas à parcela Γ e considera-se que a matriz de inércia é modelada adequadamente. Assumindo-se que Δ_Γ descreva a dinâmica não modelada (incertezas), passível de ser estimada pelas variáveis medidas do processo físico e da excitação, pode-se propor um modelo aditivo à parcela nominal, como sugere a Equação (4.26).

$$\Gamma = \hat{\Gamma} + \Delta_\Gamma \qquad (\Gamma \text{ valor verdadeiro, } \hat{\Gamma} \text{ valor nominal e } \Delta_\Gamma \text{ incertezas}) \tag{4.26}$$

Portanto, admitindo-se que apenas o valor nominal é conhecido, a Equação (4.22) torna-se:

$$\tau_M = \mathbf{M}\mathbf{B}^{-1}\underline{\mathbf{B}}\mathbf{u} + \mathbf{M}\begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}\Omega - \hat{\Gamma} \tag{4.27}$$

É objetivo do procedimento que segue demonstrar que a estabilidade em malha fechada é mantida ao se utilizar redes neurais RBF para estimar a incerteza. Substituindo a lei de controle (Equação 4.27) na Equação (4.6), obtém-se:

$$\dot{\Omega} = \mathbf{A}\Omega + \mathbf{B}\mathbf{M}^{-1}\left(\mathbf{M}\mathbf{B}^{-1}\underline{\mathbf{B}}\mathbf{u} + \mathbf{M}\begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}\Omega - \hat{\Gamma} + \Gamma\right) \tag{4.28}$$

Utilizando a Equação (4.26), obtém-se:

$$\dot{\Omega} = \left(\mathbf{A} + \mathbf{B}\begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}\right)\Omega + \underline{\mathbf{B}}\mathbf{u} + \mathbf{B}\mathbf{M}^{-1}\Delta_\Gamma \tag{4.29}$$

Utilizando a Equação (4.18), resulta:

$$\dot{\Omega} = \underline{\mathbf{A}}\Omega + \underline{\mathbf{B}}\mathbf{u} + \mathbf{B}\mathbf{M}^{-1}\Delta_\Gamma \tag{4.30}$$

A dinâmica do erro é obtida pela diferença matemática entre as Equações (4.30) e (4.17),

$$\dot{\mathbf{e}} = \underline{\mathbf{A}}\,\mathbf{e} + \mathbf{B}\mathbf{M}^{-1}\Delta_{\Gamma} \tag{4.31}$$

Comparando as Equações (4.25) e (4.31), o rastreamento assintótico é ainda garantido pela escolha adequada da matriz associada ao vetor de estado do modelo de referência. Entretanto, as incertezas causam perturbações à convergência do processo de rastreamento do sinal de referência, diminuindo a robustez do sistema de controle. Mas na seqüência, é demonstrado que a energia de uma função de *Lyapunov* (dependente dos pesos da rede neural RBF e do sinal de erro) é minimizada ao longo do tempo, podendo-se garantir assim a estabilidade assintótica do sistema em malha fechada. E que, com um termo aditivo à lei de controle, é possível melhorar a robustez do sistema de controle.

Neste contexto, as redes neurais artificiais são ferramentas interessantes para identificar parcelas que não estão consideradas no modelo nominal (incertezas). Em especial, as redes neurais de base radial (RBF) fornecem a capacidade de mapeamento não-linear, a adaptabilidade de seus parâmetros em *tempo real* e, principalmente, a garantia da estabilidade do sistema de controle em malha fechada. Além disso, o número de camadas é bem definido (Figura 3.3c) e o processo de aprendizagem é simplificado, como sugerido pela Equação (3.14). Assim sendo e de acordo com a Equação (3.8), os componentes de Δ_{Γ} podem ser estimados pela Equação (4.32), escrita na forma matricial. Admite-se que o processo de identificação esteja sujeito a erros, fornecendo, portanto, valores aproximados para as incertezas.

$$\Delta_{\Gamma} = \begin{bmatrix} \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_n \end{bmatrix} \equiv \begin{bmatrix} \mathbf{v}_1^T\boldsymbol{\rho} \\ \mathbf{v}_{21}^T\boldsymbol{\rho} \\ \vdots \\ \mathbf{v}_n^T\boldsymbol{\rho} \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix} = \{\mathbf{v}\}^T \bullet \{\boldsymbol{\rho}\} + \Xi \tag{4.32}$$

Onde:

$$\Delta_k = \begin{bmatrix} 1 & \mathbf{v}_1 & \cdots & \mathbf{v}_m \end{bmatrix} \begin{bmatrix} \mathbf{v}_k \\ \boldsymbol{\rho}_1 \\ \vdots \\ \boldsymbol{\rho}_m \end{bmatrix} + \varepsilon_k \equiv \mathbf{v}_k^T\boldsymbol{\rho} + \varepsilon_k \tag{4.33}$$

$$\Xi = \{\varepsilon_1 \quad \varepsilon_2 \quad \cdots \quad \varepsilon_n\}^T$$

$\in_k$ Erro de identificação da k-ésima rede neural RBF.

A estabilidade e a melhoria da robustez do sistema em malha fechada pode ser assegurada assumindo-se uma parcela aditiva (τ_R) à ação de controle (GE *et al.*, 1998). Reescrevendo-se a Equação (4.31) por meio de (4.32) e empregando a parcela aditiva τ_R, resulta:

$$\dot{\mathbf{e}} = \underline{\mathbf{A}}\mathbf{e} + \mathbf{B}\mathbf{M}^{-1}\left(\left\{\{\mathbf{v}\}^T \bullet \{\rho\}\right\} + \Xi\right) + \mathbf{B}\mathbf{M}^{-1}\tau_R \tag{4.34}$$

Admitindo-se a Equação (4.34) como candidata à função de *Lyapunov*, segue:

$$\mathbf{V}(\mathbf{e},\mathbf{v}) = \mathbf{e}^T\mathbf{P}\mathbf{e} + \sum_{i=1}^{n} \mathbf{v}_i^T \Pi_i^{-1} \mathbf{v}_i \tag{4.35}$$

Onde:

e sinal de erro, Equação (4.24);

v Vetor de pesos da rede neural RBF;

$\mathbf{P}=\mathbf{P}^T$ Solução da equação de *Lyapunov* $\underline{\mathbf{A}}^T\mathbf{P} + \mathbf{P}\underline{\mathbf{A}} = -\mathbf{Q}$; (4.36)

Π_i Matriz simétrica definida positiva, constante. (4.37)

Tomando a derivada de (4.35), resulta:

$$\dot{\mathbf{V}}(\mathbf{e},\mathbf{v}) = \mathbf{e}^T\mathbf{P}\dot{\mathbf{e}} + \mathbf{e}^T\mathbf{P}^T\dot{\mathbf{e}} + \sum_{i=1}^{n} \mathbf{v}_i^T\left(\Pi_i^{-1}\dot{\mathbf{v}}_i + \left(\Pi_i^{-1}\right)^T\dot{\mathbf{v}}_i\right) \tag{4.38}$$

E utilizando a Equação (4.34), obtém-se:

$$\dot{V}(e,v) = 2e^T P\left[\underline{A}e + BM^{-1}\left\{\{v\}^T \bullet \{\rho\}\right\} + BM^{-1}(\Xi + \tau_R)\right] + 2\sum_{i=1}^{n} v_i^T \Pi_i^{-1} \dot{v}_i \qquad (4.39)$$

Para garantir estabilidade em malha fechada é necessário que a Equação (4.39) seja definida negativa. Assim, a taxa de variação temporal dos pesos da rede neural RBF pode assumir a seguinte forma (GE *et al.*, 1998).

$$\dot{v}_i = -\Pi_i \rho_i e^T P b_i M^{-1} \qquad (4.40)$$

Onde:

$\dot{v}_i$	Taxa de variação temporal dos pesos do i-ésimo neurônio;
Π_i	Equação (4.37);
ρ_i	Função de base radial associada ao i-ésimo neurônio;
e	sinal de erro, Equação (4.24);
b_i	Vetor coluna relativo Equação (4.8);
M	Matriz de inércia.

Substituindo a Equação (4.40) em (4.39), segue-se que:

$$\dot{V}(e,v) = 2e^T P\left[\underline{A}e + BM^{-1}\left\{\{v\}^T \bullet \{\rho\}\right\} + BM^{-1}(\Xi + \tau_R)\right] - 2\sum_{i=1}^{n} v_i^T \rho_i e^T P b_i M^{-1} \qquad (4.41)$$

Pela Equação (4.32),

$$\dot{V}(e,v) = 2e^T P\left[\underline{A}e + BM^{-1}\left\{\{v\}^T \bullet \{\rho\}\right\} + BM^{-1}(\Xi + \tau_R)\right] - 2\, e^T PBM^{-1}\left\{\{v\}^T \bullet \{\rho\}\right\}$$

$$\dot{V}(e,v) = 2e^T P\underline{A}e + 2e^T PBM^{-1}(\Xi + \tau_R) \qquad (4.42)$$

Modificando a Equação (4.42) por intermédio de (4.36), resulta:

$$\dot{V}(e,v) = -e^T Qe + 2e^T PBM^{-1}(\Xi + \tau_R) \tag{4.43}$$

Dados que **P** e **B** são matrizes definidas positivas e que a matriz **M** seja não singular, a Equação (4.43) será definida negativa se sua segunda parcela for negativa ou nula.

$$e^T PBM^{-1}(\Xi + \tau_R) \leq 0 \tag{4.44}$$

De onde resulta a análise que segue.

1. O produto $e^T PBM^{-1}$ é positivo, implica em $(\Xi+\tau_R) \leq 0 \ \therefore -\tau_R \geq \Xi$;
2. O produto $e^T PBM^{-1}$ é negativo, implica em $(\Xi+\tau_R) \geq 0 \ \therefore -\tau_R \leq \Xi$;

Assim, a parcela aditiva τ_R deve levar em consideração o sinal algébrico do produto $e^T PBM^{-1}$, o que fornece a seguinte lei de controle aditiva, sendo sgn(•) a função sinal e κ um vetor de ganhos cujas componentes é definido pela Equação (4.46),

$$\tau_R = -\kappa \, \mathrm{sgn}(e^T PBM^{-1}) \tag{4.45}$$

$$\kappa_i \geq |\hat{\Gamma}_i| \tag{4.46}$$

Utilizando a Equação (4.45) em (4.43), obtém-se:

$$\dot{V}(e,v) = -e^T Qe + 2e^T PBM^{-1}(\Xi - \kappa \, \mathrm{sgn}(e^T PBM^{-1})) \tag{4.47}$$

Assim, utilizando a conclusão (4.46), resulta:

$$\dot{V}(e,v) = -e^T Qe - 2\|e^T PBM^{-1}\| < 0 \tag{4.48}$$

Dos resultados obtidos, as Equações (4.40) e (4.46) definem os termos adaptativos correspondentes aos pesos da rede neural RBF e do vetor de ganho associado ao termo aditivo τ_R (Equação 4.45), respectivamente. A lei de controle (Equação 4.27) deve ser reescrita para contemplar o termo aditivo τ_R, como sugere a Equação (4.49).

$$\tau = \tau_M + \tau_R \equiv \mathbf{M}\mathbf{B}^{-1}\underline{\mathbf{B}}\mathbf{u} + \mathbf{M}\begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}\boldsymbol{\Omega} - \left(\hat{\Gamma} + \Delta_\Gamma\right) - \kappa\,\mathrm{sgn}\left(\mathbf{e}^T\mathbf{P}\mathbf{B}\mathbf{M}^{-1}\right) \tag{4.49}$$

A lei de controle (Equação 4.49) requer a constituição da Equação (4.32). O algoritmo *OLS* (Chen *et al.*, 1991; Munzir *et al.*, 2000) ilustrado na Figura 3.4 foi empregado para determinar o número de neurônios da rede RBF, fornecendo os centros (μ) mais significativos de um conjunto de centros candidatos.

Os centros candidatos foram obtidos por simulação computacional, utilizando-se a Equação (4.6), assumindo desvios de vinte por cento dos valores nominais relativos à massa de cada elo (Tabela 2.2) e eliminando-se os termos de acoplamentos dissipativos, $\mathbf{F} \equiv \mathbf{0}$. E admitindo-se que os demais termos representam adequadamente a dinâmica do robô bípede. Uma excitação em varredura (*chirp*) foi aplicada a Equação (4.6) e o método de *Runge-Kutta* de quarta ordem foi empregado para a integração numérica, adotando-se um passo de integração variado. Foram armazenadas $\boldsymbol{\Omega}(t)$ (o vetor de estados) e $\hat{\Gamma}(t)$ (parcela nominal), para t = 0, 1, ..., L amostras.

Para a seleção dos centros (dentre os centros candidatos) pelo emprego do algoritmo *OLS* são necessários à montagem das matrizes de excitação (entrada da rede RBF) e de saída (padrões de comparação), bem como a composição dos centros candidatos.

A Equação (4.50) define a matriz de saída (padrões de comparação), composta pela parcela nominal. Cada vetor coluna contém N amostras (N < L), escolhidas aleatoriamente dentre as L amostras de $\hat{\Gamma}(t)$.

$$\hat{\Gamma}(t) = \begin{bmatrix} \hat{\Gamma}_1(1) & \hat{\Gamma}_2(1) & \cdots & \hat{\Gamma}_{10}(1) \\ \hat{\Gamma}_1(2) & \hat{\Gamma}_2(1) & \vdots & \hat{\Gamma}_{10}(2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{\Gamma}_1(N) & \hat{\Gamma}_2(N) & \cdots & \hat{\Gamma}_{10}(N) \end{bmatrix} \tag{4.50}$$

De maneira análoga à Equação (4.50), a Equação (4.51) descreve o vetor de excitação aplicado à entrada da rede RBF, onde cada vetor coluna contém N amostras (N < L), escolhidas aleatoriamente dentre as L amostras de $\Omega(t)$.

$$x(t) = [x_1(t) \quad x_2(t) \quad \cdots \quad x_{10}(t)] \tag{4.51}$$

Para i = 1, 2, ... 10, cada vetor coluna da Equação (4.51) é composto por:

$$\begin{aligned} x_i(t) = [&1(t) \quad q_1(t-1) \quad q_1(t-2) \quad \cdots \quad q_i(t-n_q) \quad \cdots \quad q_{10}(t) \quad \cdots \\ &q_{10}(t-n_q) \quad \dot{q}_1(t-1) \quad \cdots \quad \dot{q}_1(t-n_\tau) \quad \cdots \quad \dot{q}_{10}(t-1) \quad \cdots \quad \dot{q}_{10}(t-n_\tau)]^T \end{aligned} \tag{4.52}$$

Análogo às Equações (4.51) e (4.52), a Equação (4.53) define os 1750 centros candidatos selecionados a partir do vetor de estados, onde cada vetor coluna contém N amostras (N < L), escolhidas aleatoriamente dentre as L amostras de $\Omega(t)$.

$$\mu(t) = [\mu_1(t) \quad \mu_2(t) \quad \cdots \quad \mu_{1750}(t)] \tag{4.53}$$

Para i = 1, 2, ... 1750, cada vetor coluna da Equação (4.53) é composto por:

$$\begin{aligned} \mu_i(t) = [&q_1(t-1) \quad q_1(t-2) \quad \cdots \quad q_1(t-n_1) \quad \cdots \quad q_{10}(t-2) \quad \cdots \quad q_{10}(t-n_{10}) \quad \cdots \\ &\dot{q}_1(t-1) \quad \cdots \quad \dot{q}_1(t-n_1) \quad \cdots \quad \dot{q}_{10}(t-1) \quad \cdots \quad \dot{q}_{10}(t-n_{10})]^T \end{aligned} \tag{4.54}$$

Com as Equações (4.50), (4.51) e (4.53) o programa *OLS_MIMO*, que implementa o algoritmo *OLS* em comandos de *Matlab*® e descrito no **Anexo J**, foi empregando para selecionar entre os 1750 centros candidatos, os mais significativos. Foi utilizada a função inversa de *Hardy* (Equação 3.11) e a variância foi computada pela Equação (4.55).

$$\sigma_i = \frac{\max(\mu_i)}{\sqrt{2N}} \tag{4.55}$$

Com uma tolerância adotada de $\rho = 0{,}2$, foram selecionados 167 centros significativos dentre os centros candidatos. E os pesos da rede neural RBF foram computados a cada instante de tempo pela Equação (4.40).

A segunda etapa do projeto diz respeito às escolhas dos parâmetros do modelo de referência (Equação 4.17), que podem ser escolhidos prontamente utilizando parâmetros que definem uma resposta de sistemas de segunda ordem a uma entrada degrau.

A última etapa está relacionada às escolhas da constante multiplicativa associada ao termo robusto (Equação 4.45), da matriz de ponderação (Equação 4.37) e da solução para a Equação (4.36). A obtenção desses valores foi realizada através de tentativas e erros e os mais adequados são apresentados na Tabela 4.1.

Tabela 4.1 – Resumo dos parâmetros utilizados para o controlador

	Membros Inferiores	*Pêndulo Invertido*
Equação (4.45)	κ (conforme Equação 4.46)	κ (conforme Equação 4.46)
Equação (4.36)	$Q = I_{20\times20}$	$Q = I_{4\times4}$
Equação (4.373)	$\Pi_i = 0{,}05\ I_{m\times m}$	$\Pi_i = 0{,}1\ I_{m\times m}$

4.3. Implementação Computacional

Nesta seção os resultados obtidos em Gonçalves *et al.* (2003) são estendidos. Como já mencionado, o robô bípede foi subdividido em dois subsistemas: tronco (pêndulo invertido) e os membros inferiores. O acoplamento entre esses subsistemas pode ser definido como indicam as Equações (4.56) e (4.57), que descrevem as dinâmicas do pêndulo invertido e dos membros inferiores, respectivamente.

$$\begin{bmatrix} \dot{q}_{11} \\ \dot{q}_{12} \\ \ddot{q}_{11} \\ \ddot{q}_{12} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{2x2} & \mathbf{I}_{2x2} \\ \mathbf{0}_{2x2} & \mathbf{0}_{2x2} \end{bmatrix} \begin{bmatrix} q_{11} \\ q_{12} \\ \dot{q}_{11} \\ \dot{q}_{12} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{2x2} \\ \mathbf{I}_{2x2} \end{bmatrix} \begin{bmatrix} d_{22} & 0 \\ 0 & d_{33} \end{bmatrix}^{-1} \left(\begin{bmatrix} \tau_{11} \\ \tau_{12} \end{bmatrix} - \begin{bmatrix} \Gamma_{11} \\ \Gamma_{12} \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ d_{13}\,{}^{0}\ddot{z}_5 \end{bmatrix} \right) \tag{4.56}$$

$$\begin{bmatrix} \dot{\mathbf{q}}_1 \\ \vdots \\ \dot{\mathbf{q}}_{10} \\ \ddot{\mathbf{q}}_1 \\ \vdots \\ \ddot{\mathbf{q}}_{10} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{10\times10} & \mathbf{I}_{10\times10} \\ \mathbf{0}_{10\times10} & \mathbf{0}_{10\times10} \end{bmatrix} \begin{bmatrix} \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_{10} \\ \dot{\mathbf{q}}_1 \\ \vdots \\ \dot{\mathbf{q}}_{10} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{10\times10} \\ \mathbf{M}^{-1}(\tau - \Gamma - \Delta) \end{bmatrix} \tag{4.57}$$

Onde:

$$\Delta = \begin{bmatrix} \Delta_1 \\ \vdots \\ \Delta_5 \\ \vdots \\ \Delta_{10} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ \mathbf{d}_{13}\ddot{\theta}_{12} - \mathbf{c}_1(\mathbf{q}_{11}, \dot{\mathbf{q}}_{12}) \\ \vdots \\ 0 \end{bmatrix} \tag{4.58}$$

As equações (4.56) e (4.57) foram implementadas em Matlab/Simulink® empregando-se as *S-functions*, conforme mostram as figuras que seguem.

Objetivando esclarecer os pontos fortes da abordagem proposta e implementada neste trabalho, a Figura 4.1 esquematiza o Gerador Automático de Marcha (*GAM*) que a partir de parâmetros de entrada (comprimento do passo $\boldsymbol{\lambda}$, velocidade de trajeto $\boldsymbol{\nu}$, altura máxima para o pé em balanço $\mathbf{d}$ e a inclinação entre o pé o solo $\mathbf{q}$) permite especificar uma marcha no plano cartesiano do robô bípede que, pelo emprego da cinemática inversa, gera trajetórias angulares de referência. Um Controlador Adaptativo por Modelo de Referência, empregando redes neurais artificiais de base radial, permite rastrear as trajetórias de referência. Sistemas de controle similares são empregados para o tronco e para os membros inferiores, porém independentes. E por fim, um Gerador Automático de Trajetórias para o Tronco (GAT) que emprega redes neurais

artificiais MLP para processar informações de posições e acelerações dos membros inferiores e fornecer uma trajetória do tronco para manter o balanço dinâmico.

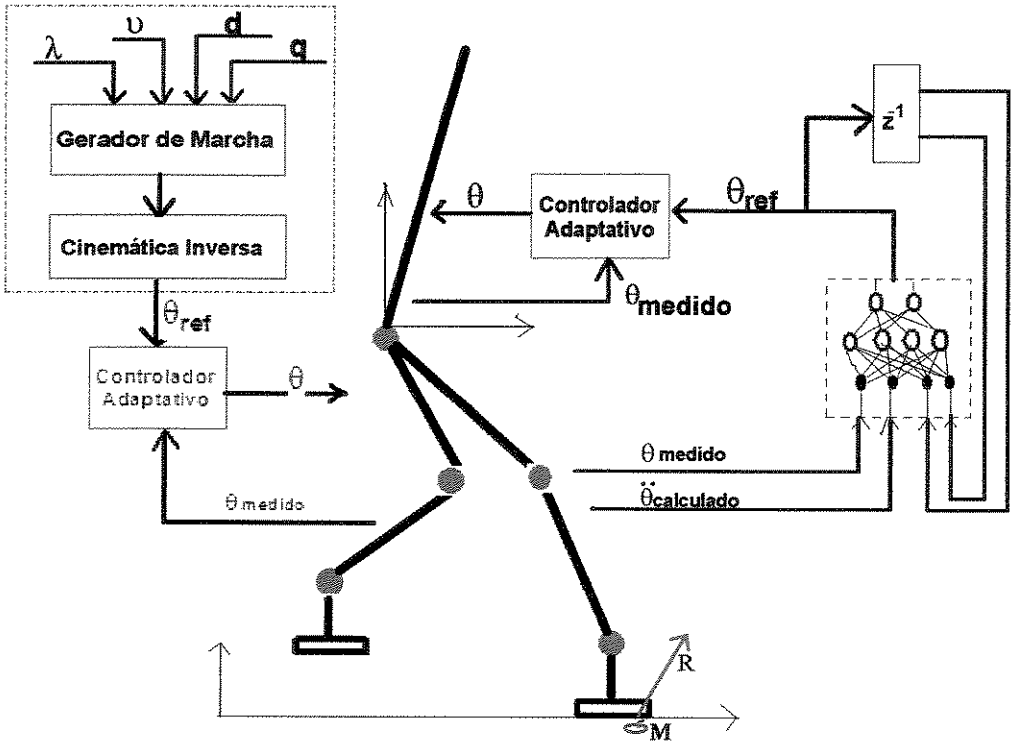


Figura 4.1 – Abordagem proposta e implementada neste trabalho.

A abordagem proposta foi implementada em comandos do Matlab/Simulink, como apresentado na seqüência. A Figura 4.2 é a implementação das equações dinâmicas para os subsistemas que compõem o robô bípede: tronco (Equação 4.56, implementada no **Anexo C**) e membros inferiores (Equação 4.57, implementada no **Anexo D**). Também são mostradas as conexões entre esses subsistemas, caracterizando as perturbações provocadas pelo tronco aos membros inferiores (Perturbação_Tronco) e vice-versa (Perturbação_MMI). O **Anexo L** traz as suas respectivas implementações computacionais. A conexão denominada por "u_tronco" recebe a trajetória para o tronco, proveniente do *gerador de trajetórias do tronco*. A conexão "u_MMI" recebe a marcha planejada pelo *gerador automático de marchas*.

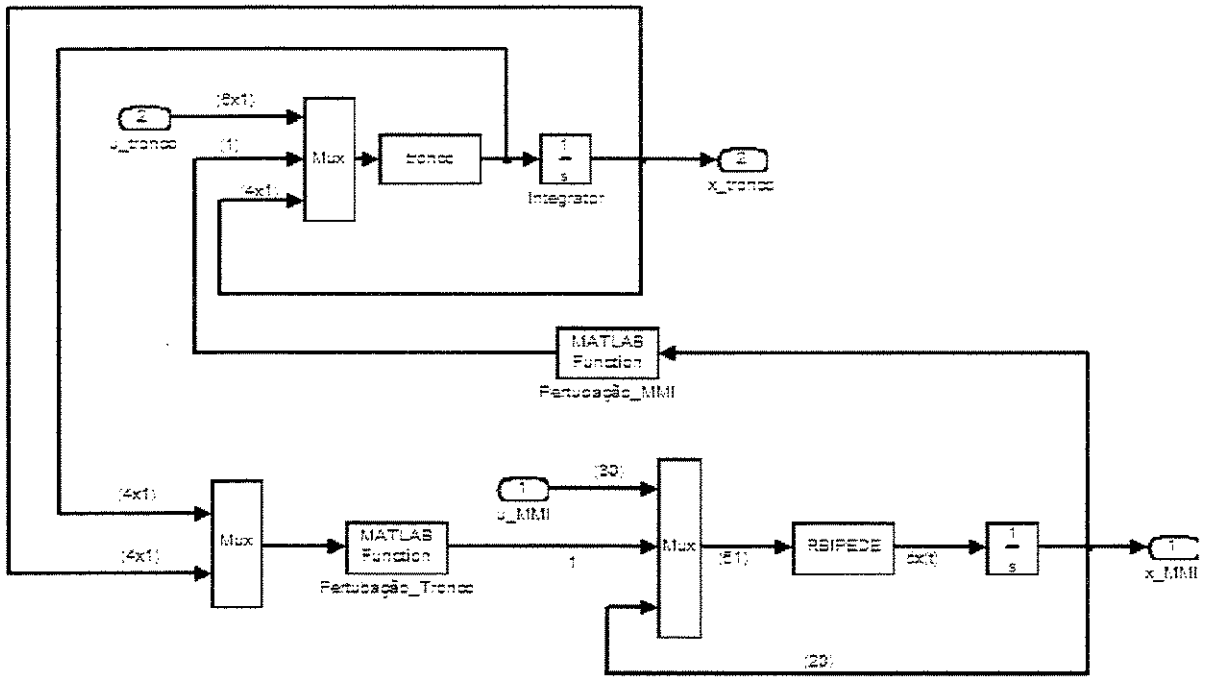


Figura 4.2 – Implementação do modelo matemático do robô bípede

O *sistema de controle adaptativo* e o *gerador de trajetórias para o tronco* compõem os mecanismos propostos para a solução no modo de andar dinâmico, conforme a Figura 4.1. A Figura 4.3 ilustra a implementação do sistema integrado. O bloco "RB" contém os modelos dos subsistemas e das perturbações, conforme Figura 4.2. Os sistemas de controle adaptativos para o tronco (SC_Tronco) e para os membros inferiores (SC_MMI) são independentes, porém similares, e fornecem as ações de controle para os membros inferiores (u_MMI) e para o tronco (u_tronco). AC_Tronco recebe os sinais de posições angulares dos membros inferiores (Ramo 3) que são comparados com as correspondentes trajetórias planejadas pelo *gerador automático de marcha GAM* (Ramo 1). SC_tronco recebe os sinais de posições angulares do tronco (Ramo 4) que são comparados com as correspondentes trajetórias planejadas pelo *gerador automático de trajetórias GAT* (Ramo 2). O *gerador automático de trajetórias para o tronco GAT* recebe os sinais de posições angulares do *GAT (Ramo 1)* e da retro-alimentação (Ramo 4) para compor os sinais de excitação da rede neural recorrente e fornecer uma trajetória de referência para o tronco (Ramo 2).

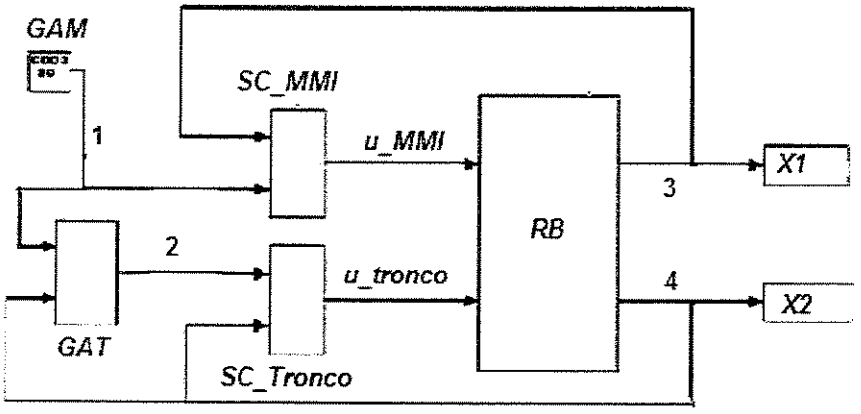


Figura 4.3 – Implementação Sistêmica do Controle Postural.

A Figura 4.4 ilustra o sistema de controle adaptativo projetado para os membros inferiores. A conexão 2 (ref) recebe a marcha planejada (sinais de posições angulares) e a conexão 1 (x) recebe os sinais retro-alimentados. Um modelo similar descreve o sistema de controle para o tronco. O **Anexo M** traz a implementação computacional da Rede RBF (GAMA_RBF) projetada para identificar termos não modelados referentes aos membros inferiores e o **Anexo N** traz a implementação da Rede RBF projetada para o tronco do robô bípede.

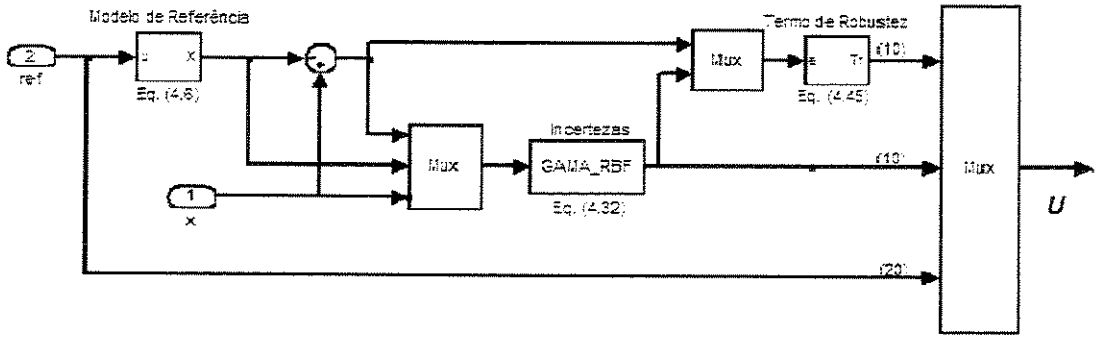


Figura 4.4 – Implementação do modelo de referência e da lei de controle (SC_MMI)

No quinto capítulo serão realizadas simulações e análises do sistema apresentado na Figura 4.3.

Capítulo 5

Simulaçoes Computacionais

Neste capítulo, o gerador automático de marcha (*GAM*) é empregado para definir trajetórias de referências para os membros inferiores do robô bípede. O comprimento do passo, a altura máxima para o pé em balanço, a velocidade de trajeto e os instantes de tempo para cada postura básica são pré-especificados. Então, o *GAM* seleciona uma marcha cuja característica principal é a de fornecer estabilidade postural (minimizando o erro quadrático médio da trajetória de momentos, que devem ser nulos quando o pé toca o solo). As trajetórias de posições e acelerações angulares são computadas pelo gerador automático de trajetórias (*GAT*). As velocidades nos pontos-via são obtidas por meio de Jacobianos.

Planejada uma marcha, as posições e velocidades angulares são gravadas em arquivos e são fornecidos como sinais de referência ao sistema de controle integrado. Para efetivação do sistema em malha fechada proposto é necessário medir as posições e velocidades relativas aos membros inferiores e ao tronco, que são utilizadas como sinais de retroação para computar os desvios em relação às trajetórias de referências. Os sinais de desvios são utilizados pelo sistema de controle para corrigir os níveis de torque aplicados às juntas. As acelerações angulares requeridas pelas redes neurais recorrentes (utilizadas para fornecer trajetórias para o tronco) e pelas redes neurais da base radial (utilizadas na identificação dos parâmetros não modelados) são computadas por

diferenciação numérica. Admite-se que a superfície de apoio é plana e que o pé de apoio não desliza sobre a mesma.

5.1. Geração de Marcha

Por exemplo, seja uma marcha com as seguintes características: o quinto elo permanece paralelo em relação ao solo, o comprimento do passo é igual **0,17** m, a velocidade de trajeto é igual a **0,55** m/s, o ângulo entre o pé em balanço e o solo é igual a **0,2** rad e a altura máxima para o pé em balanço é igual a **0,0386** m. O tempo total para completar uma passada é de **0,9** s. Vinte por cento deste tempo é gasto na fase de bi-apoio.

Com esses requisitos impostos, o gerador de marcha foi acionado e os resultados são apresentados na seqüência. Os parâmetros utilizados para os algoritmos genéticos relativos ao *GAM* foram assim estipulados:

gmax	= 50	máximo número de evoluções (critério de parada)
popsize	= 130	tamanho da população
pc	= 0,250	taxa de recombinação
pm	= 0,025	taxa de mutação

Para a transição da postura básica B a D obteve-se os seguintes parâmetros (Eq. 3.103):

$$\gamma_x = \ 0{,}0100 \quad \gamma_y = -0{,}0100 \quad \gamma_z = \ 0{,}5396$$

O vetor posição do quadril na postura D está definido a seguir:

$$p_4 = \{0{,}0017 \quad -0{,}0017 \quad 0{,}5396\}^T$$

A Figura 5.1 mostra a transição da postura básica B para a D, sendo que à esquerda está definida a trajetória no plano sagital. À direita, no canto superior, está a trajetória no plano frontal e logo abaixo estão as curvas de momentos, segundo a equação (3.28). A Figura 5.2 mostra as

trajetórias dos elos no plano cartesiano, divididas em seis grupos. A primeira coluna traz as trajetórias na direção X; na segunda coluna, as trajetórias na direção Y e na última, as trajetórias na direção Z. A fileira de cima traz as trajetórias para a perna de apoio e a fileira de baixa, para a perna em balanço. As Figuras 5.3 e 5.4 apresentam as velocidades e acelerações.

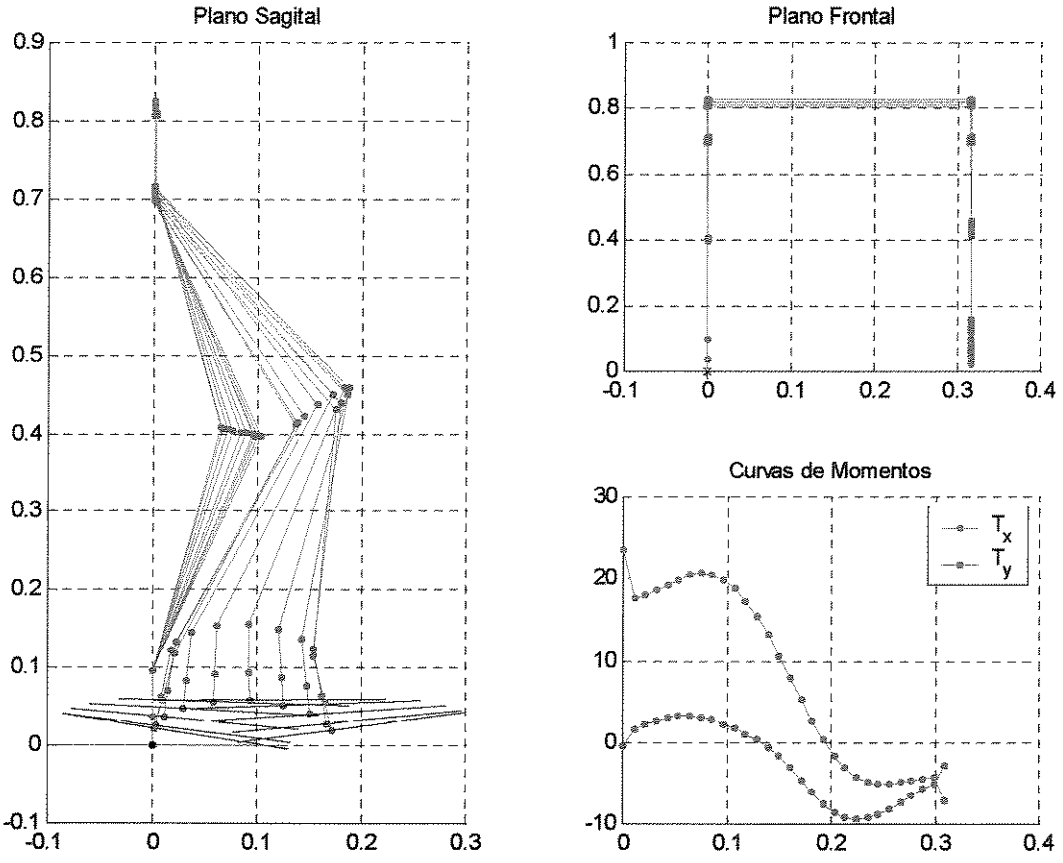


Figura 5.1 – Transição da postura básica B para a D.

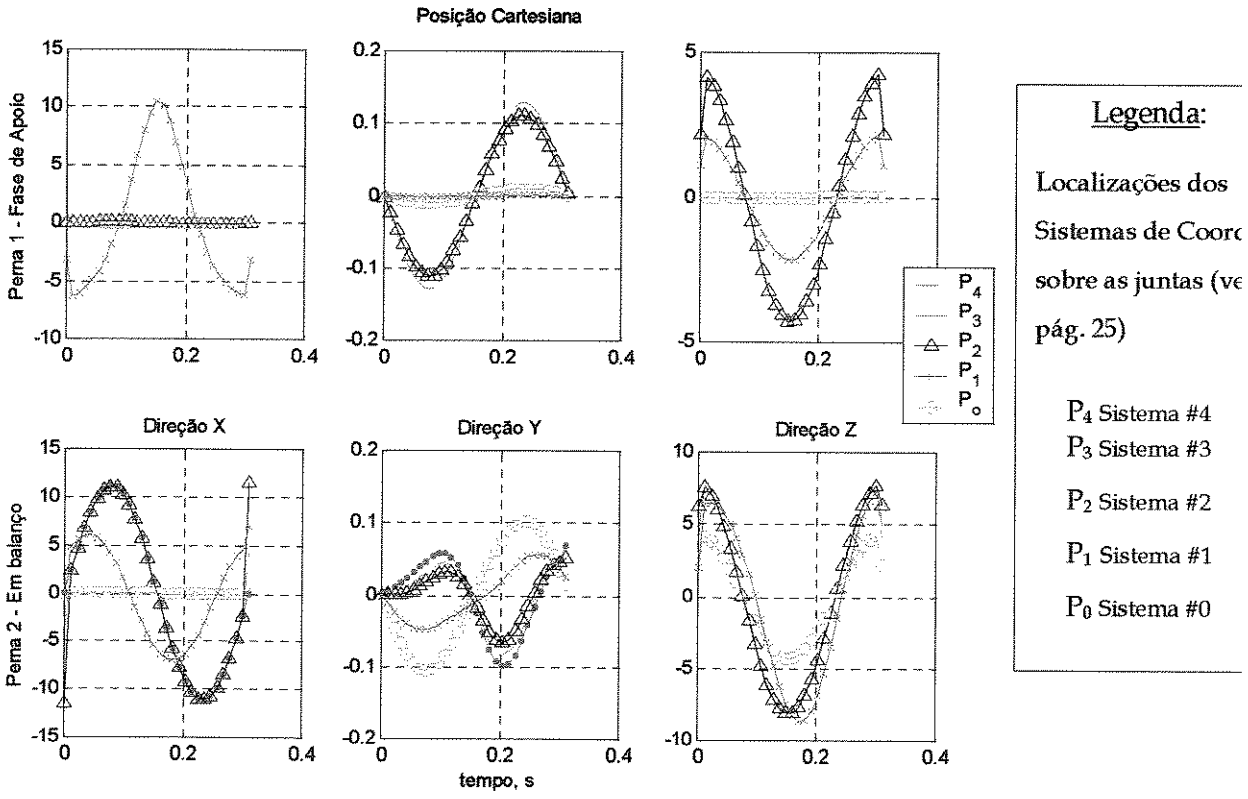


Figura 5.2 – Deslocamentos, m (plano cartesiano)

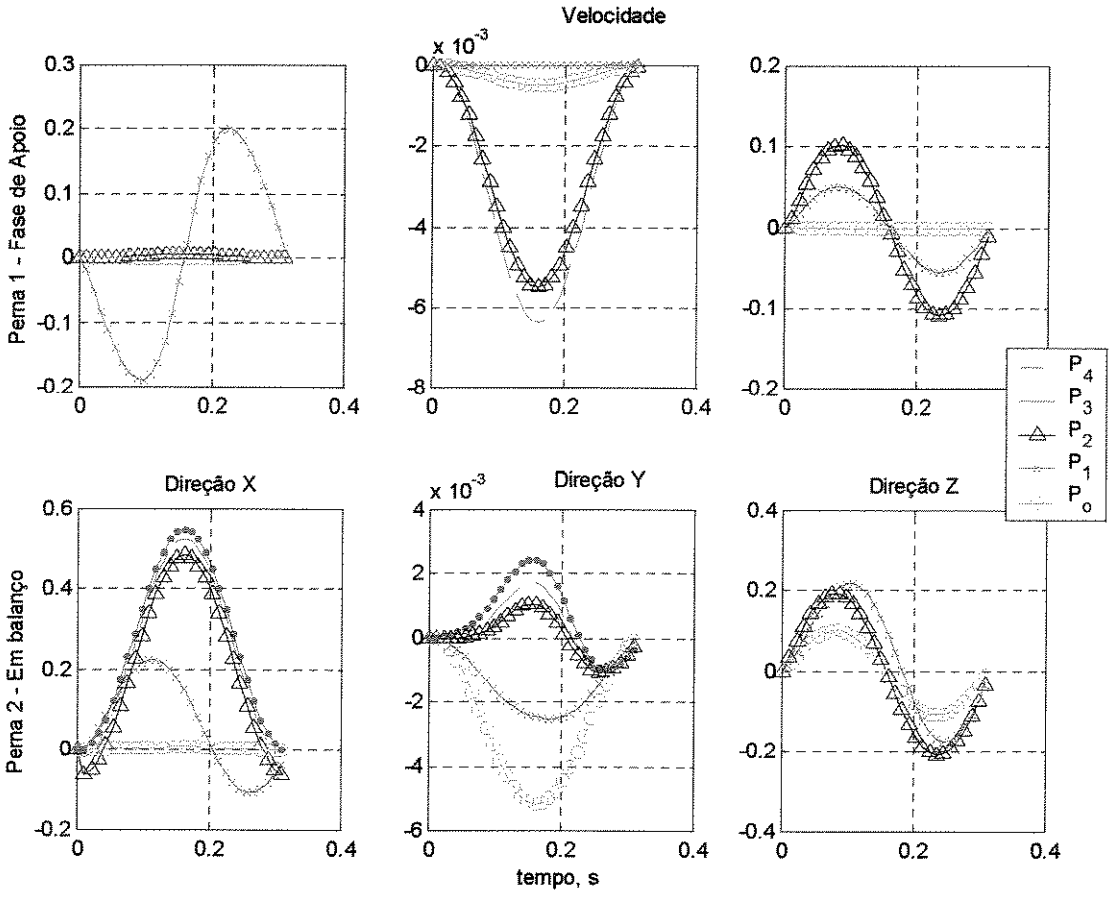


Figura 5.3 – Velocidade, m/s (plano cartesiano)

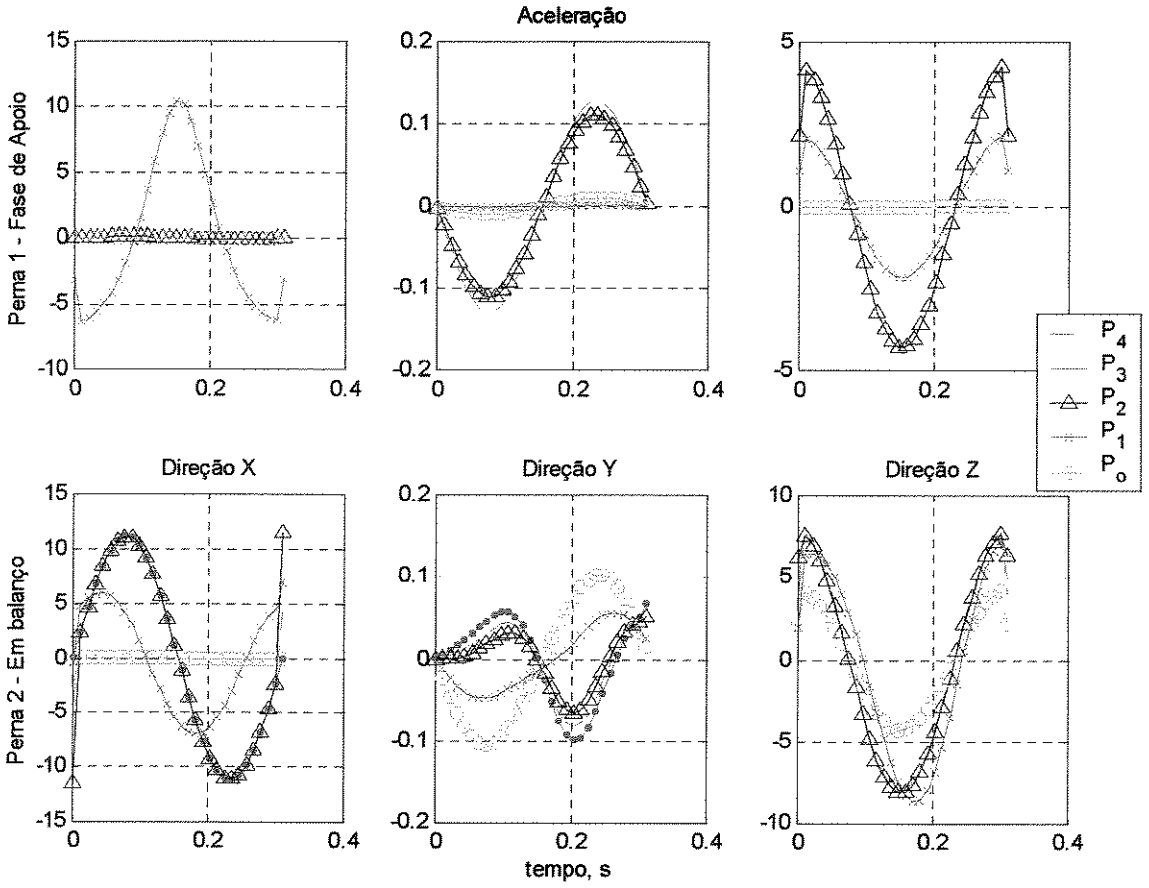


Figura 5.4 – Aceleração, m/s^2 (plano cartesiano)

Foram impostos os mesmos requisitos de marcha para sintetizar a transição da postura básica E para a D, com exceção do comprimento do passo que passa a ser igual ao dobro do valor inicial. A condição inicial do robô bípede é a condição final da fase anterior e a contagem dos tempos é reinicializada. Nessas circunstâncias, os seguintes resultados foram obtidos:

- Parâmetros relativos à equação (3.103):

$$\gamma_x = 0{,}0390 \quad \gamma_y = 0{,}0006 \quad \gamma_z = 0{,}5311$$

- Vetor posição do quadril na postura D:

$$p_4 = \{0{,}0084 \quad -0{,}0002 \quad 0{,}6671\}^T$$

A Figura 5.5 mostra a transição da postura básica E para a D, sendo que à esquerda está a trajetória no plano sagital. À direita, no canto superior, está a trajetória no plano frontal e logo abaixo estão as curvas de momentos, segundo a Equação (3.28).

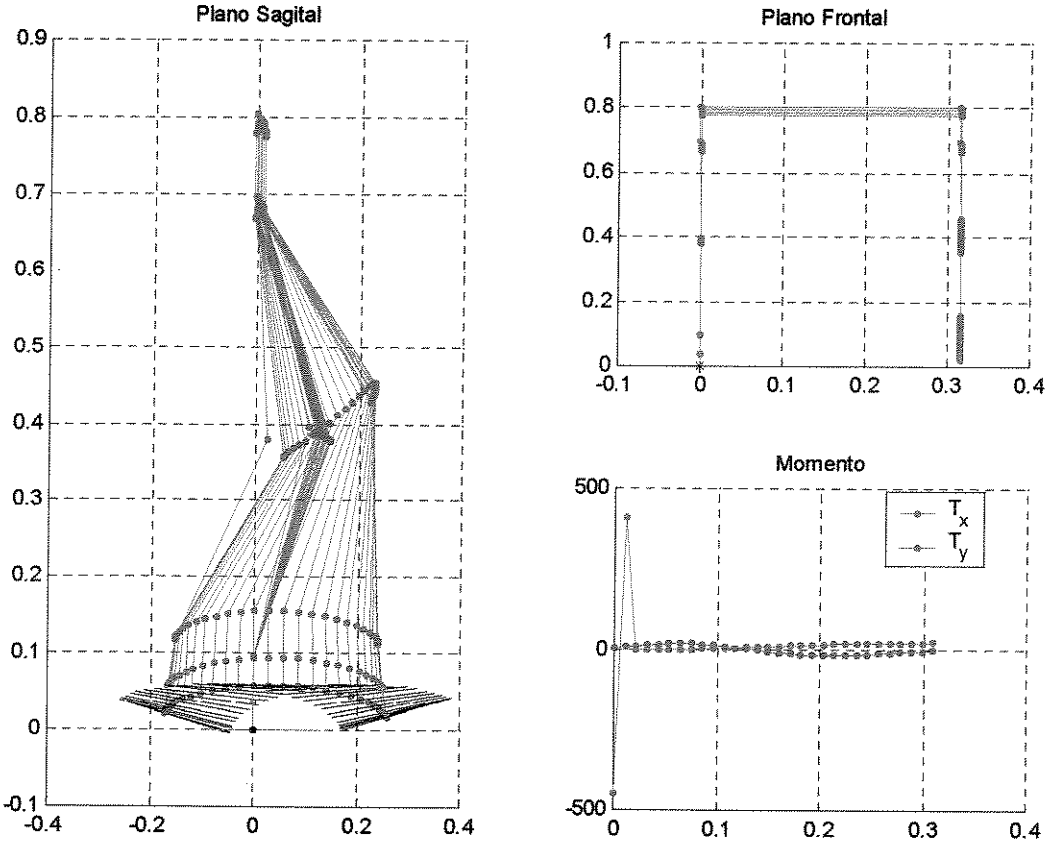


Figura 5.5 – Transposição da postura básica E para a D.

As Figuras 5.6, 5.7 e 5.8 mostram as trajetórias, as velocidades e as acelerações dos elos, no plano cartesiano, divididas em seis grupos de forma similar às figuras anteriores.

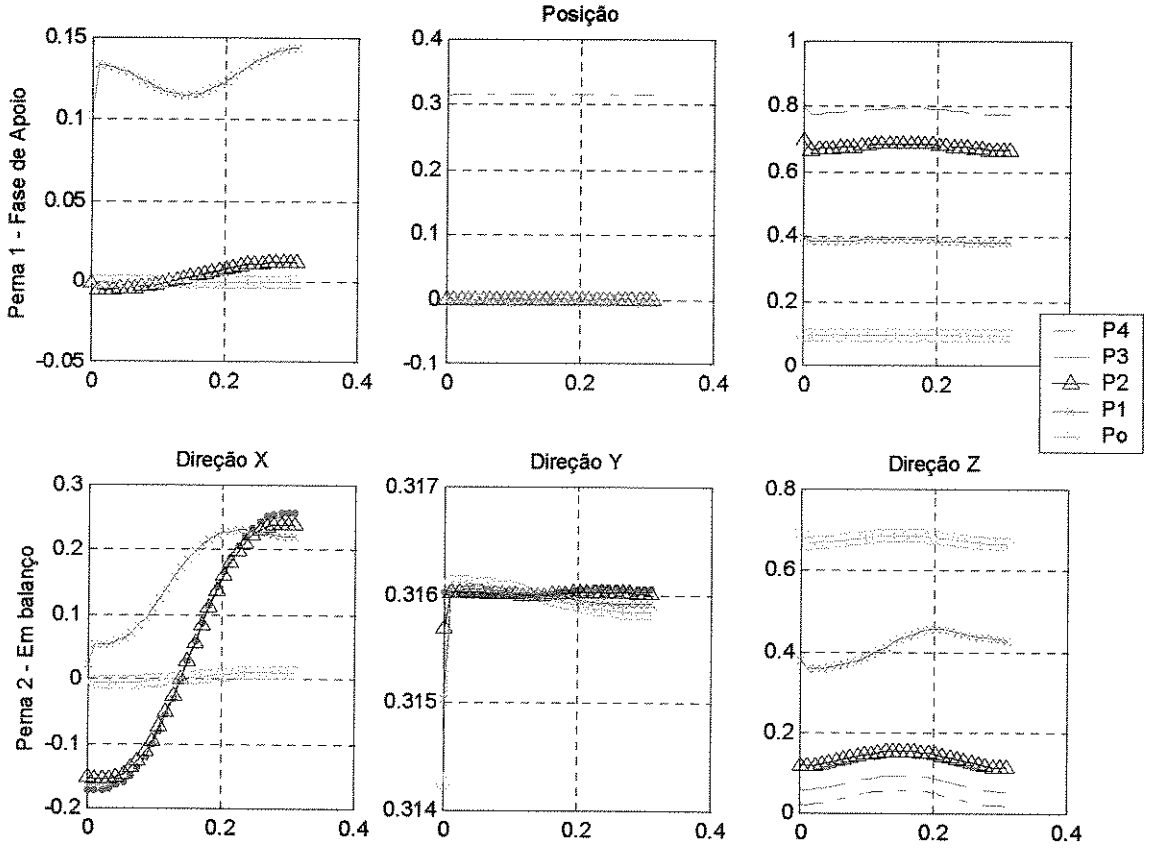


Figura 5.6 – Posição, m (plano cartesiano).

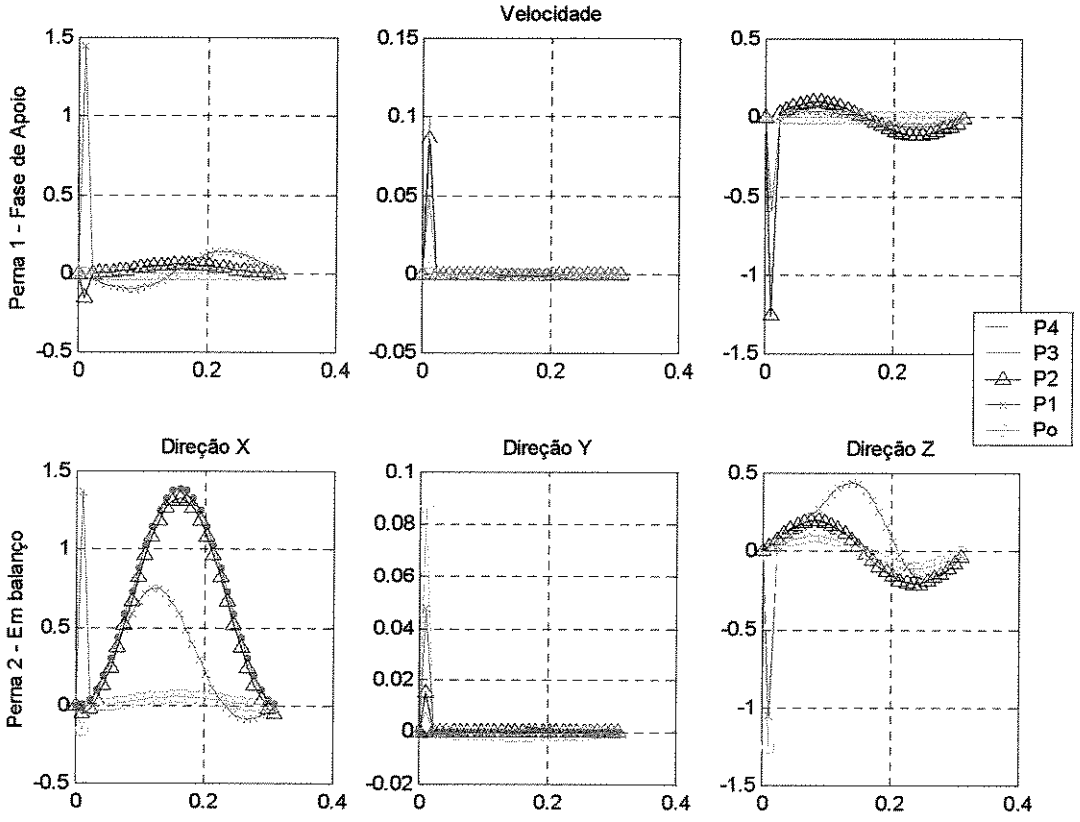


Figura 5.7 – Velocidade, m/s (plano cartesiano).

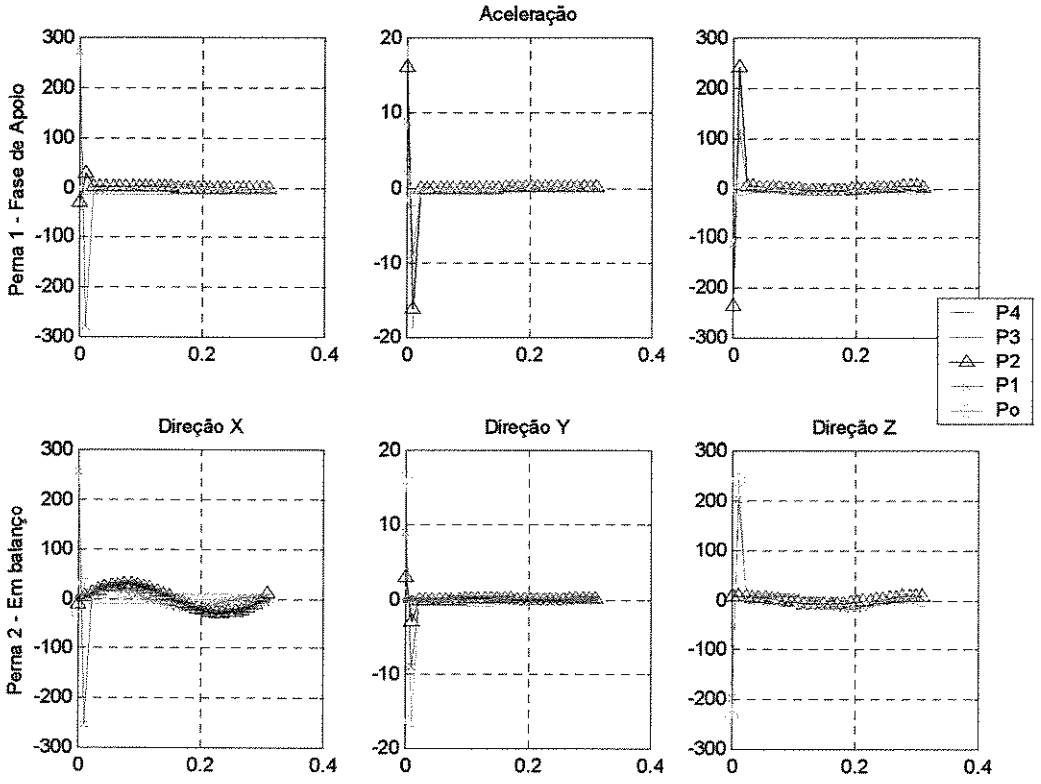


Figura 5.8 – Aceleração, m/s^2 (plano cartesiano).

Para a transição da postura básica A para a B, o ângulo relativo entre o pé em balanço e o solo foi subdividido em intervalos regulares (para essa simulação, foram utilizados 30 pontos). O robô bípede parte da posição especificada para o início da postura básica B. A posição final para o pé em balanço é automaticamente definida. A Figura 5.9 apresenta o resultado obtido.

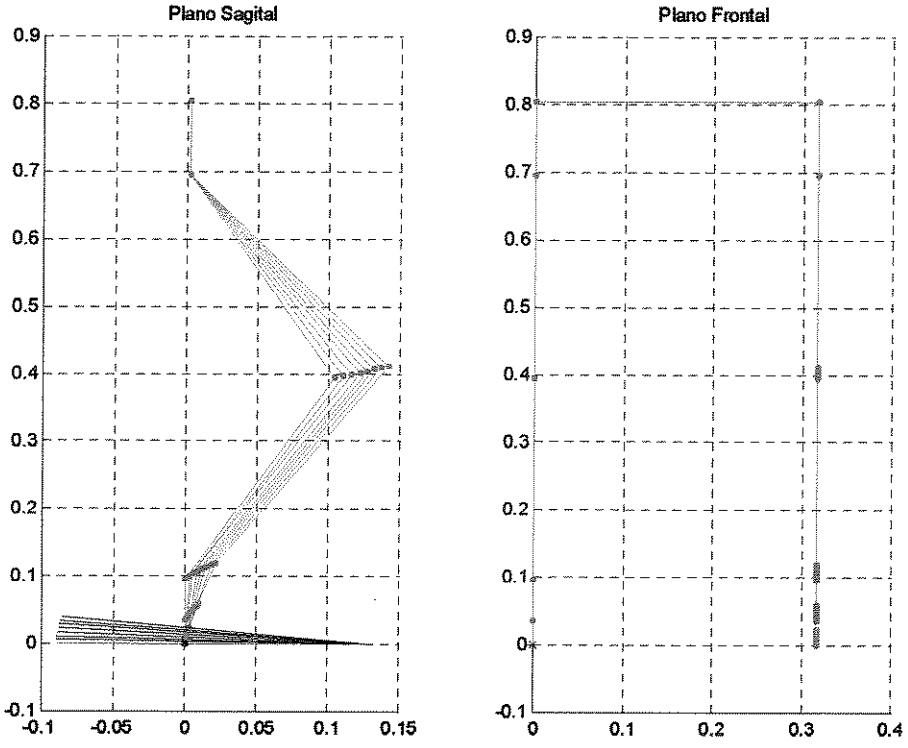


Figura 5.9 – Movimentos do robô nos planos sagital e frontal.

Para os casos apresentados acima, as posições angulares foram computadas por intermédio da cinemática e as velocidades, por intermédio de Jacobianos. Para gerar os sinais de referência para os membros inferiores, os vetores relativos às posições e às velocidades angulares foram montados seguindo a seqüência de uma marcha pré-estabelecida: deixar a postura básica A, passando pela postura B, e finalizando em D. Os resultados obtidos estão apresentados nas figuras que seguem, cujas informações estão dispostas segundo os mesmos critérios das anteriores.

A Figura 5.10 mostra as posições angulares[1], divididas em quatro grupos. À esquerda e no canto superior estão as trajetórias para a primeira e quinta junta. Logo abaixo, estão as trajetórias para a sexta e décima junta. Á direita e no canto superior estão as trajetórias para a segunda, terceira e quarta junta. Logo abaixo, estão as trajetórias relativas à sétima, oitava e nona junta.

[1] Ângulos relativos às juntas rotativas do robô bípede, definidas na Figura 2.3 (ver pág. 25)

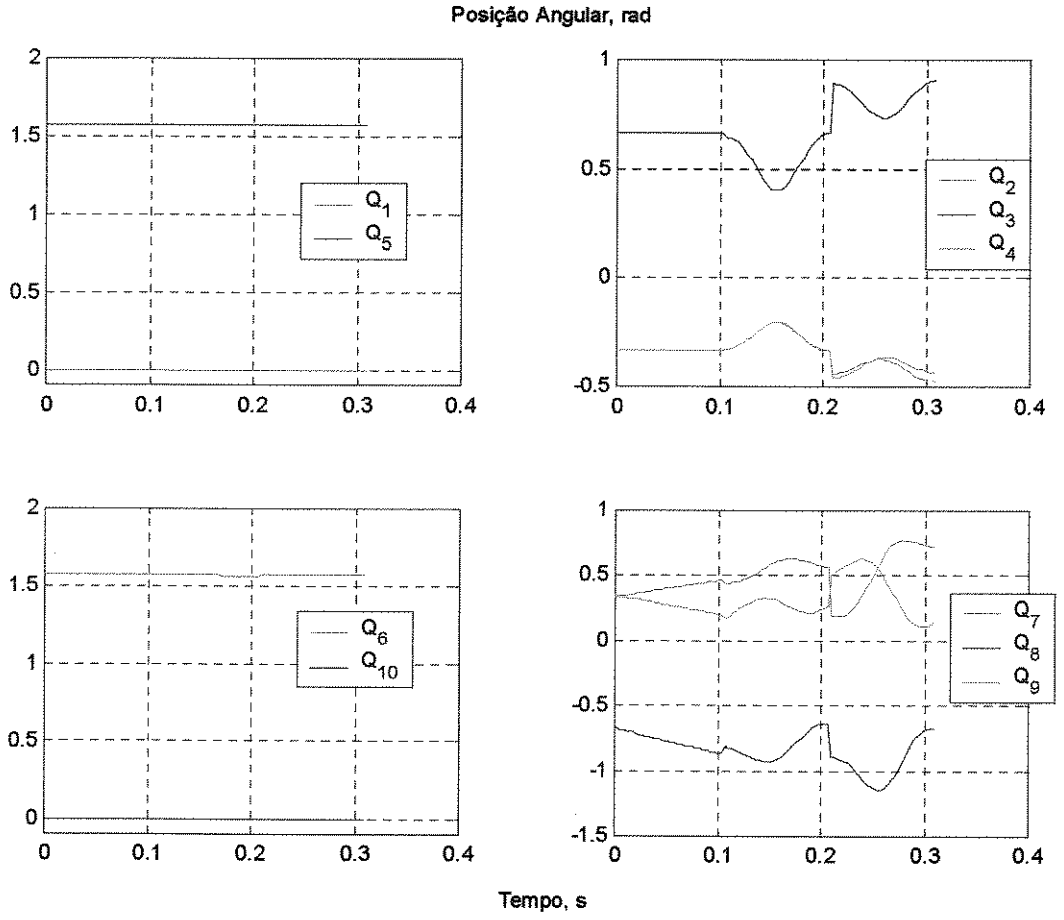


Figura 5.10 – Posições angulares relativas à transição da postura básica B para a D.

Seguindo as mesmas disposições no quadro, a Figura 5.11 apresenta as trajetórias de velocidades angulares.

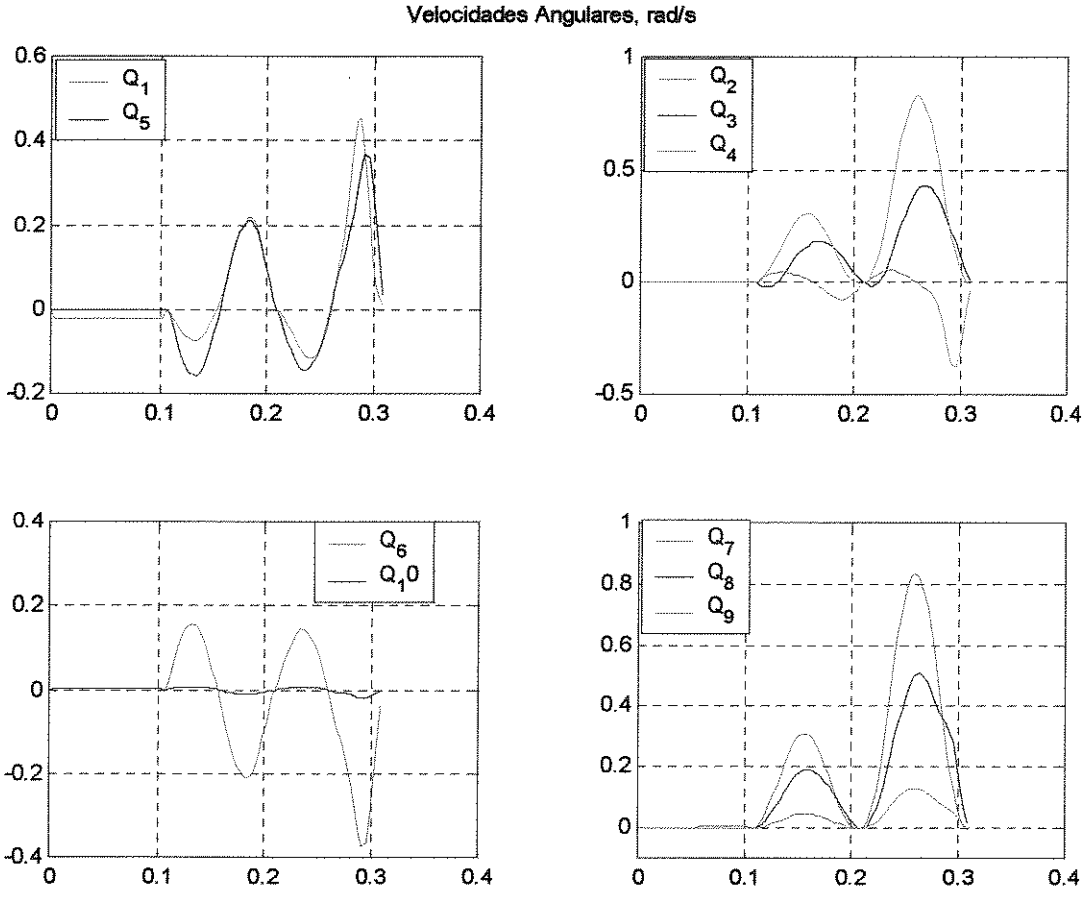


Figura 5.11 – Velocidades angulares relativas a transição da postura básica B para a D.

As Figuras 5.10 e 5.11 mostram os sinais de referência utilizados para acionar o sistema de controle adaptativo, apresentado no capítulo quarto. A Tabela 5.1 mostra os parâmetros associados aos modelos de referência do tronco e dos membros inferiores. A Tabela 5.2 mostra os parâmetros associados às redes RBF do tronco e dos membros inferiores.

Tabela 5.1 – Parâmetros associados aos modelos de referência.

Parâmetros	**Membros Inferiores**	**Tronco**
ω_n	150 $\mathbf{I}_{10\times10}$	50 $\mathbf{I}_{2\times2}$
ξ	0,3 $\mathbf{I}_{10\times10}$	0,3 $\mathbf{I}_{2\times2}$

Tabela 5.2 – Parâmetros associados às redes RBF.

Parâmetros	Membros Inferiores	Tronco
Π	$0{,}1\ \mathbf{I}_{10\times10}$	$0{,}1\ \mathbf{I}_{2\times2}$
κ	Cômputo Dinâmico	Cômputo Dinâmico

A Tabela 5.3 mostra os parâmetros utilizados para essa simulação realizada em Matlab/Simulink®, empregando um computador PC Pentium 4, com 512 Mb de memória e co-processador matemático.

Tabela 5.3 – Parâmetros utilizados na simulação.

Parâmetros	**Tempo inicial, s.**	0
	Tempo final, s.	0,3
Método de Integração	***Runge-Kutta***	rk45
	Comprimento do passo	variável
	Tolerância relativa	10^{-8}
	Tolerância absoluta	Auto

As Figuras 5.12 e 5.13 mostram as posições e as velocidades angulares geradas pelo modelo de referência, associado aos membros inferiores, respectivamente. Comparando os gráficos de posições com os correspondentes gráficos mostrado na Figura 5.10, é possível notar pequenas oscilações típicas de sistemas de segunda ordem, em torno dos valores desejados. Os correspondentes gráficos de velocidades apresentam oscilações maiores, principalmente para as articulações de ambas as pernas, indicando que o ganho utilizado pode ser diminuído.

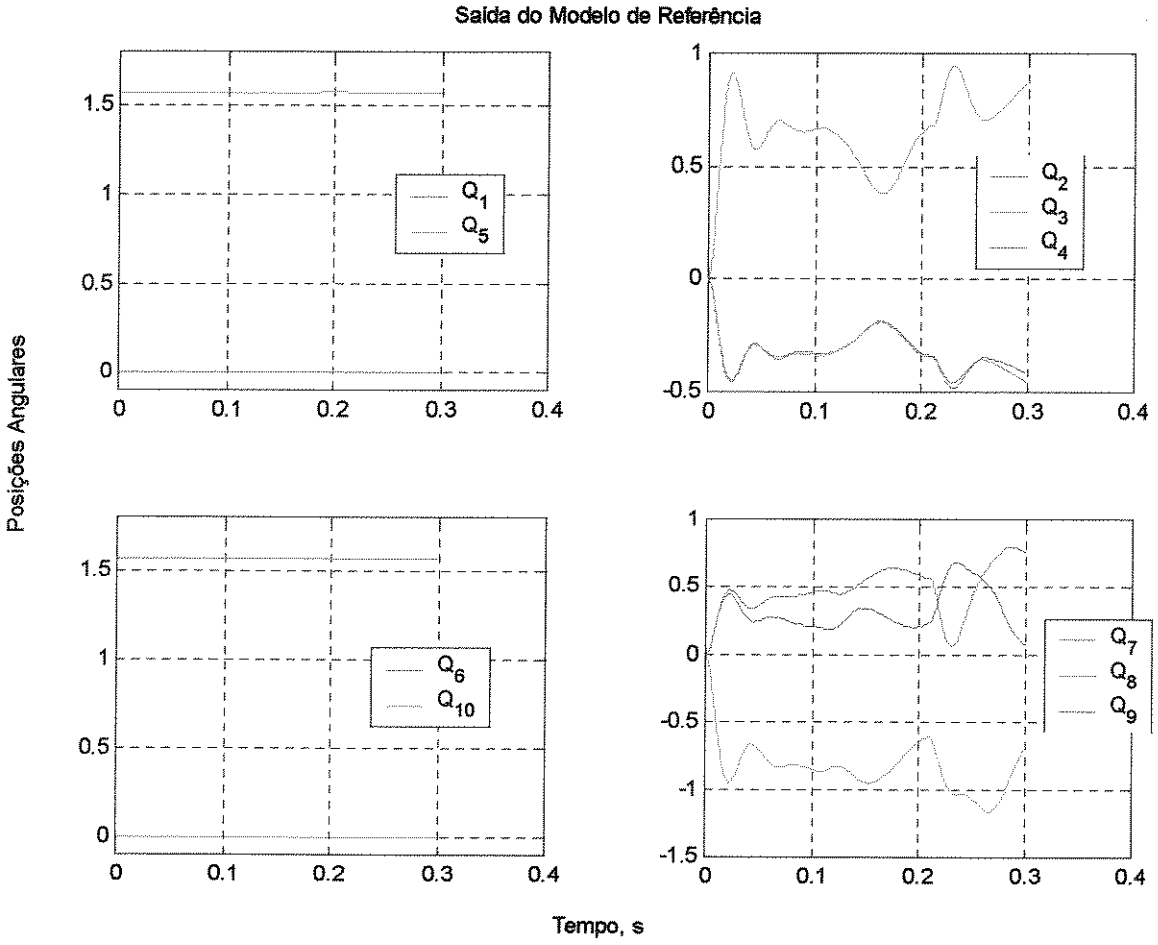


Figura 5.12 – Saída do modelo de referência: posições angulares, rad.

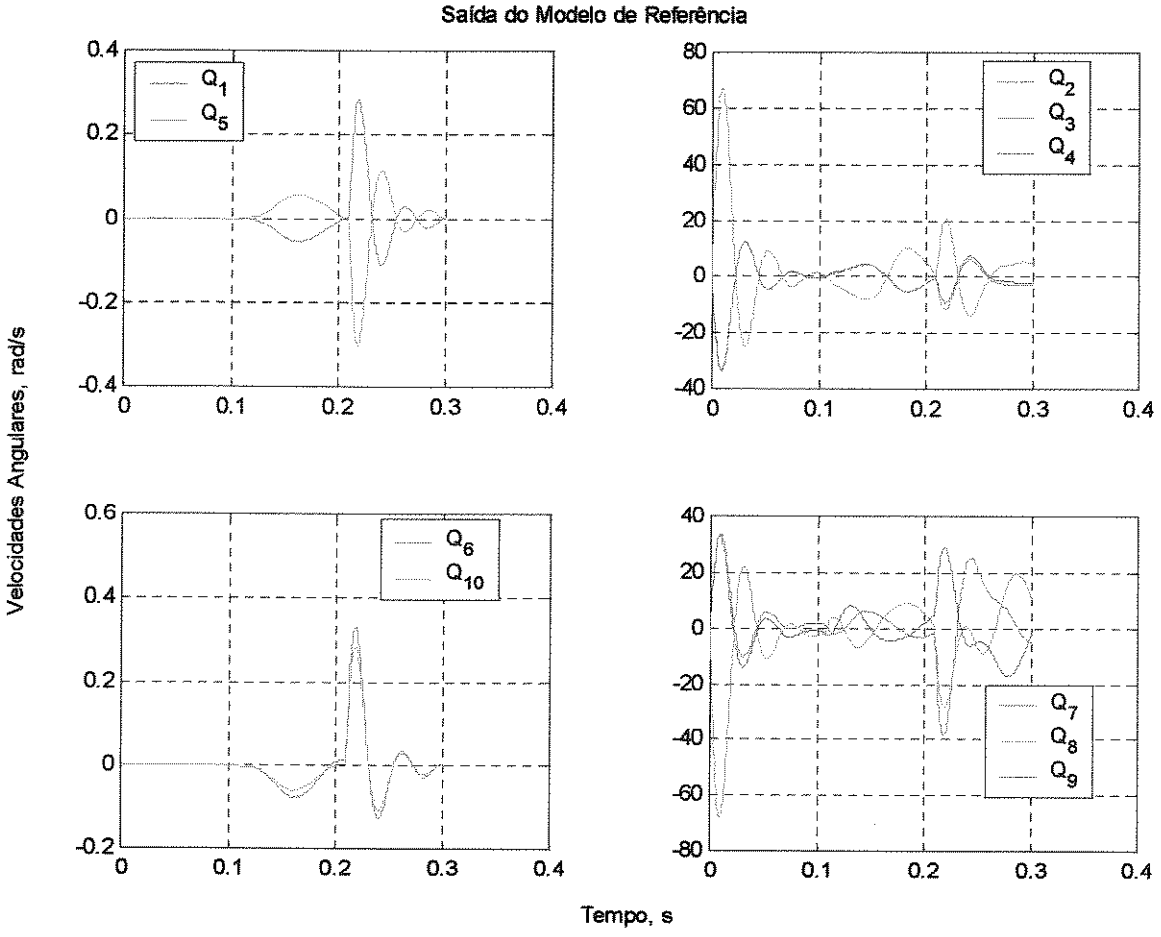


Figura 5.13 – Saída do modelo de referência: velocidades angulares, rad/s.

As Figuras 5.14 e 5.15 mostram as posições e velocidades angulares correspondentes às articulações dos membros inferiores. Essas variáveis rastrearam os correspondentes sinais mostrados nas Figuras 5.12 e 5.13, os valores de erros serão mostrados mais adiante.

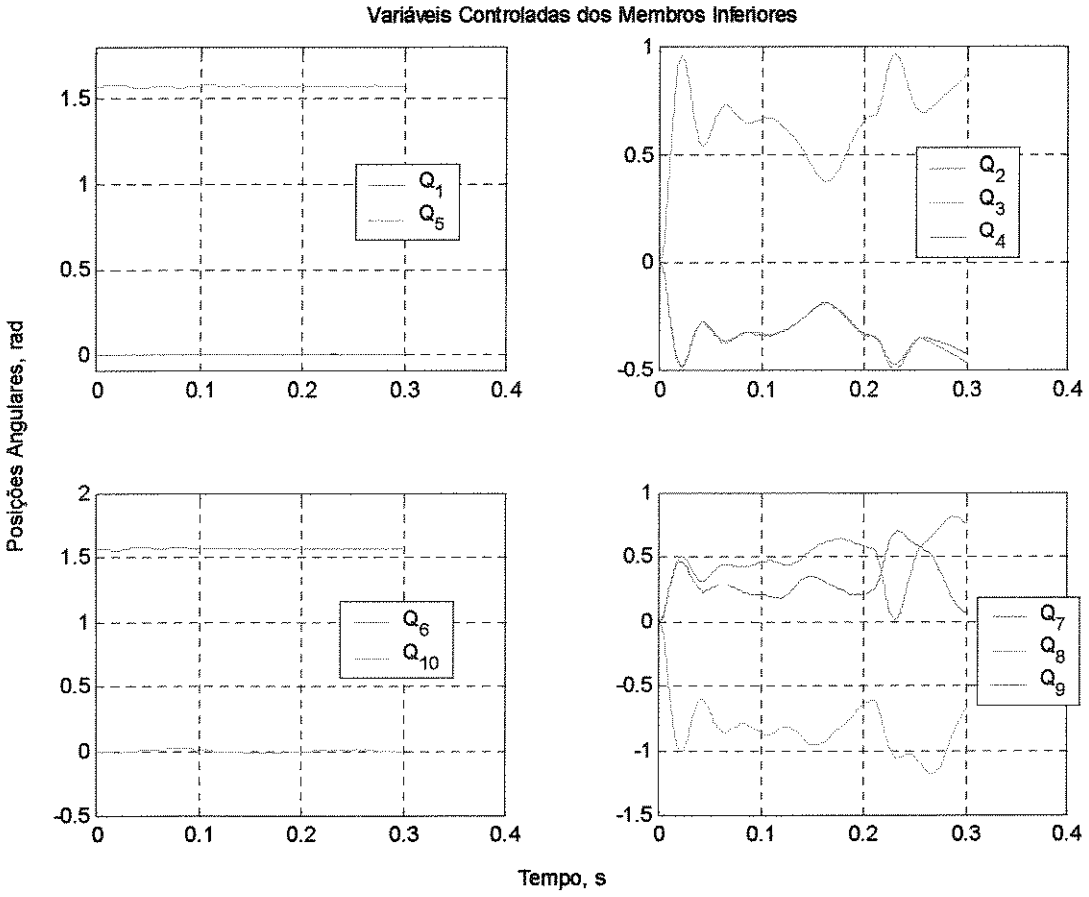


Figura 5.14 – Posições Angulares, saídas controladas dos membros inferiores.

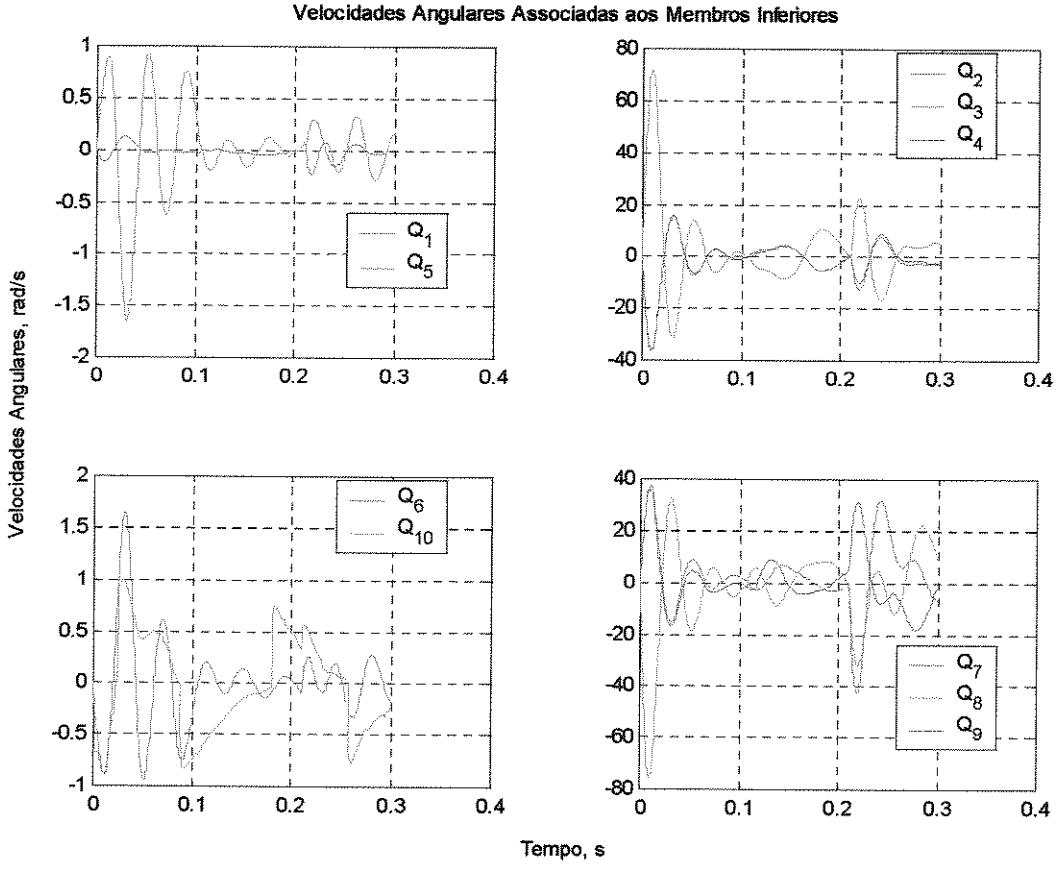


Figura 5.15 – Velocidades Angulares, saída controlada dos membros inferiores.

A Figura 5.16 mostra as trajetórias para posições e velocidades angulares gerados pelo modelo de referência do tronco. A rede neural recorrente resolve o problema de posicionamento do tronco (Capítulo 3) e envia sinais de posições angulares ao modelo de referência do tronco. A Figura 5.17 mostra as variáveis controladas do tronco

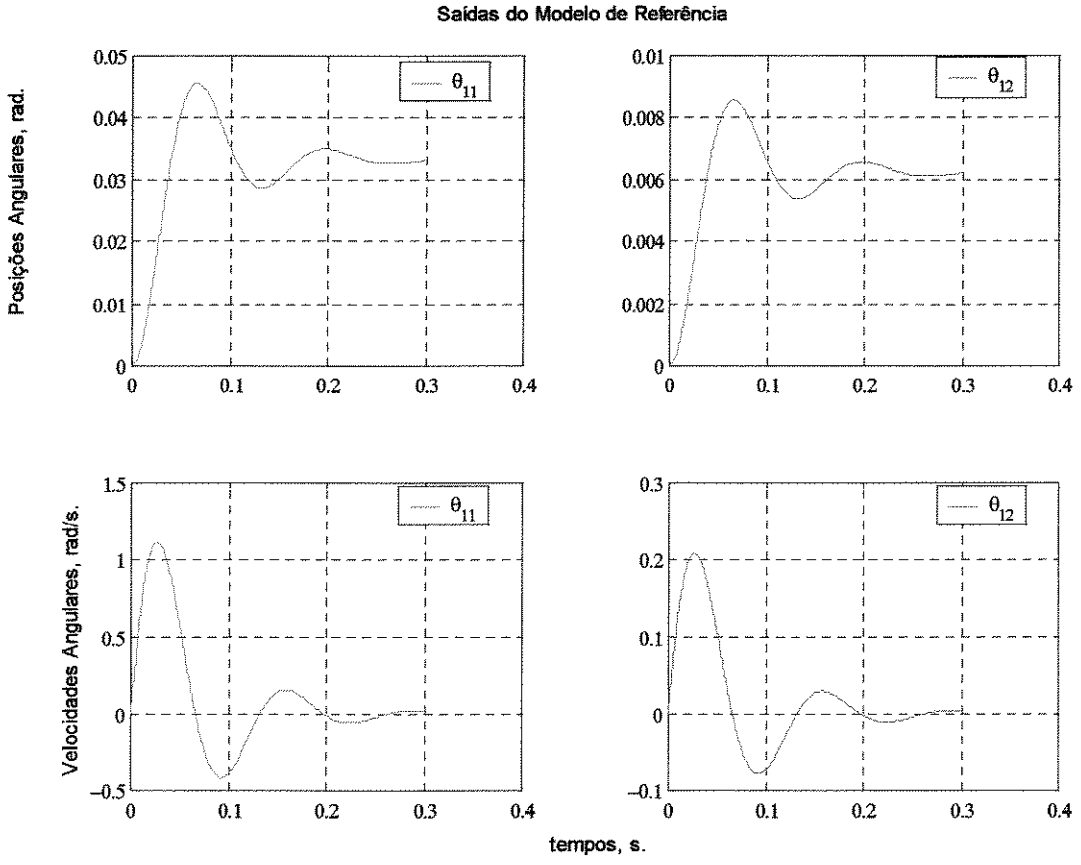


Figura 5.16 – Sinais de referências gerados pelo modelo de referência.

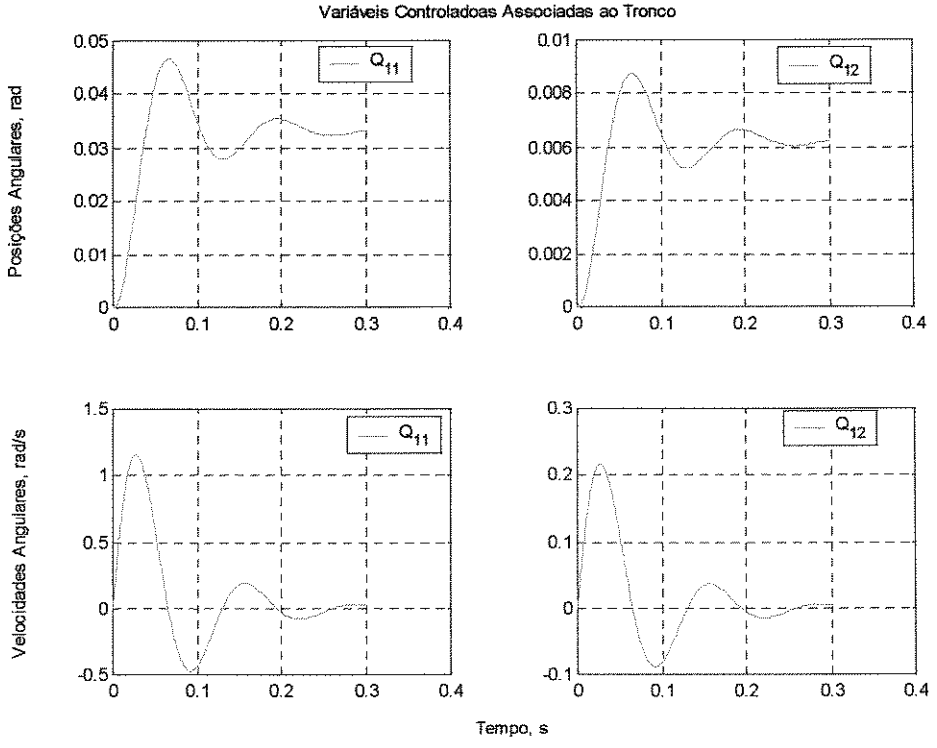


Figura 5.17 – Variáveis controladas, associadas ao tronco.

A Figura 5.18 mostra os erros de rastreamento cometidos pelo sistema de controle associado ao tronco. A posição angular θ_{11} apresenta valores de erro bastante baixo, limitado a $\pm$ $\mathbf{1{,}5\times10^{-3}}$ rad. Contudo, mantém um comportamento oscilatório, exponencialmente amortecido. De forma análoga, a variável θ_{12} está limitada a $\pm$ $\mathbf{3\times10^{-4}}$ rad, apresentando um comportamento oscilatório e exponencialmente amortecido. As correspondentes velocidades angulares apresentam comportamentos similares, cujos desvios estão na casa decimal.

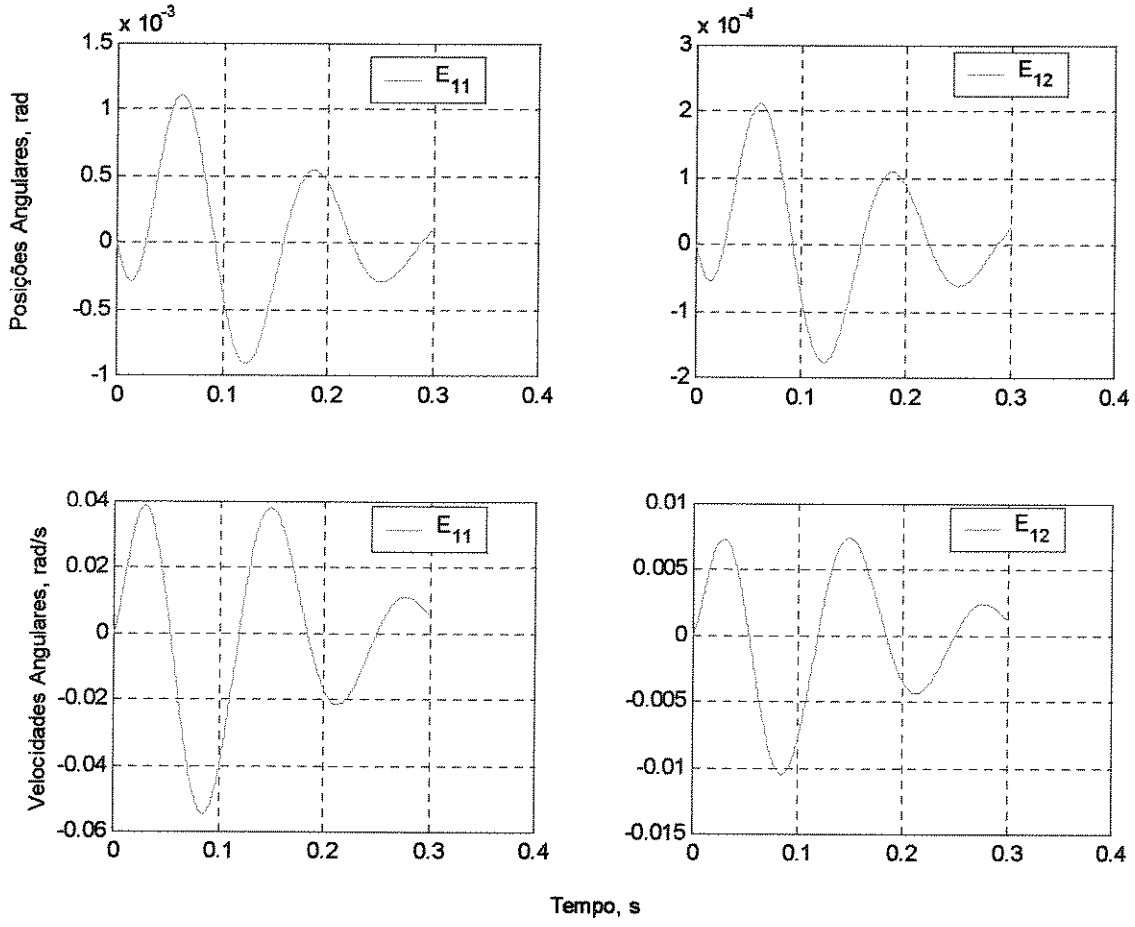


Figura 5.18 – Erro de rastreamento, associado ao modelo do tronco.

As Figuras 5.19 e 5.20 apresentam os erros de rastreamento correspondentes à posição e à velocidade, associados ao modelo dos membros inferiores, respectivamente. Erros de posição estão limitados a ± **0,06** rad, atingindo a faixa de ± **0,02** rad em aproximadamente **0,15** s. Comportamento similar é verificado para os erros correspondentes à velocidade, com valores elevados na fase inicial (≈ **0,1** s.) e atingindo valores em torno de ± **5** rad/s, após **0,1** s.

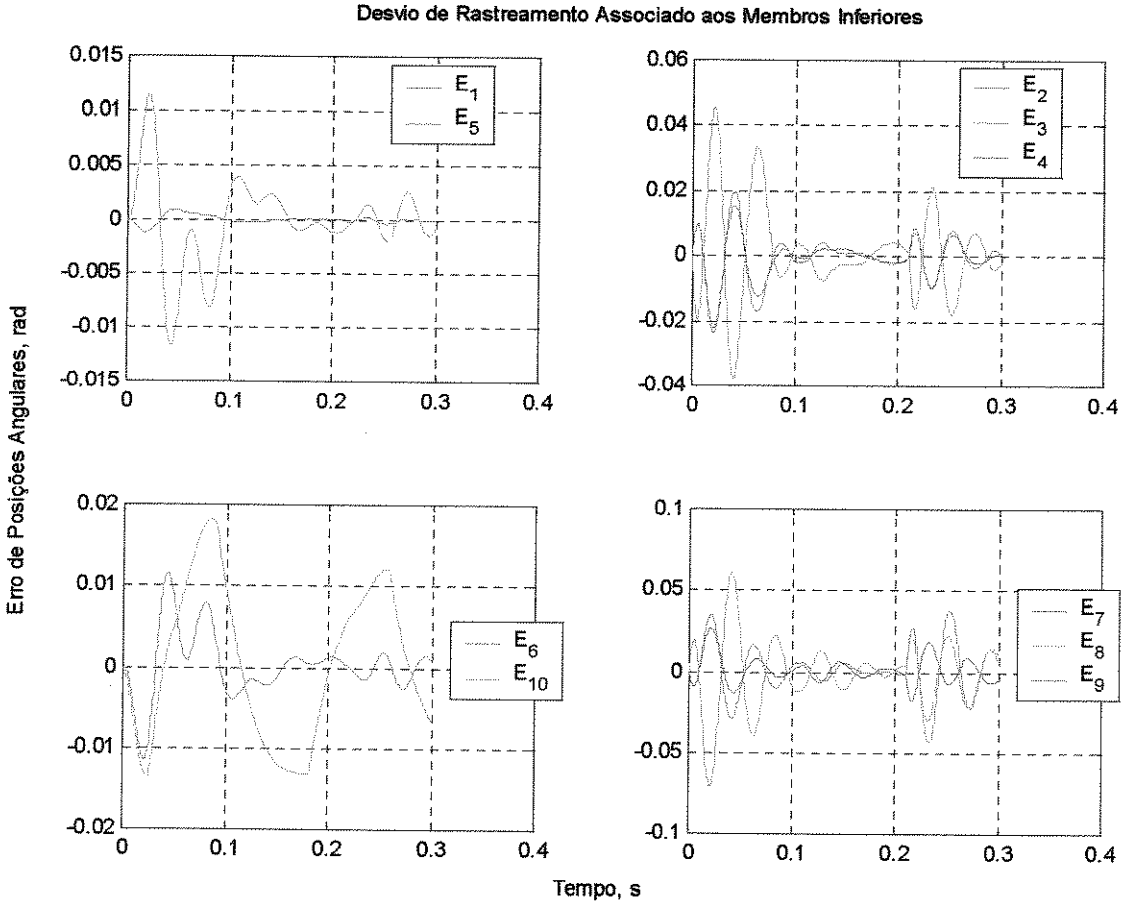


Figura 5.19 – Erro de rastreamento, associado aos membros inferiores.

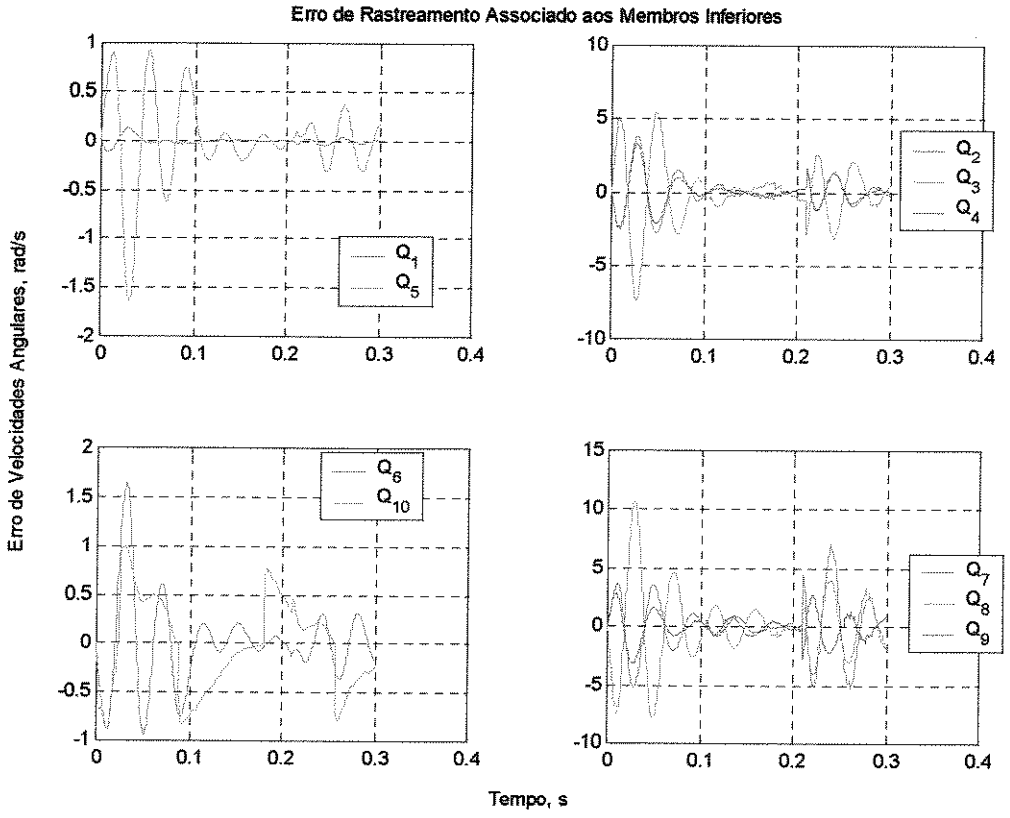


Figura 5.20 – Erro de rastreamento, associado aos membros inferiores.

A Figura 5.21 ilustra os valores de torques, computados de acordo com as Equações (3.28), para a marcha planejada sem a utilização do tronco. Os valores de torques nas direções X e Y são elevados (em virtude dos requisitos de marcha impostos) e causam rotações no pé de apoio, levando o robô bípede a cair. Portanto, sem a efetiva utilização do tronco não é possível garantir a estabilidade postural para essa marcha planejada.

A Figura 5.22 ilustra os valores de torques, computados de acordo com as Equações (3.28), para a marcha planejada com a utilização do tronco. Os valores de torques nas direções X e Y são fortemente atenuados e minimizam as rotações no pé de apoio, podendo, possivelmente, manter o balanço dinâmico. Portanto, o Gerador Automático de Trajetória para o Tronco (*GAT*) projetado e implementado permite gerar trajetórias de forma automática para o tronco a fim de garantir a estabilidade postural para essa marcha planejada.

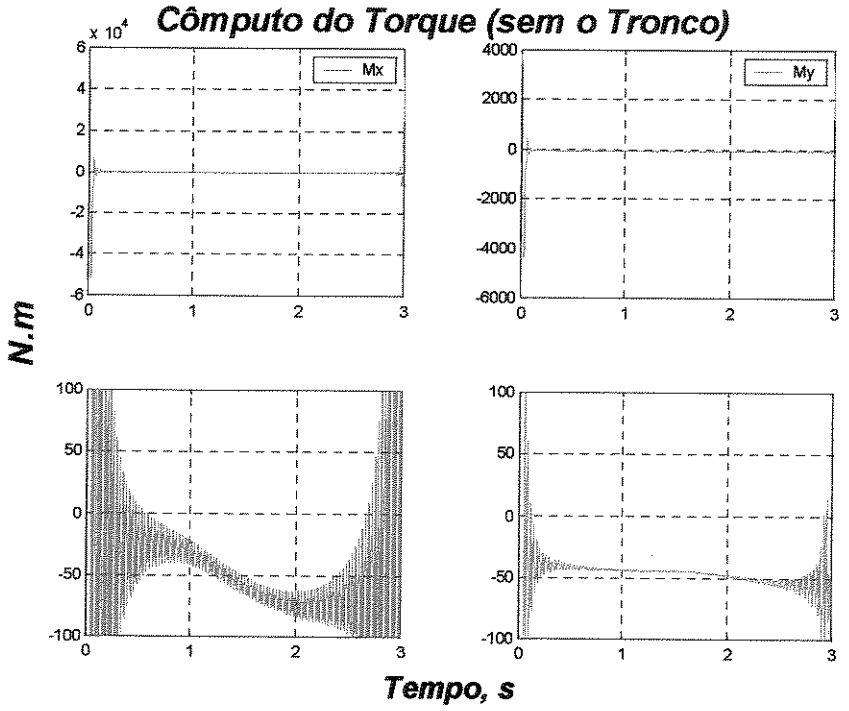


Figura 5.21 – Cômputo dos torques, sem a utilização do tronco.

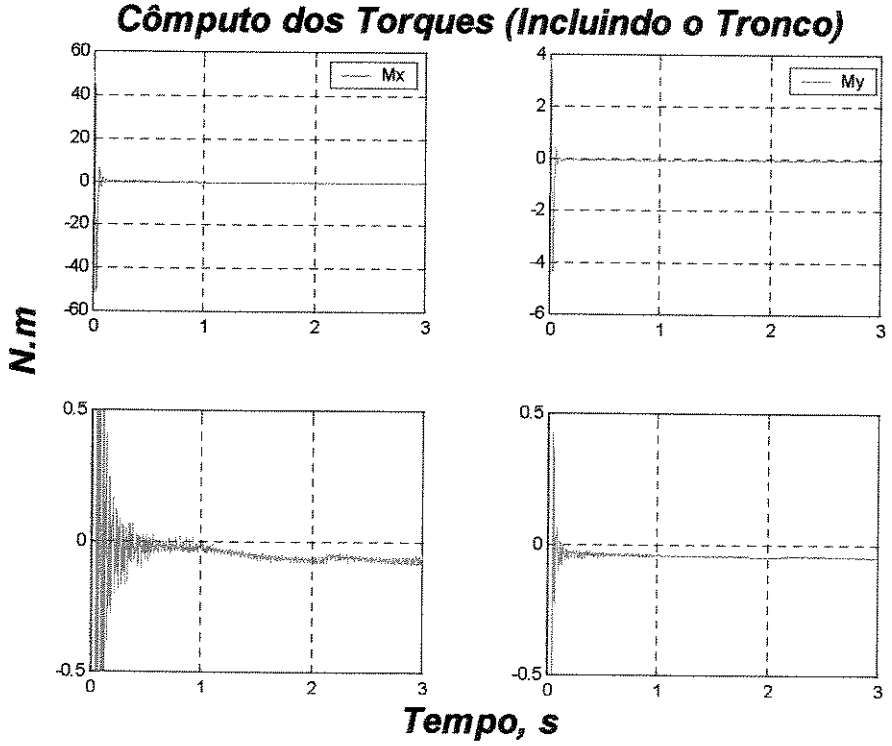


Figura 5.22 – Cômputo dos torques, incluindo a trajetória para o tronco.

Capítulo 6

Conclusões e Comentários Finais

Este trabalho objetivou contribuir à área de robôs bípedes que exploram o modo de andar dinâmico. Foi concebido um robô bípede dotado de tronco, composto por uma sucessão de elos rígidos interconectados por articulações rotativas, totalizando doze articulações que permitem posicionamentos no espaço tridimensional. O robô bípede foi modelado como dois subsistemas que interagem por meio de forças restritivas, atuantes na junção entre a pelve e o tronco. A modelagem simbólica foi automatizada, implementando-se o formalismo de *Newton-Euler* em ambiente do *Maple*®, oferecendo um modelador simbólico (**Anexo A**) que facilita a modelagem de sistemas multi-corpos.

Foram projetados e implementados sistemas de controle adaptativos para cada subsistema. A lei de controle possui termos do modelo dinâmico da planta, do modelo de referência e de incertezas. Redes neurais de base radial foram empregadas para identificação *on-line* dos termos dissipativos, que foram suprimidos do modelo dinâmico para gerar os sinais de entrada e de saída utilizados para treinar a referida rede. A rede RBF assegura a estabilidade em malha fechada no sentido de *Lyapunov*. Um gerador automático de marcha, adaptável às condições locais do terreno, foi concebido e implementado, funcionando perfeitamente apenas a partir da definição da velocidade de trajeto, do comprimento do passo e da altura máxima permitida para o pé em balanço. Com base na marcha planejada, a trajetória para o tronco é determinada por uma rede

neural MLP, integrada à malha de controle, permitindo atualizar o posicionamento angular do tronco a partir da evolução temporal dos membros inferiores do robô bípede. O sistema de controle e o gerador automático de trajetórias para o tronco compõem os mecanismos adaptativos desenvolvidos para solucionar a andadura dinâmica.

Foi planejada uma marcha com requisitos similares à marcha humana, com velocidade de trajeto de **2** km/h. A partir dos requisitos impostos, o Gerador Automático de Marcha (*GAM*) foi empregado, obtendo-se a marcha descrita nas Figuras 5.1 a 5.9. Empregando a cinemática inversa foram computados os correspondentes deslocamentos angulares e as velocidades angulares, por meio de Jacobianos, como mostram as Figuras 5.10 e 5.11. As posições e velocidades angulares foram gravadas em um arquivo e serviram para acionar o sistema de controle integrado.

Os parâmetros associados aos sistemas de controle integrados, obtidos por meios heurísticos, totalizam 26 valores para o modelo de referência e mais 24 valores para as redes RBF. Por simplificação, foram adotados valores idênticos para todos os graus de liberdade, descritos nas Tabelas 5.1 e 5.2. A Tabela 5.3 mostra os parâmetros utilizados na simulação.

A Figura 5.12 mostra as saídas do modelo de referência para os membros inferiores, que se assemelham às saídas desejadas (marcha planejada). Para rastrear os sinais de velocidades associados às articulações das pernas foi aplicado um melhor ajuste aos parâmetros do controlador para esses graus de liberdade, como mostra a Figura 5.13.

A Figura 5.14 mostra as posições angulares associadas aos membros inferiores. A Figura 5.19 indica que os erros de rastreamento são máximos no início do movimento ($\approx$ **0,05** s) e após $\approx$ **0,1** s estão bem próximos de zero. Do mesmo modo, os valores de erro associados às velocidades diminuem rapidamente, após aproximadamente **0,1** s.

O sistema de controle integrado apresenta um comportamento estável e, além de rastrear sinais de referência para os membros inferiores e para o tronco, permite rejeitar as perturbações decorrentes do acoplamento entre o tronco e os membros inferiores.

Nossas sugestões para a continuação do trabalho e aprofundamento da pesquisa são:

- Determinar os parâmetros do controlador por intermédio de técnicas de otimização, visando diminuir o erro de rastreamento;

- Projetar um sistema autônomo (possivelmente utilizando redes neurais) que permita integrar um gerador automático de marcha à malha de controle, similar ao gerador automático de trajetórias para o tronco;

- Projetar um sistema de controle hierárquico para supervisionar e integrar as ações de controle projetadas para os subsistemas: tronco e membros inferiores;

- Tratar o robô bípede com características cinemáticas alternadas entre cadeia fechada e cadeia aberta e vice-versa, incluindo a dinâmica de contato;

- Otimizar os códigos computacionais objetivando implementações em *tempo real*;

- Otimizar as dimensões das redes neurais projetadas;

- Projetar um sistema robótico, especificando sensores, microprocessadores, atuadores, redutores, etc ao modelo mecânico aqui desenvolvido;

- Implementar um robô bípede dotado de tronco para testar as estratégias aqui propostas.

Referências Bibliográficas

Adade Filho, Alberto. Fundamentos de robótica: Cinemática, dinâmica e controle de manipuladores robóticos. CTA, ITA, São José dos Campos – SP, 1992.

Antonelli, G.; Chiaverine, S. and Fusco, G. *An algorithm for on-line inverse kinematics with path tracking capability under velocity and acceleration contrains.* Proceedings of the 39th IEEE Conference on Decision and Control. Sydney, Austrália. December, 2000.

Chen, S., Cowan, C. F. N., and Grant, P. M. Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks. IEEE Transactions on Neural Networks, Vol. 2, No. 2, March 1991.

Cheng, M.-Y. and Lin, C.-S. Genetic Algorithm for control design of biped locomotion. Journal of robotic system 14(5), 365-373 (1997).

Chevallereau, C., Acustin Y., and Formal'sky A. Optimal Walking Trajectories for a Biped. IEEE, 1999.

Choi, Sang-ho, Choi, Young-ha, and Kim, Jin-Geol. *Optimal Walking Trajectory Generation for a Biped Robot Using Genetic Algorithm.* Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System, 1999.

Craig, John J. Introduction to robotics: mechanics and control. Addison-Wesley Longman. 2nd ed, 1995. 450 p.

Dasgupta, A., and Nakamura, Y. *Making Feasible Walking Motion of Humanoid Robots From Human Motion Capture Data.* Proceedings of the IEEE International Conference on Robotics & Automation. Detroit, Michigan. May, 1999.

Drummond, A. C., Oliveira, K. C. e Bauchspiess, A. *Estudo de controle de pêndulo invertido sobre carro utilizando rede neural de base radial.* Proceedings of the IV Brazilian Conference on Neural Networks, July 20-22, 1999 – ITA, São José dos Campos- SP – Brazil.

Fu, K. S.; Gonzáles, R. C. and Lee, C. S. G. *Robotics: control, sensing, vision and intelligence.* McGraw-Hill, 1988.

GE, S. S., Lee, T. H. and Harris, C. J. Adaptive Neural Networks Control of Robotic Manipulator. World Scientific Series in Robotics and Intelligent Systems, Vol. 19, 1998. 381p.

Geddes, K., Monagan, M. B. and Springer-Velarg. Maple V Programming Guide. Ed. Springer-Verlag NY. 1997. 379p.

Goddard, R., Zheng, Y. F., and Hermami, H. *Control of the Heel-off to Toe-off Motion of a Dynamic Biped Gait.* IEEE Transactions on Systems, Man, and Cybernetics, Vol. 22, No. 1. January/February 1992.

Gonçalves, J. B. and Zampieri, D. E. *Adaptive RBF Neural Network for a Biped Walking Machine.* Proceedings of the 17th International Congress of Mechanical Engineering, São Paulo, SP, Brazil, 2003.

Gonçalves, J. B. and Zampieri, D. E. *Integrated control to the biped walking robot.* Proceedings of the XI DINAME, 28th February - 4th March, 2005. Ouro Preto, MG, Brazil. Edited by D.A. Rade and V. Steffen Jr.

Gonçalves, J. B. and Zampieri, D. E. *Recurrent Neural Network Approaches for Biped Walking Robot based on Zero-Moment Point Criterion.* J. of the Braz. Soc. Mechanical Sciences, Vol. XXV, January 2003.

Goswami, A. *Postural Stability of Biped Robots and the Foot-Rotation Indicator (FRI) Point.* The International Journal of Robotics Research, Vol. 18, No. 6, June 1999, pp. 523-533.

Hasegawa, Y., Arakawa, T., and Fukuda, T. *Trajectory generation for biped locomotion robot.* Mechatronics 10 (2000) 67-89. Elsevier Science Ltd.

Haykin, Simon. Redes neurais: princípios e prática. 2001, Bookman. 900 p.

Huang, Q., Yokoi, K., Kajita, S., Kaneko, K., Arai, H., Koyachi, N., Tanie, K. *Planning Walking Patterns for a Biped Robot.* IEEE Transactions on Robotics and Automation, Vol. 17, No. 3, June 2001.

Kajita, S., and Tani, K. *Study of dynamic biped locomotion on rugged terrain: Derivation and application of the linear inverted pendulum mode.* Proceedings of the IEEE International Conference on Robotics and Automation. Sacramento, CA. pp. 1405-1411, 1991.

Kuo, A. D., Stabilization of lateral motion in passive dynamic walking. The International Journal of Robotics Research Vol. 18, No. 9, Sept. 1999, pp. 917-930.

Li, Qinghua, Takanishi, Atsuo, and Kato, Ichiro. Learning Control of Compensative Trunk Motion for biped walking robot based on ZMP stability criterion. Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robot and Systems. Raleigh, NC. July 7-10, 1992.

Marchese, S., Muscato, G., and Virk, G. S. *Dynamic Stable trajectory Synthesis for a Biped Robot during the Single-Support Phase.* Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics. 8-12 July 2001, Corno, Italy.

Munzir, S., Mohamed, H. M., Abdulmuin, M. Z., A new approach for modeling and control of MIMO nonlinear systems. IEEE, 2000.

Park, J. H., and Kim K. D. *Biped robot walking using gravity-compensated inverted pendulum mode and computed torque control.* Proceedings of the IEEE International Conference on Robotics and Automation. Lueven, Belgium. 1998.

Predabon, E. and Bocchese, C. SolidWorks 2004 – Projeto e Desenvolvimento. 2003. Ed. Érica. 408 p.

Schilling, Robert J. Fundamentals of robotics: analysis and control. 1990, Prentice-Hall. 425 p.

Shibata, M., and Natori, T. *Impact Force Reduction for Biped Robot Based on Decoupling COG Control Scheme*. IEEE AMC. Nagoya. 2000.

Shih, Ching-Long, Gruver, William A., and Lee, Tsu-Tian. Inverse kinematics and inverse dynamics for control of a biped walking machine. Journal of Robotics Systems 10(4), 531-555 (1993).

Silva, F. M., and Machado, J. A. T. Kinematic aspects of robotics biped locomotion systems (Technical paper). Modern University, Dept. of Control and Automation, Portugal. 2001.

Slotine, J. E., and Li, W. Applied Nonlinear Control. 1991. Prentice-Hall. 461p.

Sugihara, T., Nakamura, Y., and Inoue, Hirochika. Realtime humanoid motion generation through ZMP manipulation based on Inverted Pendulum Control. Proceedings of the 2002 IEEE International Conference on Robotics & Automation. Washington, DC. May 2002.

Takahashi, T. and Kawamura, A. *The high-speed numerical calculation method for the on-line inverse kinematics of redundant degree of freedom manipulators.* AMC2000-Nagoya. 2000 IEEE.

Takanishi, A. Robot Biped Walking with Trunk Motion. Proceedings of ARW on Robotics and Biological System, 1989.

Von Zuben, F. J. e Castro, L. N. *Otmized training techniques for feedforward neural networks.* Reportagem Tècnica, DCA/FEEC/UNICAMP, 1998.

Von Zuben, F. J. Notas de Aula do curso IA353 – Redes Neurais. 2000. UNICAMP/FEEC/DCA. Acesso em Março de 2000.

Yamaguchi, Jin-ichi, Takanishi, A., and Kato, I. Development of a biped walking robot compensating for three-axis moment by trunk motion. Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems, Yokohama, Japan. July, 26-30, 1993.

ANEXO A

Modelagem Automática de Sistemas

O *software Maple® V R4* foi utilizado para a implementação computacional do modelo dinâmico (2.48), automatizando o processo de modelagem simbólica. Para executar o programa *NEROBOT* é necessário escrever um arquivo (sem formatação e sem extensão) para fornecer a entrada de dados, contendo os dados físicos do sistema, conforme Tabela A1.

Tabela A1 – Arquivo de Dados

```
# ------------------------------------------------------------------------------
#   Entrada de Dados Relativa ao Robô Bípede - Versão 2003
# ------------------------------------------------------------------------------
# Created by: João Bosco Gonçalves
# Date        : 23/07/2k3
# Version     : 1.0
# ------------------------------------------------------------------------------
restart:
# ------------------------------------------------------------------------------
# Parâmetros dos Elos, metros
# ------------------------------------------------------------------------------
 a[o] := 38.6e-3:  a[1] := 61e-3:   a[2] := 316e-3:   a[3] := 316e-3:
```

```
a[4] := 110e-3:  a[5] := 316e-3:

# ------------------------------------------------------------------------------------------
# Parâmetros de Denavit-Hatemberg:
# DH := [Teta(i),Alfa(i),A(i),D(i)]
# ------------------------------------------------------------------------------------------
DH := matrix([[theta[1]  , pi/2,  a[1] , 0],
              [theta[2]  ,   0 ,  a[2] , 0],
              [theta[3]  ,   0 ,  a[3] , 0],
              [theta[4]  , -pi/2, a[4] , 0],
              [theta[5],       0,  a[5] , 0],
              [theta[6], -pi/2,   a[4], 0],
              [theta[7]  ,   0 ,  a[3] , 0],
              [theta[8]  ,   0 ,  a[2] , 0],
              [theta[9]  , pi/2 , a[1] , 0],
              [theta[10]  ,   0 ,  a[o] , 0]]);
# ------------------------------------------------------------------------------------------
# Parâmetros de Massa, kg
# Mass :=
#[Elo_11,Elo_21,Elo_31,Elo_41,Elo_5,Elo_42,Elo_32,Elo_22,Elo_12,Elo_o2,Elo_oo,Tronco]
# ------------------------------------------------------------------------------------------
Mass := [0.93,0.88,1.12,0.98,0.65,0.98,1.12,0.88,0.93,0.51,0.51,5.75]:
# ------------------------------------------------------------------------------------------
# Coordenadas Homogêneas do Centro de Massa, m
#       (1) indica o lado direito
#       (2) indica o lado esquerdo
# ------------------------------------------------------------------------------------------
Elo_o  := [-0.02,  0.00,-0.04,1]:
Elo_1  := [-0.01,  0.00, 0.00,1]:
Elo_2  := [-0.07,  0.00, 0.01,1]:
Elo_3  := [-0.09,  0.00,-0.01,1]:
```

```
Elo_4  := [-0.03,  0.00,-0.02,1]:
Elo_5  := [ 0.00,  0.00,-0.01,1]:
Elo_6  := [ 0.00,  0.03,-0.02,1]:
Elo_7  := [ 0.09,  0.00,-0.01,1]:
Elo_8  := [ 0.07,  0.00, 0.01,1]:
Elo_9  := [ 0.01,  0.00, 0.00,1]:
Elo_10 := [ 0.02,  0.00,-0.04,1]:
Tronco := [ 0.50,-0.02,-0.01,1]:
# ------------------------------------------------------------------------------------------
# Momentos de Inércia tomados no CoM do elo-i, kg*m^2
# MoI_Eloi := [Lxx,Lxy,Lxz,Lyx,Lyy,Lyz,Lzx,Lzy,Lzz]
# ------------------------------------------------------------------------------------------
MoI_Eo1 := [ 0.003, 0.000, 0.000, 0.000, 0.004, 0.000, 0.000, 0.000, 0.000]:
MoI_E11 := [ 0.003, 0.000, 0.000, 0.000, 0.001, 0.000, 0.000, 0.000, 0.003]:
MoI_E21 := [ 0.001, 0.000, 0.000, 0.000, 0.010, 0.000, 0.000, 0.000, 0.010]:
MoI_E31 := [ 0.002, 0.000, 0.000, 0.000, 0.011, 0.000, 0.000, 0.000, 0.010]:
MoI_E41 := [ 0.003, 0.000, 0.000, 0.000, 0.003, 0.000, 0.000, 0.000, 0.003]:
MoI_E51 := [ 0.008, 0.000, 0.000, 0.000, 0.002, 0.000, 0.000, 0.000, 0.006]:
MoI_E42 := [ 0.003, 0.000, 0.000, 0.000, 0.003, 0.000, 0.000, 0.000, 0.003]:
MoI_E32 := [ 0.002, 0.000, 0.000, 0.000, 0.011, 0.000, 0.000, 0.000, 0.010]:
MoI_E22 := [ 0.001, 0.000, 0.000, 0.000, 0.010, 0.000, 0.000, 0.000, 0.010]:
MoI_E12 := [ 0.003, 0.000, 0.000, 0.000, 0.001, 0.000, 0.000, 0.000, 0.003]:
MoI_Eo2 := [ 0.003, 0.000, 0.000, 0.000, 0.004, 0.000, 0.000, 0.000, 0.000]:
MoI_TR  := [ 0.000, 0.000, 0.000, 0.000, 0.500, 0.000, 0.000, 0.000, 0.500]:
#  Tab =
# [elos-i      massa-i      Ixx-i  Iyy-i  Izz-i  Ixy-i  Ixz-i  Iyz-i  x-i    y-i    z-i ]
Tab:= matrix([
[elo[1],Mass[1],MoI_E11[1],MoI_E11[5],MoI_E11[9],MoI_E11[2],MoI_E11[3],
MoI_E11[6],Elo_1[1],Elo_1[2],Elo_1[3]],
[elo[2],Mass[2],MoI_E21[1],MoI_E21[5],MoI_E21[9],MoI_E21[2],MoI_E21[3],
MoI_E21[6],Elo_2[1],Elo_2[2],Elo_2[3]],
```

```
[elo[3],Mass[3],MoI_E31[1],MoI_E31[5],MoI_E31[9],MoI_E31[2],MoI_E31[3],
MoI_E31[6],Elo_3[1],Elo_3[2],Elo_3[3]],
[elo[4],Mass[4],MoI_E41[1],MoI_E41[5],MoI_E41[9],MoI_E41[2],MoI_E41[3],
MoI_E41[6],Elo_4[1],Elo_4[2],Elo_4[3]],
[elo[5],Mass[5],MoI_E51[1],MoI_E51[5],MoI_E51[9],MoI_E51[2],MoI_E51[3],
MoI_E51[6],Elo_5[1],Elo_5[2],Elo_5[3]],
[elo[6],Mass[6],MoI_E42[1],MoI_E42[5],MoI_E42[9],MoI_E42[2],MoI_E42[3],
MoI_E42[6],Elo_6[1],Elo_6[2],Elo_6[3]],
[elo[7],Mass[7],MoI_E32[1],MoI_E32[5],MoI_E32[9],MoI_E32[2],MoI_E32[3],
MoI_E32[6],Elo_7[1],Elo_7[2],Elo_7[3]],
[elo[8],Mass[8],MoI_E22[1],MoI_E22[5],MoI_E22[9],MoI_E22[2],MoI_E22[3],
MoI_E22[6],Elo_8[1],Elo_8[2],Elo_8[3]],
[elo[9],Mass[9],MoI_E12[1],MoI_E12[5],MoI_E12[9],MoI_E12[2],MoI_E12[3],
MoI_E12[6],Elo_9[1],Elo_9[2],Elo_9[3]],
[elo[10],Mass[10],MoI_Eo2[1],MoI_Eo2[5],MoI_Eo2[9],MoI_Eo2[2],MoI_Eo2[3],
MoI_Eo2[6],Elo_10[1],Elo_10[2],Elo_10[3]],
[elo[o],Mass[11],MoI_Eo1[1],MoI_Eo1[5],MoI_Eo1[9],MoI_Eo1[2],MoI_Eo1[3],
MoI_Eo1[6],Elo_o[1],Elo_o[2],Elo_o[3]]]);
# ------------------------------------------------------------------------------
# Vetor Gravitacional, Eixo vertical: x
# ------------------------------------------------------------------------------
Grav := matrix(4,1,[-g[o],0,0,0]);
# ------------------------------------------------------------------------------
# Número de Elementos, exceto o tronco
# ------------------------------------------------------------------------------
NElos := 10:
# ------------------------------------------------------------------------------
# Vetor identificador de junta
# ------------------------------------------------------------------------------
TJ := matrix(NElos,1,1):  # 1-> junta revoluta e 0 -> junta prismática
# ------------------------------------------------------------------------------
```

```
# Vetor de forças generalizadas externas
# ------------------------------------------------------------------------------------------
Ftoos := matrix(3,1,[f[x],f[y],f[z]]):
Ttoos := matrix(3,1,[n[x],n[y],n[z]]):
```

Em seguida é necessário executar o programa *NEROBOT* digitando a seguinte instrução, no ambiente do *software Maple®V*:

read `c:\\caminho\\NEROBOT.`:

O modelo dinâmico resultante é gravado (em disco) e poderá ser manipulado por editores de texto, ou pelo próprio *Maple®V*.

A Tabela A2 mostra o programa *NEROBOT*, implementado em comandos do *software Maple® V R4*.

Tabela A2 – Programa *NEROBOT*.

```
# ------------------------------------------------------------------------------------------
restart:

# ------------------------------------------------------------------------------------------
# Carrega Biblioteca de Álgebra Linear.
  with(LinearAlgebra):
  with(linalg):

# ------------------------------------------------------------------------------------------
# Carrega Arquivo de DADOS  Relativo ao Robô Bípede.
  read `c:\\rb_modela\\rb03.`;

# ------------------------------------------------------------------------------------------
# Define os Vetores Posições, Velocidades e Acelerações.
```

```
Q     := matrix(NElos,1,0):          # Variáveis de Juntas
DQ    := matrix(NElos,1,0):          # Velocidade de Juntas
DDQ   := matrix(NElos,1,0):          # Aceleração de Juntas
DV    := matrix(3,(NElos+1),0):      # Aceleração Linear do Elo
W     := matrix(3,(NElos+1),0):      # Velocidade Angular do Elo
DW    := matrix(3,(NElos+1),0):      # Aceleração Angular do Elo


for v from 1 by 1 to NElos do
   # macro(q[v]=theta[v](t),dq[v]=dq[v](t),ddq[v]=ddq[v](t)):
   # Q[v,1]   := q[v]:
   DQ[v,1]   :=dq[v]:
   DDQ[v,1]  := ddq[v]:
 od:

# ------------------------------------------------------------------------------------------
# Define os Vetores de Torques (TQ) e de Forças (FO) externas

   TQ   := matrix(3,2,0): # Vetor de Torques
   FO   := matrix(3,2,0): # Vetor de Forças
# ------------------------------------------------------------------------------------------
# Condições Iniciais

      DV[1,1] := -Grav[1,1]:
      DV[2,1] := -Grav[2,1]:
      DV[3,1] := -Grav[3,1]:

      FO[1,2] := -Ftoos[1,1]:
      FO[2,2] := -Ftoos[2,1]:
      FO[3,2] := -Ftoos[3,1]:
```

```
        TQ[1,2] := -Ttoos[1,1]:
        TQ[2,2] := -Ttoos[2,1]:
        TQ[3,2] := -Ttoos[3,1]:

# ------------------------------------------------------------------------------------------------
# Matriz de Transformação Homogênea

MTH := (a,alpha,d,theta) -> matrix(4,4,[cos(theta),
                                        -sin(theta)*cos(alpha),sin(theta)*sin(alpha),
                                       a*cos(theta),sin(theta),cos(theta)*cos(alpha),
                                        -cos(theta)*sin(alpha), a*sin(theta),0,
                                         sin(alpha),cos(alpha),d,0,0,0,1]):

# ------------------------------------------------------------------------------------------------
# Pseudo Matriz de Inércia, obtida a partir do SC do CoM, alinhado com o SC do elo

 PMI := (Tab,k) -> matrix(3,3,[Tab[k,3],-Tab[k,6],-Tab[k,7],
                               -Tab[k,6],Tab[k,4],-Tab[k,8],
                               -Tab[k,7],-Tab[k,8],Tab[k,5]]):

# ------------------------------------------------------------------------------------------------
# Produto Vetorial V = P X Q

PVT:= (P,Q) -> matrix(3,1,[P[2,1]*Q[3,1]-P[3,1]*Q[2,1],P[3,1]*Q[1,1]-P[1,1]*Q[3,1],
                           P[1,1]*Q[2,1]-P[2,1]*Q[1,1]]):

# ------------------------------------------------------------------------------------------------
#Procedimento para Computar das equações diretas

PCED := proc(NElos::integer,TJ::matrix,Q::matrix,DQ::matrix,DDQ::matrix,DH::matrix)
```

```
        local i,j,zdq,zddq,Pk,Wk,WDQ,AI,S,T,DWk,WDQ2,WDQ3,DWDQ,k;

global W,DW,DV:

T  := matrix(4,4,[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]):  # MTH Inicial, T = I
AI  := matrix(4,4,[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]):
Pk := matrix(3,1,0):


for k from 2 by 1 to (NElos+1) do

   T := simplify(AI):
   Pk[1,1] := T[1,4]:
   Pk[2,1] := T[2,4]:
   Pk[3,1] := T[3,4]:

   for i from 1 by 1 to 3 do
        zdq[i,1]  :=  DQ[k-1,1]*T[i,3]:
        zddq[i,1] := DDQ[k-1,1]*T[i,3]:
   od:

   # Computo da Eq. (6.8.1)

   Wk[1,1]  := W[1,k-1]:
   Wk[2,1] := W[2,k-1]:
   Wk[3,1] := W[3,k-1]:

   W[1,k] := Wk[1,1] + TJ[k-1,1]*zdq[1,1]:
   W[2,k] := Wk[2,1] + TJ[k-1,1]*zdq[2,1]:
   W[3,k] := Wk[3,1] + TJ[k-1,1]*zdq[3,1]:
```

```
# Computo da Eq. (6.8.2)

WDQ := PVT(Wk,zdq):

DW[1,k] := DW[1,k-1] + TJ[k-1,1]*(zddq[1,1] + WDQ[1,1]):
DW[2,k] := DW[2,k-1] + TJ[k-1,1]*(zddq[2,1] + WDQ[2,1]):
DW[3,k] := DW[3,k-1] + TJ[k-1,1]*(zddq[3,1] + WDQ[3,1]):

# Computo da Eq. (6.8.3)

AI := evalm(&*(AI,MTH(DH[k-1,3],DH[k-1,2],DH[k-1,4],DH[k-1,1]))):
for i from 1 by 1 to 3 do
  for j from 1 by 1 to 3 do
     AI[i,j] := simplify(AI[i,j],
           [sin(-pi/2)=-1,cos(-pi/2)=0,sin(pi/2)=1
            cos(pi/2)=0,sin(pi)=0,cos(pi)=-1]):
     AI[i,j] := combine(AI[i,j]):
  od:
 od:

S[1,1] := combine(AI[1,4] - Pk[1,1]):
S[2,1] := combine(AI[2,4] - Pk[2,1]):
S[3,1] := combine(AI[3,4] - Pk[3,1]):

# Computo da Eq. (6.8.5)

Wk[1,1]  := W[1,k]:
Wk[2,1] := W[2,k]:
Wk[3,1] := W[3,k]:

DWk[1,1] := DW[1,k]:
```

```
        DWk[2,1] := DW[2,k]:
        DWk[3,1] := DW[3,k]:

        WDQ  := PVT(Wk,S):
        WDQ2 := PVT(Wk,WDQ):
        WDQ3 := PVT(Wk,zdq):
        DWDQ := PVT(DWk,S):

        DV[1,k] := DV[1,k-1] + DWDQ[1,1] + WDQ2[1,1] +
                   (1-TJ[k-1,1])*(zddq[1,1] + 2*WDQ3[1,1]):
        DV[2,k] := DV[2,k-1] + DWDQ[2,1] + WDQ2[2,1] +
                   (1-TJ[k-1,1])*(zddq[2,1] + 2*WDQ3[2,1]):
        DV[3,k] := DV[3,k-1] + DWDQ[3,1] + WDQ2[3,1] +
                   (1-TJ[k-1,1])*(zddq[3,1] + 2*WDQ3[3,1]);
    od:
end:

# ------------------------------------------------------------------------------------------
# Computar os Torques aplicados às juntas

PCTJ := proc(NElos::integer,TJ::matrix,Tab::matrix,DH::matrix,W::matrix,DW::matrix,
            DV::matrix,FO::matrix,TQ::matrix)
            local i,j,k,p,AI,T,CoM,Pk,TPk,TCoM,R,S,Wk,DWk,DWR,WR,first,second,
                zdq,zddq,Tk1,Fk,Fk1,z,WR2,ROT,JI,Dk,WkD,PVSF,PVRF,PVWkD,
                DWkD,TQT,FOT,TROT, tau:

        ROT  := matrix(3,3,0):
        TROT := matrix(3,3,0):
        CoM  := matrix(4,1,1):
        TCoM := matrix(4,1,1):
        DWk  := matrix(3,1,0):
```

```
Wk    := matrix(3,1,0):
AI    := matrix(4,4,0):
JI    := matrix(3,3,0):
z     := matrix(3,1,0):

for k from NElos by -1 to 1 do

        # Computo da Eq. (6.8.7)

        T := matrix(4,4,[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]):
        Tk1 := matrix(4,4,[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]):
        TQT  := matrix(1,3,0):
        FOT  := matrix(1,3,0):
        for p from 1 by 1 to k do
                AI := MTH(DH[p,3],DH[p,2],DH[p,4],DH[p,1]):  #Q[p,1]
                for i from 1 by 1 to 3 do
                  for j from 1 by 1 to 3 do
                     AI[i,j] := simplify(AI[i,j],
                                        [sin(-pi/2)=-1,cos(-pi/2)=0,sin(pi/2)=1,
                                         cos(pi/2)=0,sin(pi)=0,cos(pi)=-1]):
                     AI[i,j] := combine(AI[i,j]):
                   od:
                od:
                T  := evalm(&*(T,AI)):
                if (p=k-1) then
                     Tk1 := evalm(&*(Tk1,T)):
                fi:
        od:

        CoM[1,1] := Tab[k,9]:          # CoM do elo-k
        CoM[2,1] := Tab[k,10]:
```

```
CoM[3,1] := Tab[k,11]:

TCoM := evalm(&*(T,CoM)):

R[1,1] := combine(TCoM[1,1] - T[1,4]):
R[2,1] := combine(TCoM[2,1] - T[2,4]):
R[3,1] := combine(TCoM[3,1] - T[3,4]):

S[1,1] := combine(TCoM[1,1] - Tk1[1,4]):
S[2,1] := combine(TCoM[2,1] - Tk1[2,4]):
S[3,1] := combine(TCoM[3,1] - Tk1[3,4]):

# Computo da Eq. (6.8.8)

DWk[1,1] := DW[1,k+1]:
DWk[2,1] := DW[2,k+1]:
DWk[3,1] := DW[3,k+1]:

Wk[1,1]  := W[1,k+1]:
Wk[2,1] := W[2,k+1]:
Wk[3,1] := W[3,k+1]:

DWR := PVT(DWk,R):
WR   := PVT(Wk,R):
WR2  := PVT(Wk,WR):

FO[1,1] := FO[1,2] + Tab[k,2]*(DV[1,k+1] + DWR[1,1] + WR2[1,1]):
FO[2,1] := FO[2,2] + Tab[k,2]*(DV[2,k+1] + DWR[2,1] + WR2[2,1]):
FO[3,1] := FO[3,2] + Tab[k,2]*(DV[3,k+1] + DWR[3,1] + WR2[3,1]):

Fk[1,1] := FO[1,1]: Fk1[1,1] := FO[1,2]:
```

```
Fk[2,1] := FO[2,1]: Fk1[2,1] := FO[2,2]:
Fk[3,1] := FO[3,1]: Fk1[3,1] := FO[3,2]:

# Computo da Eq. (6.2.5)

ROT[1,1] := T[1,1]: ROT[1,2] := T[1,2]: ROT[1,3] := T[1,3]:
ROT[2,1] := T[2,1]: ROT[2,2] := T[2,2]: ROT[2,3] := T[2,3]:
ROT[3,1] := T[3,1]: ROT[3,2] := T[3,2]: ROT[3,3] := T[3,3]:

JI  := PMI(Tab,k):
TROT := transpose(ROT):
Dk := evalm(&*(ROT,JI,TROT)):

WkD   := evalm(&*(Dk,Wk)):
DWkD := evalm(&*(Dk,DWk)):

# Computo da Eq. (6.8.9)

PVSF := PVT(S,Fk):
PVRF := PVT(R,Fk1):
PVWkD := PVT(Wk,WkD):

TQ[1,1] := TQ[1,2] + PVSF[1,1] - PVRF[1,1] + DWkD[1,1] +
           PVWkD[1,1]:
TQ[2,1] := TQ[2,2] + PVSF[2,1] - PVRF[2,1] + DWkD[2,1] +
           PVWkD[2,1]:
TQ[3,1] := TQ[3,2] + PVSF[3,1] - PVRF[3,1] + DWkD[3,1] +
           PVWkD[3,1]:

# Computo da Eq. (6.8.10)
```

```
for i from 1 by 1 to 3 do
    z[i,1] := Tk1[i,3]:
od:

TQT[1,1] := TQ[1,1]: TQT[1,2] := TQ[2,1]: TQT[1,3] := TQ[3,1]:
FOT[1,1] := FO[1,1]:  FOT[1,2] := FO[2,1]:  FOT[1,3] := FO[3,1]:

first  := TJ[k,1]*evalm(&*(TQT,z)):
second := (1-TJ[k,1])*evalm(&*(FOT,z)):

tau := first[1,1] + second[1,1] + b[k](DQ[k,1]):

if (k=10) then
        save tau, `c:\\rb_modela\\resultados\\tau10.mat`;
fi:
if (k=9) then
        save tau, `c:\\rb_modela\\resultados\\tau9.mat`;
fi:
if (k=8) then
        save tau, `c:\\rb_modela\\resultados\\tau8.mat`;
fi:
if (k=7) then
        save tau, `c:\\rb_modela\\resultados\\tau7.mat`;
fi:
if (k=6) then
        save tau, `c:\\rb_modela\\resultados\\tau6.mat`;
fi:
if (k=5) then
        save tau, `c:\\rb_modela\\resultados\\tau5.mat`;
fi:
if (k=4) then
```

```
            save tau, `c:\\rb_modela\\resultados\\tau4.mat`;
        fi:
        if (k=3) then
            save tau, `c:\\rb_modela\\resultados\\tau3.mat`;
        fi:
        if (k=2) then
            save tau, `c:\\rb_modela\\resultados\\tau2.mat`;
        fi:
        if (k=1) then
            save tau, `c:\\rb_modela\\resultados\\tau1.mat`;
        fi:

        FO[1,2] := FO[1,1]:
        FO[2,2] := FO[2,1]:
        FO[3,2] := FO[3,1]:

        TQ[1,2] := TQ[1,1]:
        TQ[2,2] := TQ[2,1]:
        TQ[3,2] := TQ[3,1]:
    od:

end:

# ------------------------------------------------------------------------------------------
# Chamada do Procedimento para computar as equações diretas

PCED (NElos,TJ,Q,DQ,DDQ,DH):
# ------------------------------------------------------------------------------------------
# Chamada do Procedimento para computar torques

PCTJ(NElos,TJ,Tab,DH,W,DW,DV,FO,TQ):
```

ANEXO B

Validação do programa *NEROBOT*

Alguns problemas de modelagem de sistemas robóticos foram selecionados para exemplificar e validar o programa *NEROBOT*.

1º Problema: A Figura 2.4 ilustra um pêndulo invertido com um grau de liberdade (Schilling, 1990). Os sistemas de coordenadas foram atribuídos conforme o procedimento sistemático de *Denavit-Hartemberg*.

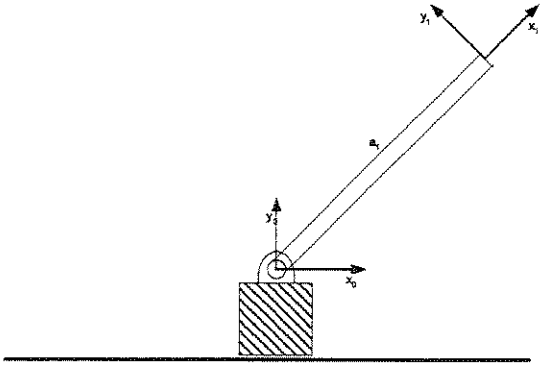


Figura 2.4 – Pêndulo Invertido..

A Tabela 2.4 apresenta os parâmetros cinemáticos, obtidos conforme o PS 2.2 e a Tabela 2.5 apresenta os parâmetros físicos do problema em questão.

Tabela 2.4 – Parâmetros cinemáticos relativos a Figura 2.2.

Elo	θ	α	a	D	Variáveis de Junta
1	q	0	a_1	0	$\theta(t)$

Tabela 2.5 – Parâmetros físicos relativos a Figura 2.4.

Elos	Massa	I_{xx}	I_{yy}	I_{zz}	I_{xy}	I_{xz}	I_{yz}	x_c	y_c	z_c
1	m_1	0	$m_1a_1^2/12$	$m_1a_1^2/12$	0	0	0	$-a_1/2$	0	0

As Tabelas 2.4 e 2.5 servem de base de dados para o programa *NEROBOT*. Adicionalmente, sejam as variáveis a seguir:

N_Elos := 1	número de elos;
GRAV := -matrix(3,1,[0,-g[o],0])	vetor gravitacional;
TJ := matrix(N_Elos,1,1)	vetor identificador de juntas revolutas (1);
Ftoos := matrix(3,1,0)	forças de contato do órgão terminal com o meio;
Ttoos := matrix(3,1,0)	momentos de contato do órgão terminal e o meio.

A partir destes dados de entrada, o modelo dinâmico foi obtido, sendo definido a seguir.

$$\frac{m_1 a_1^2}{3}\ddot{q} + \frac{m_1 a_1 c_1}{2} g_o + b_1(\dot{q}) = \tau$$

2° Problema: Considere-se um pêndulo invertido montado sobre uma base móvel, capaz de transladar na direção horizontal (Fig. 2.5) (Drummond, 1999). Os sistemas coordenados foram atribuídos conforme o procedimento sistemático de *Denavit-Hartemberg* (PS 2.1).

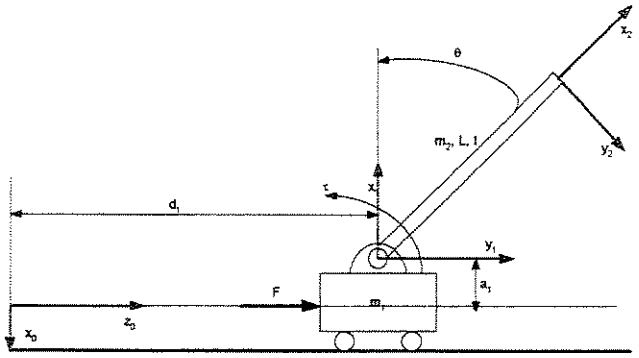


Figura 2.5 – Pêndulo invertido sobre uma base móvel.

Os parâmetros cinemáticos são definidos na Tabela 2.5, onde q define as variáveis generalizadas, e os parâmetros físicos do modelo são definidos na Tabela 2.6.

Tabela 2.5 – Parâmetros cinemáticos relativos à Figura 2.4.

Elo	θ	α	a	d	Variáveis de Junta
1	0	$\pi/2$	a_1	q_1	$d_1(t)$
2	q_2	0	a_2	0	$\theta_2(t)$

Tabela 2.6 – Parâmetros físicos relativos à Figura 2.4.

Elos	Massa	I_{xx}	I_{yy}	I_{zz}	I_{xy}	I_{xz}	I_{yz}	x_c	y_c	z_c
1	m_1	0	0	0	0	0	0	0	0	0
2	m_2	0	0	I	0	0	0	0	0	0

Adicionalmente sejam as seguintes variáveis:

NElos := 2 número de elos;

GRAV := matrix(3,1,[g[o],0,0]) vetor gravitacional ;

TJ := matrix(NElos,1,1) vetor identificador de juntas revolutas (1);

Ftoos := matrix(3,1,0) forças de contato do órgão terminal com o meio;

Ttoos := matrix(3,1,0) momentos de contato do órgão terminal e o meio.

A partir destes dados de entrada, o modelo dinâmico foi obtido, sendo definido a seguir:

$$\begin{bmatrix} m_1 + m_2 & m_2 Lc_\theta \\ m_2 Lc_\theta & I + m_2 L^2 \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} - \begin{bmatrix} m_2 Ls_\theta \dot{q}_2^2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ m_2 Lg_o s_\theta \end{bmatrix} + \begin{bmatrix} b_1(\dot{q}_1) \\ b_2(\dot{q}_2) \end{bmatrix} = \begin{bmatrix} F \\ \tau \end{bmatrix}$$

3º Problema: Seja um robô de dois graus de liberdade, ilustrado na Figura 2.6 (Schilling, 1990). Os sistemas de coordenadas foram atribuídos conforme o procedimento sistemático de *Denavit-Hartemberg* (PS. 2.1).

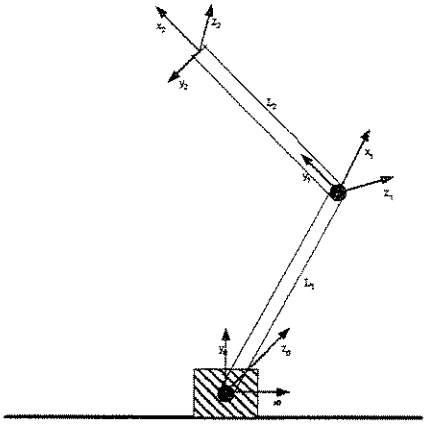


Figura 2.6 – Exemplo de um robô com dois graus de liberdade.

A Tabela 2.7 define os parâmetros cinemáticos e a Tabela 2.8, os parâmetros físicos.

Tabela 2.7 – Parâmetros cinemáticos relativos à Figura 2.2.

Elo	θ	α	a	d	Variáveis de Junta
1	q_1	0	L_1	0	$\theta_1(t)$
2	q_2	0	L_2	0	$\theta_2(t)$

Tabela 2.8 – Parâmetros físicos relativos à Figura 2.2.

Elos	Massa	I_{xx}	I_{yy}	I_{zz}	I_{xy}	I_{xz}	I_{yz}	x_c	y_c	z_c
1	m_1	0	$m_1a_1^2/12$	$m_1a_1^2/12$	0	0	0	$-a_1/2$	0	0
2	m_2	0	$m_2a_2^2/12$	$m_2a_2^2/12$	0	0	0	$-a_2/2$	0	0

As Tabelas 2.7 e 2.8 servem de base de dados para o programa *NEROBOT*. Adicionalmente sejam as seguintes variáveis:

N_Elos := 2	número de elos;
GRAV := matrix(3,1,[0,-g[o],0])	vetor gravitacional;
TJ := matrix(N_Elos,1,1)	vetor identificador de juntas revolutas (1);
Ftoos := matrix(3,1,0)	forças de contato do órgão terminal com o meio;
Ttoos := matrix(3,1,0)	momentos de contato do órgão terminal e o meio.

A partir destes dados de entrada, o modelo dinâmico foi obtido, sendo definido a seguir:

$$\begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix}\begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix}+\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}+\begin{bmatrix} g_1 \\ g_2 \end{bmatrix}+\begin{bmatrix} b_1(\dot{q}_1) \\ b_2(\dot{q}_2) \end{bmatrix}=\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$

Onde:

$$d_{11}=\left(\frac{m_1}{3}+m_2\right)a_1^2+\left(\frac{m_2a_2}{3}+m_2a_1c_2\right)a_2 \qquad d_{22}=\frac{m_2a_2^2}{3}$$

$$d_{12}=\left(\frac{a_1c_2}{2}+\frac{a_2}{3}\right)m_2a_2 \qquad d_{21}\equiv d_{12}$$

$$c_{11}=0 \qquad c_{12}=-m_2a_1a_2s_2\left(\dot{q}_1+\frac{\dot{q}_2}{2}\right)$$

$$c_{21} = \frac{m_2 a_1 a_2 s_2 \dot{q}_1}{2} \qquad c_{22} = 0$$

$$g_1 = \left[\left(\frac{m_1}{2} + m_2\right) a_1 c_1 + \frac{m_2 a_2 c_{12}}{2}\right] g_o \qquad g_2 = \frac{m_2 a_2 c_{12} g_o}{2}$$

4° Problema: Para finalizar a demonstração do programa *NEROBOT*, considere-se um robô SCARA com três graus de liberdade, conforme ilustra a Figura 2.7 (Schilling, 1990).

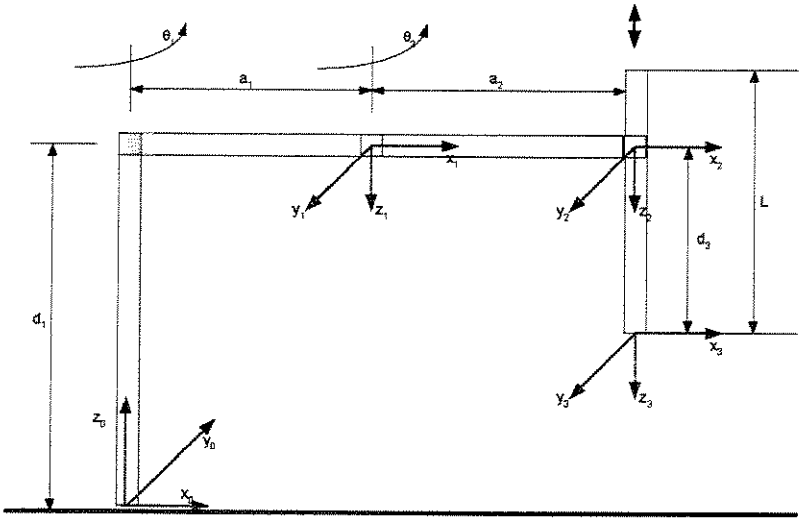


Figura 2.7 – Robô SCARA de três graus de liberdade

Os parâmetros geométricos estão definidos na Tabela 2.9 e os parâmetros físicos, na Tabela 2.10.

Tabela 2.9 – Parâmetros geométricos relativos à Figura 2.2.

Elo	θ	α	a	d	Variáveis de Junta
1	q_1	π	a_1	d_1	$\theta_1(t)$
2	q_2	0	a_2	0	$\theta_2(t)$
3	0	0	0	q_3	$d_3(t)$

Tabela 2.10 – Parâmetros físicos relativos à Figura 2.2.

Elo	Massa	I_{xx}	I_{yy}	I_{zz}	I_{xy}	I_{xz}	I_{yz}	x_c	y_c	z_c
1	m_1	0	$m_1a_1^2/12$	$m_1a_1^2/12$	0	0	0	$-a_1/2$	0	0
2	m_2	0	$m_2a_2^2/12$	$m_2a_2^2/12$	0	0	0	$-a_2/2$	0	0
3	m_3	$m_3L^2/12$	$m_3L^2/12$	0	0	0	0	0	0	-L/3

As Tabelas 2.9 e 2.10 são base de dados para o programa *NEROBOT*. Adicionalmente sejam as seguintes variáveis:

N_Elos := 2	número de elos;
GRAV := -matrix(3,1,[0,0,-g[o]])	vetor gravitacional;
TJ := matrix(N_Elos,1,1)	vetor identificador de juntas revolutas (1);
Ftoos := matrix(3,1,0)	forças de contato do órgão terminal com o meio;
Ttoos := matrix(3,1,0)	momentos de contato do órgão terminal e o meio.

A partir destes dados de entrada, o modelo dinâmico obtido foi:

$$\begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{12} & d_{22} & d_{23} \\ d_{13} & d_{23} & d_{33} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ m_3 g_0 \end{bmatrix} + \begin{bmatrix} b_1(\dot{q}_1) \\ b_2(\dot{q}_2) \\ b_3(\dot{q}_3) \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix}$$

Onde:

$$d_{11} = \left(\frac{m_1}{3} + m_2 + m_3\right) a_1^2 + (m_2 + 2m_3) a_1 a_2 c_2 + \left(\frac{m_2}{3} + m_3\right) a_2^2$$

$$d_{12} = -\left(\frac{m_2}{2} + m_3\right) a_1 a_2 c_2 - \left(\frac{m_2}{3} + m_3\right) a_2^2$$

$$d_{13} = 0 \qquad d_{23} = 0$$

$$d_{22} = \left(\frac{m_2}{3} + m_3\right) a_2^2$$

$$d_{33} = m_3$$

$$c_1 = -\left[(m_2 + 2m_3)\dot{q}_1\dot{q}_2 - \left(\frac{m_2}{2} + m_3\right)\dot{q}_2^2\right] a_1 a_2 s_2$$

$$c_2 = \left(\frac{m_2}{2} + m_3\right) a_1 a_2 s_2 \dot{q}_1^2$$

$$c_3 = 0$$

Os modelos dinâmicos obtidos, se comparados com os resultados apresentados nas respectivas referências bibliográficas, demonstram a correta implementação do formalismo *Newton-Euler*, permitindo validar o programa *NEROBOT*.

ANEXO C

Implementação Computacional do modelo do tronco

A Tabela C1 mostra a implementação computacional para o modelo dinâmico (2.50) que diz respeito ao tronco (pêndulo invertido) do robô bípede, gerado pelo programa *NEROBOT*. Foi utilizado o *software* Matlab® e o ambiente de programação *Simulink*®, que permite criar objetos denominados *S-functions*.

Tabela D1 – Implementação computacional do modelo do tronco

```
%=============================================================
% Tronco montado sobre a pelve - Contendo 2 graus de liberdade
% Implementa o modelo dinâmico não-linear do tronco (2.51).
%
%  Este M-file S-function implementa o seguinte modelo no espaço de estados:
%
%       x1  = q11       posição angular de j11
%       x2  = q12       posição angular de j12
%       x3  = q11'      veloc.  angular de j11
%       x4  = q12'      veloc.  angular de j12
%
%       x1' = x3
```

```
%       x2' = x4
%       x3' = f1(x)
%       x4' = f2(x)
%
%  Sendo:
%
%           F(x) = - M^-1(x)*(C(x) + G(x) + F(X) - u)
%
% Onde:
%           M(x)        matriz inercial (2x2)
%           C(x)        matriz de Corioulis e Centrifuga (2x1)
%           G(x)        vetor gravitacional (2x1)
%           F(x)        vetor de forças dissipativas (2x1)
%           u           ação de controle
%
%  Copyright (c) Jun, 2004 by João Bosco Gonçalves. All Rights Reserved.
%  $Revision: 1.0 $
% ================================================================================

function [sys,x0,str,ts] = tronco(t,x,u,flag)

% --------------------------------------------------------------------------------
% Dados Físicos do Rb:
%   L comprimento do tronco (m),
%   M massa (kg)
%   I momento de Inércia (kg m^2)
% --------------------------------------------------------------------------------

L = [0.05, 0.680];

M = [0.63, 5.10];
```

```
I = [0.26, 0.25];

% ------------------------------------------------------------------
% Define o método a ser empregado pelo valor da variável "flag"
% ------------------------------------------------------------------

switch flag,
  % ==============================================================
  % Inicialização das Variáveis do Modelo
  % ==============================================================
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % ==============================================================
  % Computo das Saídas Desejadas
  % ==============================================================
  case 3,
    sys=mdlOutputs(L,M,I,t,x,u);

 % ==============================================================
 % Métodos não utilizados (definidos pelos seguintes valores dos "flags"
 % ==============================================================
 case { 1, 2, 4, 9 },
   sys = [];

 % ==============================================================
 % Unexpected flags
 % ==============================================================
 otherwise
   error(['Unhandled flag = ',num2str(flag)]);
```

```
end

% ------------------------------------------------------------------
% Impementação dos Métodos
% ------------------------------------------------------------------

%=============================================================================
% mdlInitializeSizes
% Return the sizes, (x0) initial conditions, and (ts) sample times for the S-function.
%=============================================================================
function [sys,x0,str,ts]=mdlInitializeSizes,

  sizes = simsizes;
    sizes.NumContStates  = 0;         % 2 posições angulares (rad/s)
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 4;
    sizes.NumInputs      = 6;       % Vetor de torque nas juntas
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;

  sys = simsizes(sizes);
    x0  = [];
    str = [];
  ts  = [0.001 0]; % sample time: [period, offset]

%=============================================================================
% mdlOutputs
% Return the continuous states atualizados.
%=============================================================================
function sys=mdlOutputs(L,M,I,t,x,u),
```

```
X = u(3:6);                % variáveis de estados
U = u(1:2);                % vetor de torques

G = calc_G(L,M,X);         % vetor de força gravitacional
H = calc_H(L,M,I,X);       % matriz de inercia
D = calc_D(X);             % vetor dissipativo
C = calc_C(L,M,I,X);       % Forças de Coriulis e Centrifuga

F = -inv(H)*(C + G + D - U);

NULL = zeros(2,2);
EYE  = eye(2,2);

A  = [NULL,EYE;NULL,NULL];
B  = [NULL;F];

sys = A*X + B;

%=============================================================
% Computo do vetor G(j,1) = g*z(j,1)
%=============================================================
function [G] = calc_G(L,M,X),

G(1,1) = -9.81*(L(1)*sin(X(1))*(M(1)+M(2))+2*M(2)*cos(X(1))*cos(X(2))*L(2));
G(2,1) = -9.81*2*M(2)*sin(X(1))*sin(X(2))*L(2);

%=============================================================
% Computo da Matriz de Inércia, M
%=============================================================
function [M] = calc_M(L,M,I,X),
```

```
M(1,1) = M(2)*(0.5*(cos(2*X(2))*(3*L(2)^2 + I(1)) + 2*L(2)^2 + I(1)) +
4*L(1)*L(2)*cos(X(2))+L(2)^2);
M(1,2) = 0;
M(2,1) = 0;
M(2,2) = 4*M(2)*L(2)^2 + I(2);

% ==========================================================
% Computo do vetor de Forças Courioulis e Centrifugas, C
% ==========================================================
function [C] = calc_C(L,M,I,X),

C(1,1) = -M(2)*(sin(2*X(2))*(4*L(2)^2 + I(1)) + 4*L(1)*sin(X(2))*L(2))*X(3)*X(4);
C(2,1) =  M(2)*(sin(2*X(2))*((4*L(2)^2 + I(1))/2) + 2*sin(X(2))*L(1)*L(2))*X(4)^2;

% ==========================================================
% Computo do vetor de Forças Dissipativas, F
% ==========================================================
 function [F] = calc_F(X),

F = .3*eye(2,2)*X(1:2);
```

ANEXO D

Implementação Computacional do modelo do robô bípede

A Tabela D1 mostra a implementação computacional para o modelo dinâmico que diz respeito aos membros inferiores do robô bípede, gerado pelo programa *NEROBOT*. Foi utilizado o *software* Matlab® e o ambiente de programação *Simulink*®, que permite criar objetos denominados *S-functions*.

Tabela D1 – Implementação Computacional para o modelo do robô bípede.

```
function [sys,x0,str,ts] = RBIPEDE(t,x,u,flag)
% RBIPEDE implementa o modelo dinâmico não-linear do RB-03 (Gonçalves, 2003 - p.476).
% Contendo 12 graus de liberdade: 10 do MMI e 2 Tronco
%
%  Este M-file S-function implementa o seguinte modelo matricial:
%
%    x1' = x2
%    x2' = -PI(x)*(H(x)*x + G(x) + D(x))
%
%
% Onde:
%           P(x)    matriz inercial (10x10)          --> PI = inv(P)
%           H(x)    matriz de rigidez (10x10)
```

```
%           G(x)    vertor gravitacional (10x1)
%           D(x)    vetor de forças dissipativas (10x1)
%           x1 = q  vetor de posição angular (10x1)
%
%
%  Copyright (c) Dez, 2003 by João Bosco Gonçalves. All Rights Reserved.
%  $Revision: 1.0 $
%=============================================================

% -------------------------------------------------------------
% Define o método a ser empregado pelo valor da variável "flag"
% -------------------------------------------------------------

switch flag,

 %=============================================================
 % Inicialização das Variáveis do Modelo
 %=============================================================
 case 0,
  [sys,x0,str,ts] = mdlInitializeSizes;

%=============================================================
 % Computo das Saídas Desejadas
%=============================================================
 case 3,
   sys = mdlOutputs(t,u);

 %=============================================================
 % Métodos não utilizados (definidos pelos seguintes valores dos "flags"
 %=============================================================
 case {1,2,4,9},
  sys = [];

 %=============================================================
```

```
% Unexpected flags
%==========================================================================
otherwise
  error(['Unhandled flag = ',num2str(flag)]);

end
% ---------------------------------------------------------------------------
% Impementação dos Métodos, definidos pelo valor da variável "flag"
% ---------------------------------------------------------------------------

%=============================++============================================
% mdlInitializeSizes
% Return the sizes, (x0) initial conditions, and (ts) sample times for the S-function.
%=============================+============================================
function [sys,x0,str,ts]=mdlInitializeSizes,

    sizes = simsizes;
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 20;   % x(20x1) (10 posições + 10 veloc. angulares)
    sizes.NumInputs      = 61;        % 51 u = [Tr(10x1) Gama(10x1) Uref(20x1) PT(1) x(20x1)]
    sizes.DirFeedthrough = 0;
    sizes.NumSampleTimes = 1;

  sys = simsizes(sizes);
    x0  = []; %[0 0 0 0 pi pi 0 0 0 0 0 0 0 0 0 0 0 0 0 0]/2;  %Condição Inicial
    str = [];
  ts  = [1e-3 0];  % Sistema Contínuo --> sample time: [period, offset]

%=============================================================================
% mdlOutputs
% Return the continuous states atualizados.
%=============================================================================
function sys = mdlOutputs(t,u),
```

```
% ------------------------------------------------------------------------
% Carregamento Graviatcional
% ------------------------------------------------------------------------

    x = u(42:61);   % vetor de estados
    g = 9.81;

    G( 1,1) = g*termo_g1(x);
    G( 2,1) = g*termo_g2(x);
    G( 3,1) = g*termo_g3(x);
    G( 4,1) = g*termo_g4(x);
    G( 5,1) = g*termo_g5(x);
    G( 6,1) = g*termo_g6(x);
    G( 7,1) = g*termo_g7(x);
    G( 8,1) = g*termo_g8(x);
    G( 9,1) = g*termo_g9(x);
    G(10,1) = g*termo_g10(x);

% ------------------------------------------------------------------------
% Matriz de Inércias
% ------------------------------------------------------------------------

    M = zeros(10,10);

    M(1, 1) = termo_d11(x);
    M(1, 2) = termo_d12(x);     M(2, 2) = termo_d22(x);
    M(1, 3) = termo_d13(x);     M(2, 3) = termo_d23(x);     M(3, 3) = termo_d33(x);
    M(1, 4) = termo_d14(x);     M(2, 4) = termo_d24(x);     M(3, 4) = termo_d34(x);
    M(1, 5) = termo_d15(x);     M(2, 5) = termo_d25(x);     M(3, 5) = termo_d35(x);
    M(1, 6) = termo_d16(x);     M(2, 6) = termo_d26(x);     M(3, 6) = termo_d36(x);
    M(1, 7) = termo_d17(x);     M(2, 7) = termo_d27(x);     M(3, 7) = termo_d37(x);
    M(1, 8) = termo_d18(x);     M(2, 8) = termo_d28(x);     M(3, 8) = termo_d38(x);
    M(1, 9) = termo_d19(x);     M(2, 9) = termo_d29(x);     M(3, 9) = termo_d39(x);
```

```
M(1,10) = termo_d110(x);    M(2,10) = termo_d210(x);    M(3,10) = termo_d310(x);

M(4, 1) = M(1,4);           M(5, 1) = M(1,5);           M(6, 1) = M(1,6);
M(4, 2) = M(2,4);           M(5, 2) = M(2,5);           M(6, 2) = M(2,6);
M(4, 3) = M(3,4);           M(5, 3) = M(3,5);           M(6, 3) = M(3,6);
M(4, 4) = termo_d44(x);
M(4, 5) = termo_d45(x);     M(5, 5) = termo_d55(x);
M(4, 6) = termo_d46(x);     M(5, 6) = termo_d56(x);     M(6, 6) = termo_d66(x);
M(4, 7) = termo_d47(x);     M(5, 7) = termo_d57(x);     M(6, 7) = termo_d67(x);
M(4, 8) = termo_d48(x);     M(5, 8) = termo_d58(x);     M(6, 8) = termo_d68(x);
M(4, 9) = termo_d49(x);     M(5, 9) = termo_d59(x);     M(6, 9) = termo_d69(x);
M(4,10) = termo_d410(x);    M(5,10) = termo_d510(x);    M(6,10) = termo_d610(x);

M(7, 1) = M(1,7);           M(8, 1) = M(1,8);           M(9, 1) = M(1,9);
M(7, 2) = M(2,7);           M(8, 2) = M(2,8);           M(9, 2) = M(2,9);
M(7, 3) = M(3,7);           M(8, 3) = M(3,8);           M(9, 3) = M(3,9);
M(7, 4) = M(4,7);           M(8, 4) = M(4,8);           M(9, 4) = M(4,9);
M(7, 5) = M(5,7);           M(8, 5) = M(5,8);           M(9, 5) = M(5,9);
M(7, 6) = M(6,7);           M(8, 6) = M(6,8);           M(9, 6) = M(6,9);
M(7, 7) = termo_d77(x);
M(7, 8) = termo_d78(x);     M(8, 8) = termo_d88(x);
M(7, 9) = termo_d79(x);     M(8, 9) = termo_d89(x);     M(9, 9) = termo_d99(x);
M(7,10) = termo_d710(x);    M(8,10) = termo_d810(x);    M(9,10) = termo_d910(x);

M(10, 1) = M(1,10);         M(2, 1) = M(1,2);
M(10, 2) = M(2,10);         M(3, 1) = M(1,3);
M(10, 3) = M(3,10);         M(3, 2) = M(2,3);
M(10, 4) = M(4,10);         M(5, 4) = M(4,5);
M(10, 5) = M(5,10);         M(6, 4) = M(4,6);
M(10, 6) = M(6,10);         M(6, 5) = M(5,6);
M(10, 7) = M(7,10);         M(8, 7) = M(7,8);
M(10, 8) = M(8,10);         M(9, 7) = M(7,9);
M(10, 9) = M(9,10);         M(9, 8) = M(8,9);
M(10,10) = termo_d1010(x);
```

```
    MI = pinv(M);

% ------------------------------------------------------------------------
% Forças centrifugas e corioulis
% ------------------------------------------------------------------------
    C(1,1) = termo_c1(x);
    C(2,1) = termo_c2(x);
    C(3,1) = termo_c3(x);
    C(4,1) = termo_c4(x);
    C(5,1) = termo_c5(x);
    C(6,1) = termo_c6(x);
    C(7,1) = termo_c7(x);
    C(8,1) = termo_c8(x);
    C(9,1) = termo_c9(x);
    C(10,1) = termo_c10(x);

% ------------------------------------------------------------------------
% Forças Dissipativas
% ------------------------------------------------------------------------

  F1 = [ 0.2,-0.2,  0,   0,  0,  0,  0,  0,  0,  0;...
      -0.2, 0.4,-0.2,   0,  0,  0,  0,  0,  0,  0;...
        0,-0.2, 0.4, -0.2,  0,  0,  0,  0,  0,  0;...
        0,  0,-0.2,  0.4,-0.2,  0,  0,  0,  0,  0;...
        0,  0,  0, -0.2, 0.4,-0.2,  0,  0,  0,  0;...
        0,  0,  0,   0,-0.2, 0.4,-0.2,  0,  0,  0;...
        0,  0,  0,   0,  0,-0.2, 0.4,-0.2,  0,  0;...
        0,  0,  0,   0,  0,  0,-0.2, 0.4,-0.2,  0;...
        0,  0,  0,   0,  0,  0,  0,-0.2, 0.4,-0.2;...
        0,  0,  0,   0,  0,  0,  0,  0,-0.2, 0.2]*x(11:20);

  bd = 0.62;
  bs = 0.91;
  for k=1:10,
```

```
   F2(k,1) = sign(x(10+k))*(bd + (bs - bd)*exp(-abs(x(10+k))/0.2));
 end
 F  = F1 + F2;

% ------------------------------------------------------------------------
% Modelo Dinâmico
% ------------------------------------------------------------------------

 % Termos da Eq. (4.10)
 Gamma   = u(11:20);      % parcela estimada via RBF
 taur    = u( 1:10);      % termo de robustez, Eq. (4.12)
 Pert    = zeros(10,1);   % Eq. (2.86)
 Pert(5,1) = (41,1);
 ref     = u(21:40);      % Sinal de referência, posição e veloc. angular

   % Modelo Nomimal
   Z = zeros(10,10);
   I = eye(10,10);
   A = [Z,I;Z,Z];
   B = [Z;I];

 % Modelo de Referência
 wn  = 100;
 qsi = 0.3;
 K   = 15;

 A1  = -wn^2*eye(10,10);
 A2  = -2*qsi*wn*eye(10,10);
 Bmr = [Z,-K*eye(10,10);-A1,Z];

 % Sinal de controle, Eq. (4.10)
 tau = M*(pinv(B)*Bmr*ref + [A1 A2]*x) + (C + G + F + Gamma) -  taur;
```

```
% Computo do Modelo Nominal, Eq. (4.2)

sys = A*x + B*MI*(tau + Pert - C - G - F);
```

ANEXO E

Solução para o modelo do tronco

A Tabela E1 mostra o arquivo gerado para resolver o problema de otimização proposto em (3.38), cuja solução fornece as variáveis de posição, velocidade e aceleração para o modelo do tronco. Foi utilizado o *software* Matlab® V5.3 R11.1 para geração do arquivo (extensão **.m**). A Tabela E2 apresenta a função *fotimo(x)* que implementa o funcional (3.38).

Tabela E1 – Solução para o modelo do tronco.

```
clear all, close all, clc, pause(.1)

load 'c:\matlabr11\work\traj_tronco\otimizador\ZMP.mat' -ascii

XO = zeros(size(ZMP,1)+3,2);
lb  = -pi/3*ones(1,2);
ub  = pi/3*ones(1,2);

options = optimset('Display','iter','MaxFunEvals','15000');
x       = fmincon('fotimo',XO,[],[],[],[],lb,ub,[],options);
```

Tabela E2 – Função de otimização.

```
function [F2] = fotimo(x),

% Carrega a marcha planejada
load 'c:\matlabr11\work\traj_tronco\otimizador\POS.mat'   -ascii
load 'c:\matlabr11\work\traj_tronco\otimizador\ACE.mat'   -ascii
load 'c:\matlabr11\work\traj_tronco\otimizador\POSQ.mat'  -ascii
load 'c:\matlabr11\work\traj_tronco\otimizador\ACEQ.mat'  -ascii
load 'c:\matlabr11\work\traj_tronco\otimizador\GRAV.mat'  -ascii
load 'c:\matlabr11\work\traj_tronco\otimizador\MASSA.mat' -ascii
load 'c:\matlabr11\work\traj_tronco\otimizador\ZMP.mat'   -ascii
load 'c:\MATLABR11\work\traj_tronco\dados_treinamento\tempo.mat' -ascii

[N,p] = size(ZMP);

L1 = 0.05;
L  = 1.00;
S  = [];

% Separando os dados otmizados
THETA = x(1:N,:);
DDQO  = x(N+1:N+3,1);
DDQF  = x(N+1:N+3,2);

% Localização do Tronco segundo SC Global
S(:,1) = -L.*sin(THETA(:,2));
S(:,2) =  sin(THETA(:,1)).*(L1+ (L-L1).*cos(THETA(:,2)));
S(:,3) =  cos(THETA(:,1)).*(L1+ (L-L1).*cos(THETA(:,2)));

h      = tempo(2)-tempo(1);
PX     = zeros(N+2,1);
```

```
PY    = PX;
PZ    = PX;

% Condições iniciais e de contornos
PX(1,1)       = 2*S(1,1) - S(2,1) + (h^2)*DDQO(1);
PX(2:N+1,1) = S(:,1);
PX(N+2,1)    = (h^2)*DDQF(1) + 2*S(N,1) - S(N-1,1);

PY(1,1)       = 2*S(1,2) - S(2,2) + (h^2)*DDQO(2);;
PY(2:N+1,1) = S(:,2);
PY(N+2,1)    = (h^2)*DDQF(2) + 2*S(N,2) - S(N-1,2);

PZ(1,1)       = 2*S(1,3) - S(2,3) + (h^2)*DDQO(3);;
PZ(2:N+1,1) = S(:,3);
PZ(N+2,1)    = (h^2)*DDQF(3) + 2*S(N,3) - S(N-1,3);

% Rotina para diferencia central
dx  = dycentral(PX,h);
dy  = dycentral(PY,h);
dz  = dycentral(PZ,h);

ddx = ddycentral(PX,h);
ddy = ddycentral(PY,h);
ddz = ddycentral(PZ,h);

NE  = size(MASSA,2)-1;
NA  = size(ZMP,1);

% Implementa as equações (3.35) e (3.36)
zmp = calc_zmp(NE,NA,POS,ACE,POSQ,ACEQ,[ddx,ddy,ddz],...
          S(:,1:2),S(:,3),GRAV,MASSA);
```

```
% Funcional
F2 = norm(ZMP-zmp);

if (F2<0.01),
  subplot(231),plot(tempo,S(:,1),'b',tempo,S(:,2),'r',tempo,S(:,3),'k'),legend('X','Y','Z')
  subplot(232),plot(tempo,dx,'b',tempo,dy,'r',tempo,dz,'k')
  subplot(233),plot(tempo,ddx,'b',tempo,ddy,'r',tempo,ddz,'k')

  subplot(223),plot(tempo,x(1:N,1)*180/pi,'b',tempo,x(1:N,2)*180/pi,'r'),legend('Q11','Q12')
  subplot(224),plot(zmp(:,1),zmp(:,2),'k.-',ZMP(:,1),ZMP(:,2),'b')
  pause(.1)
  save c:\matlabr11\work\traj_tronco\valer\DX.mat   dx  dy  dz -ascii -double
  save c:\matlabr11\work\traj_tronco\valer\DDX.mat ddx ddy ddz -ascii -double
end
```

ANEXO F

Generalização para a solução do modelo do tronco

A Tabela F1 mostra o arquivo gerado para o treinamento da rede neural recorrente que utiliza os dados fornecidos pelo *GAM* (Anexo H) e pela solução do tronco (Anexo E). Foi utilizado o *software* Matlab® V5.3 R11.1 para geração do arquivo (extensão **.m**). A Tabela F2 apresenta a função *fotimo(x)* que implementa o funcional (3.38).

Tabela F1 – Treinamento da rede neural recorrente.

```
% ==========================================================
% Realiza o Treinamento de uma Rede Neural para a Solução do Tronco
% ==========================================================
% ==========================================
% Created by: João Bosco Gonçalves
% Date        : 03/08/2002
% Version     : 01
% ==========================================


% ------------------------------------------------------------
% Configuração do Sistema
% ------------------------------------------------------------
```

```
config

% ------------------------------------------------------------------------
% Dados do Robô
% ------------------------------------------------------------------------
robo_bipede2k3

% ------------------------------------------------------------------------
% Prepara Dados para o Treinamento da Rerde Neural
% ------------------------------------------------------------------------
LAMBDA = 0.15;  H   = ao;  teta2xyz(LAMBDA,H,50)

% ------------------------------------------------------------------------
% Carrega os arquivos de entrada, gerado pela função teta2xyz,
% para computar a excitação da rede neural
% ------------------------------------------------------------------------
% Posições Cartesianas das Pernas e do Centro do Corpo
load 'c:\MATLABR11\work\traj_tronco\dados_treinamento\mtp1.mat' -ascii
load 'c:\MATLABR11\work\traj_tronco\dados_treinamento\mtp2.mat' -ascii
load 'c:\MATLABR11\work\traj_tronco\dados_treinamento\mtpc.mat' -ascii
% Acelerações Cartesianas das Pernas e do Centro do Corpo
load 'c:\MATLABR11\work\traj_tronco\dados_treinamento\mta1.mat' -ascii
load 'c:\MATLABR11\work\traj_tronco\dados_treinamento\mta2.mat' -ascii
load 'c:\MATLABR11\work\traj_tronco\dados_treinamento\mtac.mat' -ascii
% Base de tempo
load 'c:\MATLABR11\work\traj_tronco\dados_treinamento\tempo.mat' -ascii

mtp  = [mtp1;mtp2]; mta  = [mta1;mta2]; mtpq = mtpc; mtaq = mtac;

% ------------------------------------------------------------------------
% Dados da Simulação
```

```
% ------------------------------------------------------------------------
dados_sim

% ------------------------------------------------------------------------
% Gera a saída padrão (ytrein)
% ------------------------------------------------------------------------
op    = 1;  % Posição ereta, P1 na origem do sistema de coordenadas fixo
zmpdes = saida_padrao(n_amos,op,LAMBDA,a5);

% ------------------------------------------------------------------------
% Gera a excitação (xtrein)
% ------------------------------------------------------------------------
xtrein = excitacao(zmpdes,mass,grav,mtp',mta',mtpq',mtaq',n_elem,n_amos,altura_tronco);

% ------------------------------------------------------------------------
% Normalização do vetor de treinamento (padrão de entrada)
% ------------------------------------------------------------------------
xt1 = (xtrein(:,1)-mean(xtrein(:,1)))/(std(xtrein(:,1)))^2;
xt2 = (xtrein(:,2)-mean(xtrein(:,2)))/(std(xtrein(:,2)))^2;

vtrein = [xt1,xt2,zeros(n_amos,2)];
ytrein = [xt1,xt2];
XO    = zeros(6,1);   % Acelerações Iniciais e Finais
save 'c:\matlabr11\work\traj_tronco\otimizador\XO.mat'  XO -ascii -double

% ------------------------------------------------------------------------
% Treinamento da rede neural via processo interativo I - padrão-a-padrão
% ------------------------------------------------------------------------
epocas        =  20;
M1            =  15;  M2  =  30;  saida_rede  = 2;
eqm           =  0.0020;
```

```
alfa          = 0.0100;
dn            = 0.0010;
val           = 0.25;

pause(1)
yc = mbrn4camadas(epocas,M1,M2,saida_rede,vtrein,ytrein,alfa,eqm,dn,val,...
          n_elem,mtp',mta',mtpq',mtaq',grav,altura_tronco,mass,zmpdes,tempo');
```

Tabela F2 – Implementação computacional da rede neural recorrente.

```
% Treinamento de uma rede neural
% Implementa o Processo Interativo II - Método Batelada
% Recebe como argumentos:
%  m   : número de neurônios da camada intermediária
%  p   :     número de neurônios da camada de saída
%  xp  : matriz contendo os padrões de entrada
%  yp  : matriz contendo os padrões de saída (resposta desejada)
%  alfa: taxa de aprendizado > 0
%  eqm :  valor mínimo para o erro quadrático médio > 0
% Essa função retorna as matrizes de pesos ótimos: wi e ws;

function [yc] = mbrn4camadas(epocas,M1,M2,N_saida,xtrein,yp,alfa,eqm,dn,val,...
          N_ELEM,POS,ACE,POSQ,ACEQ,GRAV,Alt_tronco,MASSA,ZMP,TEMPO),

%---- Número de amostras de entrada
[N_amos,N_padrao] = size(xtrein);
%---- Inicializa o número de interações necessárias ao treinamento da rede neural
k  = 1;
%---- Introduz bias na matriz [xtrein]
xp = xpadrao(xtrein,N_amos,N_padrao);
%----- Inicializa as matrizes de pesos das camadas intermediária e de saída
```

```
wi1 = val.*pcinter(M1,N_padrao);  % pesos 1a camada interm., inclusive bias
wi2 = val.*pcinter(M2,M1);        % pesos 2a camada interm., inclusive bias
ws  = val.*pcsaida(N_saida,M2);   % pesos da camada de saída, inclusive bias
% ---- Inicializa as matrizes de busca
Nt   = (N_padrao+1)*M1 + (M1+1)*M2 + (M2+1)*N_saida;
H    = eye(Nt);
vgrad = zeros(Nt,1);
p    = vgrad;
% ---- Computa a norma dos pesos
nwi1(k) = norm(wi1);
nwi2(k) = norm(wi2);
nws(k)  = norm(ws);
eqmc(1) = 1;
%------ Implementa o Processo Interativo I: Método Padrão a Padrão
fprintf('    Treinamento da Rede Neural Recorrente   \n\n')
fprintf('Iteração No. %d, Erro Quadrádico Médio %3.5f, alfa = %3.5f \n',k,eqmc(k),alfa)
ak(k)  = alfa;
while ((eqmc(k)>= eqm) ~= (k>epocas))

  %----- Calcula as entradas para as funções de ativação da camada intermediária
  ui1 = uci(xp,wi1);
  % ---- Transforma ui para zi (entrada na 2a. camada interm.)
  zi1 = zci(ui1);
  %----- Calcula as entradas para as funções de ativação da camada intermediária
  ui2 = uci(zi1,wi2);
  % ---- Transforma ui para zi (entrada na 2a. camada interm.)
  zi2 = zci(ui2);
  %----- Calcula as entradas para as funções de ativação da camada de saída
  sj = saida_j(zi2,ws);
  % ---- Transforma sj para yi (saída da rede neural), na iteração k
  yc = rn(sj);
```

```
%----- Obtenção dos dados cartesianos para o tronco ----
TRONCO = cartesiano(Alt_tronco,yc,yp,POS,ACE,POSQ,ACEQ,ZMP,....
                                GRAV,MASSA,1);
% ---- Recorrência
xtrein(:,3:4) = yc;
%----- Calculo do valor ZMP, a partir da saída da rede neural
zmp = calc_zmp(N_ELEM,N_amos,POS,ACE,POSQ,ACEQ,TRONCO(:,4:6),...
                        TRONCO(:,1:2),TRONCO(:,3),GRAV,MASSA);


% --- Funcional
J = funcional(N_amos,size(yp,2),ZMP,zmp);
%----- Cálculo dos vetores de Sensibilidade
VSS  = senscs(TRONCO,ACEQ,GRAV,yc,ZMP,zmp,sj,N_amos);
VSI2 = sensci(VSS,ui2,ws,M2,N_saida);
VSI1 = sensci(VSI2,ui1,wi2,M1,M2);
% ---- Calculo do gradiente
DJI1 = gradci(VSI1,xp);
DJI2 = gradci(VSI2,zi1);
DJS  = gradcs(VSS ,zi2);
% ---- Computo da direção de busca
vgrada = vgrad;   % Armazena o gradiente anterior
vgrad1 = reshape(DJI1,(N_padrao+1)*M1,1);
vgrad2 = reshape(DJI2,(M1+1)*M2,1);
vgrads = reshape(DJS,(M2+1)*N_saida,1);
vgrad  = [vgrad1;vgrad2;vgrads];
d      = H*vgrad;
d      = d/norm(d);
if (rem(k+1,Nt) == 0),
  d = vgrad;
  H = eye(Nt);
end
```

```
DJI1 = reshape(d(1:(N_padrao+1)*M1),N_padrao+1,M1);
DJI2 = reshape(d((N_padrao+1)*M1+1:(N_padrao+1)*M1+(M1+1)*M2),M1+1,M2);
DJS  = reshape(d((N_padrao+1)*M1+(M1+1)*M2+1:Nt),M2+1,N_saida);
alfa = goldsec2(wi1,wi2,ws,DJI1,DJI2,DJS,xp,yc,Alt_tronco,dn);
pa   = p;
p    = alfa*d;
q    = vgrad - vgrada;
if (norm(q)>0),
  q = q/norm(q);
else
  q = vgrad/norm(vgrad);
end
if (((p')*q)<=0),
  p = pa;
end
H = H + (p*p')/((p')*q) - (H*q*(q')*H)/((q')*H*q);
% Atualiza os pesos
wi1 = wi1 + alfa*DJI1;
wi2 = wi2 + alfa*DJI2;
ws  = ws  + alfa*DJS;
% ----- armazena evolução das normas
nwi1(k) = norm(wi1);
nwi2(k) = norm(wi2);
nws(k)  = norm(ws);
nj1(k)  = norm(reshape(DJI1,M1*(N_padrao+1),1));
nj2(k)  = norm(reshape(DJI2,M2*(M1+1),1));
nj3(k)  = norm(reshape(DJS,N_saida*(M2+1),1));
% ---- Avança na contagem
k = k+1;
%------ Calcula o erro quadrático médio na interação k
eqmc(k) = J;
```

```
  ak(k) = alfa;
  fprintf('Iteração No. %d, Erro Quadrádico Médio %3.5f, alfa = %3.8f\n',k,eqmc(k),alfa)
end

save 'c:\matlabr11\work\traj_tronco\pesos\wi1.mat' wi1 -ascii -double
save 'c:\matlabr11\work\traj_tronco\pesos\wi2.mat' wi2 -ascii -double
save 'c:\matlabr11\work\traj_tronco\pesos\ ws.mat' ws  -ascii -double

%----- Calcula as entradas para as funções de ativação da camada intermediária
ui1 = uci(xp,wi1);
% ---- Transforma ui para zi (entrada na 2a. camada interm.)
zi1 = zci(ui1);
%----- Calcula as entradas para as funções de ativação da camada intermediária
ui2 = uci(zi1,wi2);
% ---- Transforma ui para zi (entrada na 2a. camada interm.)
zi2 = zci(ui2);
%----- Calcula as entradas para as funções de ativação da camada de saída
sj = saida_j(zi2,ws);
% ---- Transforma sj para yi (saída da rede neural), na iteração k
yc = rn(sj);
% ---- Obtenção dos dados cartesianos para o tronco ---- %
TRONCO = cartesiano(Alt_tronco,yc,yp,POS,ACE,POSQ,ACEQ,ZMP,GRAV,MASSA,1);
% ---- Calculo da excitação estimada
K = excestimada(POSQ,ACEQ,TRONCO,GRAV,ZMP);
% ---- Calculo do ZMP estimado
zmp = calc_zmp(N_ELEM,N_amos,POS,ACE,POSQ,ACEQ,TRONCO(:,4:6),...
          TRONCO(:,1:2),TRONCO(:,3),GRAV,MASSA);



% Figuras
```

```
figure
nt = 0:length(nwi1)-1;
subplot(221),semilogy(nt,nwi1,'r.-',nt,nwi2,'k.-',nt,nws,'b.-'),
title('||w|| e ||v||'),grid

nt = 0:length(nj1)-1;
subplot(222),semilogy(nt,nj1,'r.-',nt,nj2,'b.-',nt,nj3,'k.-'),
title('||GRAD||'),grid

nt = 0:length(eqmc)-1;
subplot(223),semilogy(nt,eqmc,'b.-'),title('EQM')
grid

nt = 0:length(ak)-1;
subplot(224),semilogy(nt,ak,'r.-'),title('alfa')
grid

figure
nt = (0:length(yp)-1)';
subplot(211),plot(nt,K,'r.-',nt,yp,'b-')
ylabel('Yest(red) vs Ydes(blue)'),grid

subplot(212),plot(zmp(:,1),zmp(:,2),'k.-',ZMP(:,1),ZMP(:,2),'b-')
ylabel('ZMP'),grid
```

ANEXO G

Implementação computacional do *GAT*

A Tabela G1 apresenta a função (*gat*) utilizada para acionar o *gera_traj*, apresentado na Tabela G2 que implementa o *G*erador *A*utomático de *T*rajetórias (GAT), que determina os valores de velocidades e acelerações nos pontos-via. As informações necessárias são: os pontos-via, os instantes associados, as condições iniciais e de contorno do problema. Foi utilizado o *software* Matlab® V5.3 R11.1 para geração das funções, com extensões **.m**.

Tabela G1 – Função para acionar o *GAT*.

```
function [X] = gat(posturas,t,range,op),

% ------------------------------------------------------------------------
% Aplica o Algoritmo Genético ao problema de otmização para determinar
% velocidades e acelerações nos pontos-vias.
% ------------------------------------------------------------------------
% Sintaxe: [F1] = gat(posturas,t,range,op),
%
%  posturas = [QA,QB ....,QF]      --> Pontos para interpolar
%  range    = [dqmin,dqmax,0 ou 1]  --> Faixa de Valores, R
%  t        = [TA,TB,....,TF]       --> tempos relacionados aos pontos-via
```

```
%
% Variável de Retorno: VBEST(popsize)
% ==========================================================
% Created by: João Bosco Gonçalves
% Date     : 23/10/2002
% Version  : 01
% ==========================================================

[nj,np] = size(posturas);

b     = 5;
gmax  = 3;
pm    = 0.013;
pc    = 0.50;
popsize = 20;

dqo  = [0,0];
ddqo = [0,0];

DVBEST(1:nj,1)  = dqo(1);
DVBEST(1:nj,np) = dqo(2);
DDVBEST(1:nj,1) = ddqo(1);
DDVBEST(1:nj,np) = ddqo(2);

opcao = 3; % retorna posição, velocidade, aceleração, jerk e tempo
figure
for j=1:nj,
 subplot(2,5,j)
 F = gera_traj(b,gmax,pm,pc,popsize,[range(j,:);range(nj+j,:)],...
                posturas(j,:),dqo,ddqo,t,op);
 title(num2str(j))
```

```
DVBEST(j,2:np-1) = F(1,1:np-2);          % Velocidades
DDVBEST(j,2:np-1) = F(1,np-1:2*(np-2));  % Acelerações nos pontos-via
F                = polig5r(t,np-1,posturas(j,:),DVBEST(j,:),DDVBEST(j,:),opcao);
QT (:,j)         = F(:,1);
DQT(:,j)         = F(:,2);
DDQT(:,j)        = F(:,3);
DDDQT(:,j)       = F(:,4);
end
subplot(256),ylabel('Evolução do Algoritmo Genético'),xlabel('No. Gerações')

T(:,1) = F(:,5);
save 'c:\matlabr11\work\gam\rb\QT'     QT  -ascii -double;
save 'c:\matlabr11\work\gam\rb\DQT'    DQT  -ascii -double;
save 'c:\matlabr11\work\gam\rb\DDQT'   DDQT -ascii -double;
save 'c:\matlabr11\work\gam\rb\DDDQT' DDDQT -ascii -double;
save 'c:\matlabr11\work\gam\rb\T'      T    -ascii -double;
```

Tabela G2 – Implementação computacional do *GAT*.

```
function [VBEST] = gera_traj(b,gmax,pm,pc,popsize,range,q,dqo,ddqo,to,op),
% ------------------------------------------------------------------------
% Aplica o Algoritmo Genético ao problema de otmização de velocidades e acelerações
% nos pontos-vias.
% ------------------------------------------------------------------------
% Sintaxe: [VBEST] = gera_traj(b,gmax,pm,pc,popsize,range,q,dqo,ddqo,to)
% Onde:
%          b                --> Parâmetro de Projeto
%                   gmax                              --> No. máximo de gerações
%          pm                --> Taxa de mutação
%          pc                --> Taxa de recombinação
%                   popsize                           --> tamanho da população
```

```
%                   range = [dqmin,dqmax,0 ou 1]  --> Faixa de Valores, R
%          q    = [PI,x1,x2,...PM,...PF] --> Pontos-via
%                   dqo  = [dxo,dxf]          --> Condições iniciais e finais
%                   ddqo  = [ddxo,ddxf]         --> Condições iniciais e finais
%                   to                        --> tempos relacionados aos pontos-via
%
% Variável de Retorno: VBEST(popsize)
% ================================================================================
% Created by: João Bosco Gonçalves
% Date     : 23/10/2002
% Version  : 01
% ================================================================================
hold on
pause(.1)

% --------------------------------------------------------------------------------
% Inicializa variáveis
% --------------------------------------------------------------------------------
geracao = 0;
[L,npv] = size(q);

% --------------------------------------------------------------------------------
% População inicial de indivíduos
% --------------------------------------------------------------------------------
pop = popinit_real(popsize,op*(npv-2),range);

% --------------------------------------------------------------------------------
% Avaliação do melhor indivíduo
% --------------------------------------------------------------------------------
x  = evalpop_real(npv,to,pop,q,dqo,ddqo,op); % x={F,Q}
[fmax,i] = max(x(:,1));
```

```
stem(geracao,fmax,'bo-')
stem(geracao,min(x(:,1)),'rx')
pause(.2)

% ------------------------------------------------------------------------
% Evolução ao longo de gerações
% ------------------------------------------------------------------------
while(geracao < gmax),

  % ------------------------------------------------------------------------
  % Próxima Geração
  % ------------------------------------------------------------------------
  geracao = geracao + 1;

  % ------------------------------------------------------------------------
  % Seleção elitista
  % ------------------------------------------------------------------------
  p = pop(i,:);

  % ------------------------------------------------------------------------
  % Seleção de Indivíduos
  % ------------------------------------------------------------------------
  newpop = popselect_real(pop,x(:,2),range);

  % ------------------------------------------------------------------------
  % Operador genético de recombinação
  % ------------------------------------------------------------------------
  popcross = crossover_real(newpop,pc,range);

  % ------------------------------------------------------------------------
  % Operador genético de mutação
```

```
% ------------------------------------------------------------------------
pop = mutation_real(popcross,gmax,geracao,pm,b,popsize,range);

% ------------------------------------------------------------------------
% Inclusão do Melhor Indivíduo à nova geração
% ------------------------------------------------------------------------
x  = evalpop_real(npv,to,pop,q,dqo,ddqo,op);
[fmin,i] = min(x(:,1));
pop(i,:) = p;
x(i,1)   = fmax;

% ------------------------------------------------------------------------
% Avaliação do melhor indivíduo
% ------------------------------------------------------------------------
[fmax,i] = max(x(:,1));
stem(geracao,fmax,'bo-')
stem(geracao,fmin,'rx')
pause(.2)
 end
grid, hold off
[fmax,i] = max(x(:,1));
VBEST   = pop(i,:);        % Retorna Velocidades e Acelerações nos Pontos-via
```

A Tabela G3 apresenta a função *crossover_real* que realiza a recombinação de indivíduos, gerando uma nova população.

Tabela G3 – Função para recombinação de indivíduos.

```
function [POPCROSS] = crossover_real(newpop,pc,range),

% ------------------------------------------------------------------------
% Realiza a Recombinação entre os cromossomos
```

```
% ------------------------------------------------------------------
% Sintaxe: [popcross] = crossover(newpop,popsize,nbits,pc)
%
% Parâmetros:
%
%                 pc                --> taxa de recombinação
%                 newpop            --> pop. selecionada
%
% Variável de Retorno: popcross
% ==================================================================
% Created by: João Bosco Gonçalves
% Date    : 03/01/2003
% Version  : 01
% ==================================================================
[popsize,C] = size(newpop);

prs = [];
k  = 0;
for (i = 1:popsize),
  r = rand;
  if (r<pc),
    k = k+1;
    prs(k,:) = newpop(i,:);
  end
end

if (isempty(prs)),
  newpop = popinit_real(popsize,C,range);
else
  [prssize,C] = size(prs);
  if (rem(prssize,2) > 0),
```

```
    prs(prssize+1,:) = newpop(prssize,:);
  end
  pares = randperm(prssize);
  for (i=1:2:prssize-1),
    temp1(1,:) = prs(pares(i+0),:);
    temp2(1,:) = prs(pares(i+1),:);

    newpop(pares(i+0),:) = pc*temp1 + (1-pc)*temp2;
    newpop(pares(i+1),:) = (1-pc)*temp1 + pc*temp2;
  end
end
POPCROSS = newpop;
```

A Tabela G4 apresenta a função *evalpop_real* utilizada para avaliar os indivíduos, gerando vetores com valores para a função de adequação e para a probabilidade acumulativa.

Tabela G4 – Função *evalpop_real.*

```
function [X] = evalpop_real(npv,to,pop,q,dqo,ddqo,op),
% ------------------------------------------------------------------
% Computa Função de Avaliação para a população de cromossomos
% ------------------------------------------------------------------
% Sintaxe: [F,Q] = evalpop(npv,nbits,njuntas,popsize,to,pop,range,dqo,ddqo)
% Onde:
%                 dqo d= [dq_i;dqo_f]          --> velocidade inicial
%                 ddqo = [ddq_i;ddqo_f]        --> aceleração inicial
%                 to = [t1;t2; ... tnpv]       --> tempo
%                 npv                                  --> número de pontos-via
%                 pop                                  --> população
%                 q  = [q1(t) q2(t) ... qn(t)] --> posições nos pontos-via
%
```

```
% Variável de Retorno: F(popsize) e Q(popsize)
%
%=========================================================================
% Created by: João Bosco Gonçalves
% Date     : 23/10/2002
% Version  : 1.1 (03/11/02)
%=========================================================================

[popsize,M] = size(pop);

pop = pop';

dq  = zeros(2+round(M/2),popsize);
ddq = zeros(2+round(M/2),popsize);

if (op==2),
  dq(1,:)   = dqo(1);
  dq(npv,:)  = dqo(2);
  dq(2:npv-1,:)  = pop(1:M/2,:);
  ddq(2:npv-1,:) = pop(((M/2)+1):M,:);
else
  dq = dqo';
  ddq(2:npv-1,:) = pop(1:M,:);
end

ddq(1,:)   = ddqo(1);
ddq(npv,:) = ddqo(2);


FT   = 0;
Q    = [];
```

```
opcao = 0;
for (i=1:popsize),
 F(i) = polig5r(to,npv-1,q,dq,ddq(:,i),opcao);
 % Calcula o fitness individual e Calcula o fitness total
 FT   = F(i) + FT;
end

for (i=1:popsize),
 % Calcula a probabilidade acumulativa, pi = eval(popi)/F
 P(i) = F(i)/FT;
 Q(i) = 0;
 for j=1:i,
  Q(i) = Q(i) + P(j);
 end
end
X = [F',Q'];
```

A Tabela G5 mostra a implementação computacional para a função de adequação.

Tabela G5 – Função de adequação.

```
function [F1] = polig5r(TC,npv,q,dq,ddq,op),
% ------------------------------------------------------------------------
% Computa a funçaõ-custo relativa à minimização de picos nas velocidades e acelerações
% ------------------------------------------------------------------------
% Sintaxe: f1 = polinômio(TS,TC,njuntas,q,dq,ddq)
% Onde:
%
%          TC          --> Tempo Decorrido
%          npv         --> No. de pontos-via
%          q           --> pontos-via (posição angular)
```

```
%                dq              --> velocidades angulares nos pontos-via
%                ddq             --> acelerações angulares nos pontos-via
%                op              --> op = 0, avalia individuo
%
% Variável de Retorno: fl
% ================================================================
% Created by: João Bosco Gonçalves
% Date    : 23/10/2002
% Version  : 1.1  (03/11/02)
% ================================================================

T    = [];
qc   = [];
dqc  = [];
ddqc = [];
dddqc = [];

for pv=1:npv,

  h = TC(pv+1)-TC(pv);

  ao =  q(pv);
  a1 =  dq(pv);
  a2 = ddq(pv)/2;

  alfa1 =  q(pv+1) - ao - a1*h - a2*h^2;

  alfa2 = dq(pv+1) - a1 - 2*a2*h;

  alfa3 = ddq(pv+1) - 2*a2;
```

```
  A = pinv([  h^3,   h^4,   h^5;...
        3*h^2, 4*h^3, 5*h^4;...
        6*h  ,12*h^2,20*h^3])*[alfa1;alfa2;alfa3];

  TS  = h/25;   % 200
  t   = TC(pv):TS:TC(pv+1);
  temp = t-TC(pv);

  QC  = ao + a1*temp + a2*(temp.^2) + A(1)*(temp.^3) + A(2)*(temp.^4) + ...
       A(3)*(temp.^5);
  DQC  = a1 + 2*a2*temp + 3*A(1)*(temp.^2) + 4*A(2)*(temp.^3) + 5*A(3)*(temp.^4);
  DDQC  = 2*a2 + 6*A(1)*temp + 12*A(2)*(temp.^2)+ 20*A(3)*(temp.^3);
  DDDQC = 6*A(1) + 24*A(2)*temp + 60*A(3)*(temp.^2);

  T  = [T,t];
  qc  = [qc,QC];
  dqc  = [dqc,DQC];
  ddqc = [ddqc,DDQC];
  dddqc = [dddqc,DDDQC];
end

if (op == 0),
  mu      = 0.5;
  tempqc1  = 0;
  tempqc2  = 0;
  tempdqc1  = 0;
  tempdqc2  = 0;
  tempddqc1 = 0;
  tempddqc2 = 0;
  tempdddqc1 = 0;
  tempdddqc2 = 0;
```

```
  NN = length(dqc);
  for k=1:(NN-1),
    tempqc1 = tempqc1 + (qc(k+1)-qc(k))^2;
    tempqc2 = tempqc2 + qc(k)^2;

    tempdqc1 = tempdqc1 + (dqc(k+1)-dqc(k))^2;
    tempdqc2 = tempdqc2 + dqc(k)^2;

    tempddqc1 = tempddqc1 + (ddqc(k+1)-ddqc(k))^2;
    tempddqc2 = tempddqc2 +  ddqc(k)^2;

    tempdddqc1 = tempdddqc1 + (dddqc(k+1)-dddqc(k))^2;
    tempdddqc2 = tempdddqc2 +  dddqc(k)^2;
  end
  tempqc2   = tempqc2 + qc(k+1)^2;
  tempdqc2  = tempdqc2 + dqc(k+1)^2;
  tempddqc2 = tempddqc2 + ddqc(k+1)^2;
  tempdddqc2 = tempdddqc2 + dddqc(k+1)^2;

  tempq=(tempqc1/(eps+tempqc2))+(tempdqc1/(eps+tempdqc2))+...
          (tempddqc1/(eps+tempddqc2))+(tempdddqc1/(eps+tempdddqc2));

 F1 = 1/(eps+sqrt(tempq));

elseif (op==3),
  F1 = [qc',dqc',ddqc',dddqc',T'];
else
  F1 = [qc',ddqc'];
end
if (op == 5)
   figure
```

```
color = ['b','r','k','m','g','c','y'];
pos   = randperm(7);
k     = fix(6*rand) + 1;

hold on
subplot(411),plot(T,qc,color(pos(k))),ylabel('qc')
grid on,hold off

hold on
subplot(412),plot(T,dqc,color(pos(k))),ylabel('dqc')
grid on,hold off

hold on
subplot(413),plot(T,ddqc,color(pos(k))),ylabel('ddqc')
grid on,hold off

hold on
subplot(414),plot(T,dddqc,color(pos(k))),ylabel('dddqc')
grid on,hold off
end
```

A Tabela G6 mostra a implementação computacional da função de mutação aplicada a indivíduos selecionados de acordo com os seus valores de aptidão à solução do problema, conforme mostra a Tabela G7.

Tabela G6 – Mutação de indivíduos

```
function [POPMUT] = mutation_real(newpop,gmax,g,pm,b,dimpop,range),

% ----------------------------------------------------------------------
% Aplica o operador de mutação a população corrente (após a recombinação)
```

```
% ------------------------------------------------------------------
% Sintaxe: [POPMUT] = mutation_real(newpop,pm)
% Onde:
%          newpop          --> pop. selecionada
%          pm              --> taxa de mutação,
%          gmax            --> No. máximo de gerações
%          g               --> geração atual
%          b               --> parâmetro de projeto
%
% Variável de Retorno: POPMUT
% ==================================================================
% Created by: João Bosco Gonçalves
% Date    : 20/01/2003
% Version  : 01
% ==================================================================

[popsize,C] = size(newpop);

poptemp = newpop;
p = randperm(popsize);

for (j=1:popsize),  % No.1

 for (i=1:C),      % No.3
  r = rand;
  if (r<0.5),      % No.4
   r = rand;
   newpop(p(j),i) = newpop(p(j),i) + ...
                    (max(newpop(p(j),i))-newpop(p(j),i))*(1-r^(1-g/gmax)^b);
  else
   r  = rand;
```

```
  newpop(p(j),i) = newpop(p(j),i) - ...
                   (newpop(p(j),i)-min(newpop(p(j),i)))*(1-r^(1-g/gmax)^b);
 end          % Fim No.4

 end          % Fim No.2
end           % Fim No.1
POPMUT = newpop;
```

Tabela G7 – Seleção dos melhores indivíduos.

```
Function [newpop] = popselect_real(pop,xq,range),
% ------------------------------------------------------------------------
% Escolhe uma nova população para aplicar os operadores genéticos
% Baseado na probabilidade acumulativa, q.
% Sintaxe: [NEWPOP] = popselect(pop,popsize,q)

% Parâmetros:
%                pop          --> população
%                popsize      --> tamanho da população
%                q            --> probabilidade acumulativa
%
% Variável de Retorno: newpop
%
% ========================================================================
% Created by: João Bosco Gonçalves
% Date     : 23/10/2002
% Version  : 01
% ========================================================================

[popsize,M] = size(pop);
```

```
k = 0;
newpop = [];
for i=1:popsize,
 r = rand;
 if (r < xq(i)),
  k = k+1;
  newpop(k,:) = pop(i,:);
 else
  for j=2:popsize,
   if ((r<=xq(j-1)) & (r>=xq(j))),
    k = k+1;
    newpop(k,:) = pop(i,:);
   end
  end
 end
end

dim = length(newpop);
d   = popsize-dim;
a   = 1;
if (d > 0),
 for(j=1:d),
  p = randperm(popsize);
  if (j>=popsize & a<popsize),
   k = p(a);
   a = a+1;
  elseif (a>=popsize),
   a = 1;
  else
   k = p(a);
   a = a+1;
```

```
 end
 newpop(dim+j,:) = pop(k,:);
 end
end

if (isempty(newpop)),
 newpop = popinit_real(popsize,M,range);
 disp('recomecei ...')
end
new_pop = newpop;
```

ANEXO H

Aciona o Gerador Automático de Marcha – *GAM*

A Tabela H1 mostra o arquivo para acionar o *GAM* (Anexo I) para o robô bípede. Foi utilizado o *software* Matlab® V5.3 R11.1 para geração do arquivo (extensão **.m**).

Tabela H1 – Aciona o *GAT*.

```
%=============================================================
% Movimento Planejado - 22/01/2004
%=============================================================
clear all ,close all, clc, pause(.1)

%=============================================================
% Parâmetros do Pé, m
%=============================================================
a = 0.090;
b = 0.220-a;

%=============================================================
% Parâmetros de Denavit-Hatemberg:
% DH = [Teta(i),Alfa(i),A(i),D(i)]
```

```
%==========================================================================
DH = [0   , pi/2,  0.061, 0;...      % SC#1 ( J2) em relação ao SC#0 (J1)
      0   ,  0,   0.316, 0;...       % SC#2 ( J3) em relação ao SC#0 (J1)
      0   ,  0,   0.316, 0;...       % SC#3 ( J4) em relação ao SC#0 (J1)
      0   ,-pi/2, 0.110, 0;...       % SC#4 ( J5) em relação ao SC#0 (J1)
      pi/2,  0,   0.316, 0;...       % SC#5 ( J6) em relação ao SC#0 (J1)
      pi/2,-pi/2, 0.110, 0;...       % SC#6 ( J7) em relação ao SC#0 (J1)
      0   ,  0,   0.316, 0;...       % SC#7 ( J8) em relação ao SC#0 (J1)
      0   ,  0,   0.316, 0;...       % SC#8 ( J9) em relação ao SC#0 (J1)
      0   , pi/2, 0.061, 0;...       % SC#9 (J10) em relação ao SC#0 (J1)
      0   ,  0,   0.036, 0];         % SC#10( FE) em relação ao SC#0 (J1)

%==========================================================================
% Parâmetros dos Elos, m
%==========================================================================
ao = 0.0386;
a1 = 0.0610;
a2 = 0.316;
a3 = 0.316;
a4 = 0.110;
a5 = 0.316;
A  = [a1,a2,a3,a4,a5,ao];
HT = ao+a1+a2+a3+a4;

%==========================================================================
%            Planejamento da Marcha
% O quadril é mantido paralelo ao solo
%==========================================================================
% Primeira Fase - da postura B para a D
q  = 0.2;
```

```
H   = ao;
lam = 0.17;
vx  = 0.55;

XQ  = [0;a5;HT-2*ao];              % Postura inicial do quadril
X2IB = [b*(1-cos(q));a5;b*sin(q)];
X2FB = [b*(1-cos(q));a5;a*sin(q)];

% Segunda Fase: da postura E para a D. Inverte a Ordem dos SC
X2ID = [-(lam+b*(1-cos(q)));a5;b*sin(q)];
X2FD = [ a*cos(q);a5;a*sin(q)];            % lam+a*cos(q)

%=========================================================
% Computa a trajetória desjada para o ZMP
%=========================================================
a5    = 0.316;  % comprimento do quinto elo
op    = 0;     % Posição ereta, P1 na origem do sistema de coordenadas fixo
zmpdesB = saida_padrao(30,op,lam,a5);
zmpdesD = saida_padrao(30,op,2*lam,a5);

%=========================================================
% Parâmetros de Projeto para o Algoritmo Genético
%=========================================================
np    = 3;              % No. de parâmetros
limites = [0 , 0.618];    % gamax
nbits  = np*fix(log((limites(2)-limites(1))*10^4)/log(2));  % determina o No. de bits
pm    = 0.2;
pc    = 0.8;
popsize = 10;
gmax   = 10;
```

```
%=========================================================================
% Gerador de Posturas
%=========================================================================
PB =
busca_bin(limites,pm,pc,nbits,popsize,gmax,vx,lam,H,zmpdesB,X2IB,X2FB,DH,XQ,q);
save PB.mat PB -ascii

XQ = PB(4:6);
PE = busca_bin(limites,pm,pc,nbits,popsize,gmax,vx,2*lam,H,...
                    zmpdesD,X2ID,X2FD,DH,XQ,q);
save PE.mat PE -ascii
```

ANEXO I

Implementação computacional para o *GAM*

A Tabela I1 mostra a implementação computacional para o *G*erador *A*utomático de *Marcha* (*GAM*) – o arquivo mostrado no Anexo H é utilizado para acioná-lo. A Tabela I2 mostra a função que converte os valores binários para valores reais (utilizando a função *decimal*) para cada indivíduo e, utilizando a função de adequação *xyz2teta*, prepara as informações para computar as componentes do *zmp*. A Tabela I3 mostra a implementação da função *avaliador* que computa o valor de adequação para cada indivíduo. Foi utilizado o *software* Matlab® V5.3 R11.1 para geração das funções (extensões **.m**).

Tabela I1 – Implementação computacional do *GAM*.

```
function [PB] = busca(limites,pm,pc,nbits,popsize,gmax,vx,passo,H,...
                      ZMPDES,X2I,X2F,dh,XQ,q),
% ------------------------------------------------------------------------
% Aplica o Algoritmo Genético ao problema de determinação do Vetor de Parâmetros
% ------------------------------------------------------------------------
% Sintaxe: [M] = gerador_marcha(range,pm,pc,popsize,gmax,qo,DH,T100)
%
% Parâmetros:
        range = [min,max,0 ou 1]     --> Faixa de Valores, R
```

```
%          pm                    --> Taxa de mutação
%          pc                    --> Taxa de recombinação
%          popsize               --> tamanho da população
%          gmax                  --> No. máximo de gerações
%          qo                    --> Inclinação do Pé
%          DH                    --> Denavit-Hartemberg
%          T100                  --> Matriz de Trans. H. de 10 à 0
%
% Variável de Retorno: M = [Q,DQ,DDQ]
% ========================================================================
% Created by: João Bosco Gonçalves
% Date      : 23/10/2003
% Version   : 01
% ========================================================================

% ------------------------------------------------------------------------
% Inicializar o Vetor de Parâmetros: O = (lambda,L,H,p5,alfa), Eq. (3.100)
% ------------------------------------------------------------------------
pop = popinit(popsize,nbits);
% ------------------------------------------------------------------------
% Inicializa a geração
% ------------------------------------------------------------------------
g = 0;

% ------------------------------------------------------------------------
% Avaliação do melhor indivíduo
% ------------------------------------------------------------------------
fprintf('  Iniciando o Gerador Automático de Marcha \n\n\n')
x       = check_bin(limites,pop,popsize,nbits,vx,passo,ZMPDES,H,X2I,X2F,dh,XQ,q);
[fmax,i] = max(x(:,1));
[fmin,j] = min(x(:,1));
```

```
fprintf('Evolução No. %d , Valor Máximo %2.15g e Valor Mínimo %2.15g\n',g,fmax,fmin)
pause(.2)

% ------------------------------------------------------------------------
% Evolução ao longo das Gerações
% ------------------------------------------------------------------------
while(g < gmax),

  % ------------------------------------------------------------------------
  % Próxima Geração
  % ------------------------------------------------------------------------
  g = g + 1;

  % ------------------------------------------------------------------------
  % Seleção elitista
  % ------------------------------------------------------------------------
  p  = pop(i,:);

  % ------------------------------------------------------------------------
  % Seleção da população
  % ------------------------------------------------------------------------
    newpop = popselect(pop,popsize,x(:,2));

  % ------------------------------------------------------------------------
  % Operador genético de recombinação
  % ------------------------------------------------------------------------
    popcross = crossover(newpop,popsize,nbits,pc);

  % ------------------------------------------------------------------------
  % Operador genético de mutação
  % ------------------------------------------------------------------------
```

```
    pop = mutation(popcross,pm);

  % ------------------------------------------------------------------------
  % Inclusão do melhor individuo à nova geração
  % ------------------------------------------------------------------------
  x = check_bin(limites,pop,popsize,nbits,vx,passo,ZMPDES,H,X2I,X2F,dh,XQ,q);
  [fmin,i] = min(x(:,1));
  pop(i,:) = p;
  x(i,1)   = fmax;

  % ------------------------------------------------------------------------
  % Avaliação do melhor indivíduo
  % ------------------------------------------------------------------------
  [fmax,i] = max(x(:,1));
  fprintf('Evolução No. %d , Valor Máximo %2.15g e Valor Minímo %2.15e\n',g,fmax,fmin)
  pause(.2)
end

% ========================================================================
% Armazena a Melhor Postura Básica
% ========================================================================
disp('finalizando ...')

[fmax,i] = max(x(:,1));
PB = pop(i,:);

% ========================================================================
% Parâmetros da Marcha
% ========================================================================
WS  = vx;       % Velocidade Horizontal na direção do movimento
SL  = passo;    % Comprimento do Passo
```

```
SP  = SL/WS;      % Periodo do Passo

%=========================================================================
%                    Tempo de Simulação
% Geração dos vetores de tempo com DT constante para cada Postura
%=========================================================================
npontos = length(ZMPDES)
DT      = SP/(npontos-1);
T       = SP;
t       = 0:DT:SP;

%=========================================================================
% Trajetória do Ponto X2, em metros
%=========================================================================
a5 = 0.316;  % comprimento do quinto elo
f  = 1/SP;
x2 = X2I(1) + (WS+(X2F(1)-X2I(1))*f)*(t-(1/(2*pi*f))*sin(2*pi*t*f));
y2 = a5*ones(1,length(t));
z2 = X2I(3) + ((X2F(3)-X2I(3))*f)*(t-sin(2*pi*t/SP)/(2*pi/SP))+0.5*H*(1-cos(2*pi*t/SP));

%=========================================================================
% Trajetória do Quadril, em metros
%=========================================================================
gamax = PB(1);
gamay = PB(2);
gamaz = PB(3);

xq =  gamax*x2 + XQ(1);
yq = -gamay*x2;
zq =  gamaz*z2 - H + XQ(3);
```

```
pos = length(xq)
XQ = [xq(pos),yq(pos),zq(pos)];

PB = [PB XQ];
```

Tabela I2 – Função de avaliação dos indivíduos.

```
function [J] = check(limites,pop,popsize,nbits,vx,passo,ZMPDES,H,X2I,X2F,dh,XQ,q),

FT = 0;
nb = nbits/3;
for (p=1:popsize),
  gamax = decimal(pop(p,1:nb),limites,nb);
  gamay = decimal(pop(p,nb+1:2*nb),limites,nb);
  gamaz = decimal(pop(p,2*nb+1:3*nb),limites,nb);
  xyz2teta_bin([gamax,gamay,gamaz],size(ZMPDES,1),vx,passo,H,X2I,X2F,dh,XQ,q);
  FJ(p) = avaliador_bin(ZMPDES); % avalia a adequação do indivíduo ao critério ZMP
  FT   = FT + FJ(p);          % Somatória da Função de Adequação
end

% ------------------------------------------------------------------------
% Calcula a probabilidade acumulativa, pi = eval(popi)/F
% ------------------------------------------------------------------------
for(p=1:popsize),
 P(p) = FJ(p)/FT;
 q(p) = 0;
 for j=1:p,
  q(p) = q(p) + P(j);
 end
end
J = [FJ',q'];
```

Tabela I3 – Função para preparação de dados.

```
function xyz2teta(individuo,npontos,ws,lambda,h,X2I,X2F,dh,XQ,q),

%=========================================================================
% Dados do Robô
%=========================================================================
robo_bipede2k3

%=========================================================================
% Parâmetros da Marcha
%=========================================================================
WS  = ws;          % Velocidade Horizontal na direção do movimento
SL  = lambda;      % Comprimento do Passo
SP  = SL/WS;       % Periodo do Passo

%=========================================================================
%                    Tempo de Simulação
% Geração dos vetores de tempo com DT constante para cada Postura
%=========================================================================
DT    = SP/(npontos-1);
T     = SP;
t     = 0:DT:SP;

%=========================================================================
% Discretização do ângulo q, rad
%=========================================================================
inc = 2*q/(length(t)-1);
q   = q:-inc:-q;
R1  = eye(3);
Ro  = eye(3);
```

```
%==========================================================================
% Trajetória do Ponto X2, em metros
%==========================================================================
f  = 1/SP;
x2 = X2I(1) + (WS+(X2F(1)-X2I(1))*f)*(t-(1/(2*pi*f))*sin(2*pi*t*f));
y2 = a5*ones(1,length(t));
z2 = X2I(3) + ((X2F(3)-X2I(3))*f)*(t-sin(2*pi*t/SP)/(2*pi/SP))+0.5*h*(1-cos(2*pi*t/SP));

%==========================================================================
% Trajetória do Quadril, em metros
%==========================================================================
gamax = individuo(1);
gamay = individuo(2);
gamaz = individuo(3);

xq =  gamax*x2 + XQ(1);
yq = -gamay*x2; % - gamay*cos(pi*(t-SP/2))/(pi*f*T);
zq =  gamaz*z2 - h + XQ(3);  %a1 + a2 + a3 + ao;

%xq(1) = -XQ(1);
%yq(1) =  XQ(2);
%zq(1) =  XQ(3);

%==========================================================================
% Preparação dos Dados
%==========================================================================
ph1 = [xq;   yq; zq];
ph2 = [xq; a5+yq; zq];
X2  = [x2;   y2; z2];
X1  = [0;0;0];
```

```
%=============================================================
% Aplicação da Cinemática Inversa para computar os ângulos de juntas
% da Perna de Apoio (teta1) e da Perna em Balanço (teta2)
%=============================================================

[lin,col] = size(x2);

ao   = dh(10,3);
agora = 1;
close

for(k=1:col),

  Q1(:,1) = (cinversa(A,R1,Ro,ph1(:,k),X1))';
  Q2(:,1) = (cinversa(A,Ry(q(k)),Ro,ph2(:,k),X2(:,k)))';

  %=============================================================
  % Computo da Localização Carteziana dos sistemas de coordenadas móveis:
  % centro do corpo (Xo) e das juntas das pernas, exceto dos quadris
  %=============================================================
  TETA     = [Q1(1),Q1(2),Q1(3),Q1(4),Q1(5),Q2(5),Q2(4),Q2(3),Q2(2),Q2(1)]';
  TETAK(:,k) = TETA;

  dh(1  ,1) =  TETA(1,1);
  dh(2:4,1) =  TETA(2:4,1);
  dh(7:9,1) = -TETA(7:9,1);
  dh(5  ,1) =  pi/2+TETA(5,1);
  dh(6  ,1) =  pi/2-TETA(6,1);
  dh(10 ,1) =  TETA(10,1);

  A1  = ptrans(dh,1);z1  = A1(1,4)+ao;  y1  = A1(2,4);  x1  = -A1(3,4);
```

```
P1(:,k) = [x1;y1;z1];

A2 = ptrans(dh,2);z2 = A2(1,4)+ao; y2 = A2(2,4); x2 = -A2(3,4);
P2(:,k) = [x2;y2;z2];

A3 = ptrans(dh,3); z3 = A3(1,4)+ao; y3 = A3(2,4); x3 = -A3(3,4);
P3(:,k) = [x3;y3;z3];

A4 = ptrans(dh,4); z4 = A4(1,4)+ao; y4 = A4(2,4); x4 = -A4(3,4);
P4(:,k) = [x4;y4;z4];

A5 = ptrans(dh,5); z5 = A5(1,4)+ao; y5 = A5(2,4); x5 = -A5(3,4);
P5(:,k) = [x5;y5;z5];

Xo(:,k) = P5(:,k) - Ro*[0;dh(5,3)/2;0];

A6 = ptrans(dh,6); z6 = A6(1,4)+ao; y6 = A6(2,4); x6 = -A6(3,4);
P6(:,k) = [x6;y6;z6];

A7 = ptrans(dh,7); z7 = A7(1,4)+ao; y7 = A7(2,4); x7 = -A7(3,4);
P7(:,k) = [x7;y7;z7];

A8 = ptrans(dh,8); z8 = A8(1,4)+ao; y8 = A8(2,4); x8 = -A8(3,4);
P8(:,k) = [x8;y8;z8];

A9 = ptrans(dh,9); z9 = A9(1,4)+ao; y9 = A9(2,4); x9 = -A9(3,4);
P9(:,k) = [x9;y9;z9];

A10 = ptrans(dh,10);z10 = A10(1,4)+ao; y10 = A10(2,4); x10 = -A10(3,4);
P10(:,k) = [x10;y10;z10];
```

```
  if (k==agora),
    hold off,subplot(121),caminhaXZ(dh,a,b,q(k)),title(num2str(k)),hold on,pause(.1)
    hold off,subplot(222),caminhaYZ(dh,a,b,q(k)),title(num2str(k)),hold on,pause(.1)
    agora = k+2;
  end

end

% ================================================================
% Computo Numérico das Velocidades e Acelerações
% ================================================================
N     = length(P1);
Axyz_1 = ddyc([P1,P1(:,N-1:N);P2,P2(:,N-1:N);P3,...
                    P3(:,N-1:N);P4,P4(:,N-1:N);P5,P5(:,N-1:N)],DT);
Axyz_2 = ddyc([P6,P6(:,N-1:N);P7,P7(:,N-1:N);P8,...
                    P8(:,N-1:N);P9,P9(:,N-1:N);P10,P10(:,N-1:N)],DT);
Axyz_c = ddyc([Xo,Xo(:,N-1:N)],DT);

Vxyz_10 = velocidade ([P10,P10(:,N)],DT);
Vxyz_c  = velocidade ([Xo,Xo(:,N)],DT);

% ----------------------------------------------------------------
% Informações para o gerador de trajetórias - converte_pb_traj
% ----------------------------------------------------------------
save 'c:\MATLABR11\work\gam\gam-binario\arq\MTVQ.mat'   Vxyz_c  -ascii -double
save 'c:\MATLABR11\work\gam\gam-binario\arq\TETA.mat'   TETAK   -ascii -double
save 'c:\MATLABR11\work\gam\gam-binario\arq\VTP10.mat'  Vxyz_10 -ascii -double
save 'c:\MATLABR11\work\gam\gam-binario\arq\vtempo.mat' t       -ascii -double
% ================================================================
Montagem da Matriz de Treinamentos para a RNMLP
% ================================================================
```

```
mtp1 = [P1;P2;P3;P4;P5];
mtp2 = [P6;P7;P8;P9;P10];
ppo  = Xo';

mta1 = Axyz_1;
mta2 = Axyz_2;

mtp  = [mtp1;mtp2]';
mta  = [mta1;mta2]';
apo  = Axyz_c';

save 'c:\MATLABR11\work\gam\gam-binario\arq\MTP.mat'  mtp  -ascii -double
save 'c:\MATLABR11\work\gam\gam-binario\arq\MTA.mat'  mta  -ascii -double
save 'c:\MATLABR11\work\gam\gam-binario\arq\MTPQ.mat'  ppo  -ascii -double
save 'c:\MATLABR11\work\gam\gam-binario\arq\MTAQ.mat'  apo  -ascii -double
```

Tabela I4 – Função de Adequação.

```
function FC = avaliador(ZMPDES),
% ==========================================================================
% Dados do Robô
% ==========================================================================
robo_bipede2k3;  MASS = mass; GRAV = grav; Alt_tronco = 0.618;

load 'c:\MATLABR11\work\gam\gam-binario\arq\MTP.mat' -ascii
load 'c:\MATLABR11\work\gam\gam-binario\arq\MTA.mat' -ascii
load 'c:\MATLABR11\work\gam\gam-binario\arq\MTPQ.mat' -ascii
load 'c:\MATLABR11\work\gam\gam-binario\arq\MTAQ.mat' -ascii

POSX = [MTP(:,1)  MTP(:,4)  MTP(:,7) MTP(:,10) MTP(:,13) MTP(:,16) ...
        MTP(:,19) MTP(:,22) MTP(:,25) MTP(:,28)];
```

```
POSY = [MTP(:,2)  MTP(:,5)  MTP(:,8) MTP(:,11) MTP(:,14) MTP(:,17) ...
        MTP(:,20) MTP(:,23) MTP(:,26) MTP(:,29)];
POSZ = [MTP(:,3)  MTP(:,6)  MTP(:,9) MTP(:,12) MTP(:,15) MTP(:,18) ....
       MTP(:,21) MTP(:,24) MTP(:,27) MTP(:,30)];

ACEX = [MTA(:,1)  MTA(:,4)  MTA(:,7) MTA(:,10) MTA(:,13) MTA(:,16)....
        MTA(:,19) MTA(:,22) MTA(:,25) MTA(:,28)];
ACEY = [MTA(:,2)  MTA(:,5) MTA(:,8) MTA(:,11) MTA(:,14) MTA(:,17) ....
        MTA(:,20) MTA(:,23) MTA(:,26) MTA(:,29)];
ACEZ = [MTA(:,3)  MTA(:,6)  MTA(:,9) MTA(:,12) MTA(:,15) MTA(:,18) ....
        MTA(:,21) MTA(:,24) MTA(:,27) MTA(:,30)];

PPO  = MTPQ;
APO  = MTAQ;

POST = zeros(size(PPO));
ACET = POST;
JX = 0; JY = 0;

N_seg  = 10;
N_amos = length(PPO);
[lin,col] = size(POSX);

for(L=1:lin),
  Tx1 = 0;
  Ty1 = 0;
  for(k=1:N_seg),

    Tx1 = Tx1 - MASS(k)*(ZMPDES(L,2)*(ACEZ(L,k)+GRAV(3)) - ...
                POSY(L,k)*(ACEZ(L,k)+GRAV(3)) +
POSZ(L,k)*(ACEY(L,k)+GRAV(2)));
```

```
    Ty1 = Ty1 + MASS(k)*(ZMPDES(L,1)*(ACEZ(L,k)+GRAV(3)) - ...
                POSX(L,k)*(ACEZ(L,k)+GRAV(3)) +
POSZ(L,k)*(ACEX(L,k)+GRAV(1)));

  end;

  TX(L,1) = Tx1 + MASS(N_seg+1)*(GRAV(3) - (POST(L,1) + PPO(L,1))*GRAV(3) + ...
            (Alt_tronco+PPO(L,3))*(ACET(L,1)+GRAV(1)));

  TY(L,1) = Ty1 + MASS(N_seg+1)*(GRAV(3) - (POST(L,2) + PPO(L,2))*GRAV(3) + ...
        (Alt_tronco+PPO(L,3))*(ACET(L,2)+GRAV(2)));

  JX = JX + TX(L,1)^2;
  JY = JY + TY(L,1)^2;

End
% ==========================================================
% Computa a Função Custo
% ==========================================================

J  = 0.5*norm([JX;JY])./lin;
FC = 1/(eps+sqrt(J));
subplot(224),hold on,plot(TX(:,1)),plot(TY(:,1)),grid,title(num2str(FC)),hold off,pause(3)
```

ANEXO J

Implementação computacional do *OLS*

A Tabela J mostra a implementação computacional do *OLS* que permite escolher centros significativos de um conjunto de centros candidatos, para o compor uma rede RBF. Foi utilizado o *software* Matlab® V5.3 R11.1 para geração das funções (extensões **.m**).

Tabela J – Implementação computacional do algoritmo *Orthogonal Least Square.*

```
function [c] = ols_mimo(X,Y,tol),
% Emprega o algoritmo Orthogonal Least Square (OLS) para
% definir um conjunto de centros (função de base radial)
% entre todos os centros canditados.
% Argumentos da função:
%       X = [y1(k-1) y1(k-2)...y1(k-ny) y2(k-1) y2(k-2)... ym(k-1)...ym(k-ny)...
%            u1(k-1) u1(k-2)...un(k-1)...un(k-nu)];
%       Y = [y1(t) y2(t)...ym(t)]   -> saída desejada e t=ti:DT:tf pontos
% Onde: n -> número de entradas; m -> número de saídas; nu e ny -> número de pontos

[N,m] = size(Y);        % N pontos amostrais e m saídas
% Passo #6: Determinação do primeiro centro
for i=1:N,
```

```
  gama   = 0;
  p(:,i) = bf(X,i);
  w1     = p(:,i);
  w1t    = w1';
  for r=1:m,
    gama = gama + (w1t*Y(:,r)/(w1t*w1))^2;
  end
  err1(i) = gama*w1t*w1/trace((Y')*Y);
end
[e(1),ik(1)] = max(err1);    %armazena a coluna que forneceu o centro
sumerr = e(1);
fprintf(' iteracao       erro \n\n');
fprintf('   %d        %4.10f\n',1,1-e(1));
w(:,1) = p(:,ik(1));
c(1,:) = X(ik(1),:);

% Passo #7: Determinação dos demais centros (que minimizam o erro)
op1 = 1;
k   = 2;
while(op1),
 errk = [];
  for i=1:N,
    op2 = 1;
    for cont=1:length(ik),
      if (i==ik(cont)),
        op2 = 0;
      end
    end
    if (op2),
      bsum = zeros(size(w(:,1)));
      for j=1:(k-1),
```

```
        wjt = w(:,j)';
        beta = wjt*p(:,i)/(wjt*w(:,j));
        bsum = bsum + beta*w(:,j);
      end
      wk  = p(:,i) - bsum;
      wkt = wk';          gama = 0;
      for r=1:m,
        gama = gama + (wkt*Y(:,r)/(wkt*wk))^2;
      end
      errk(i) = gama*wkt*wk/trace((Y')*Y);
    end
  end
  [e(k),ik(k)] = max(errk);
  bsum = zeros(size(p(:,i)));
  for j=1:(k-1),
    wjt = w(:,j)';
    beta = wjt*p(:,ik(k))/(wjt*w(:,j));
    bsum = bsum + beta*w(:,j);
  end
  w(:,k) = p(:,ik(k)) - bsum;
  c(k,:) = X(ik(k),:);
  sumerr = sumerr + e(k);
  rse   = abs(1 - sumerr);
  fprintf('  %d        %4.10f\n',k,rse);
  if (rse<=tol),
    op1 = 0;
  else
    k = k+1;
  end
end
```

```
function [rbf] = bf(X,pos),
% X = [x(1) x(2) x(3) x(4) .... x(N)]: padrões de entrada
[L,C]     = size(X);
dmax      = max(max(X));
sg        = dmax/sqrt(2*L);
arg(1,1) = X(1,1);           % bias
for i=2:L,
   arg(i,1) = 1/sqrt(sg^2 + norm(X(pos,:) - X(i,:)));
end
rbf = [arg];
```

ANEXO L

Implementação computacional para as Perturbações

A Tabela L1 mostra a implementação computacional da perturbação que o tronco provoca aos membros inferiores do robô bípede e a Tabela L2, mostra a implementação do caso contrário. Foi utilizado o *software* Matlab® V5.3 R11.1 para geração das funções (extensões **.m**).

Tabela L1 – Perturbação_Tronco.

```
function P = Pertubacao_Tronco(u),

L = [0.05,0.95];   % Localização do CoM na direção X, medida no CSPELVI

M = [.098,5.7];    % [M11,M12] Massa em kg

q11 = u(5);  q12 = u(6);  dq11 = u(1);  dq12 = u(2);ddq11 = u(3);ddq12 = u(4);

P = M(2)*(L(1)+L(2))*cos(q12)*ddq12 - M(2)*cos(q12)*(L(1)+L(2))*dq12^2;
```

Tabela L2 – Perturbação_MMI

```
function [ACEZ] = Centro_Massa(u),

load c:\matlabr11\work\controlador_rb\dados_m\DH.m -ascii
load c:\matlabr11\work\controlador_rb\dados_m\VEL.m -ascii

CoMGlobal = [42.79;-158.00;-12.43]*1e-3;  DT = 1e-3; Q = u;

A4 = pjac(DH,1,4); % Computa a localização do SC4 em relação à base

CoMBODY1 = A4(1:3,4) - CoMGlobal;  %  Localização do CoM global do Robô Bípede
DH(:,1)        = Q(1:10,1);      % Atualiza o vetor ângular
A4             = pjac(DH,1,4);
CoMBODY2 = A4(1:3,4) - CoMGlobal;

VELATUAL = (CoMBODY2 - CoMBODY1)/DT; % Computa a velocidade

ACE = (VELATUAL - VEL)/DT; % Computa a aceleração
save c:\matlabr11\work\controlador_rb\dados_m\DH.m DH -ascii
save c:\matlabr11\work\controlador_rb\dados_m\VEL.m VELATUAL -ascii

ACEZ = ACE(3);
```

ANEXO M

Implementação computacional da Rede RBF

A Tabela M mostra a implementação computacional da Rede RBF projetada para identificar as parcelas não consideradas no modelo analítico referente aos membros inferiores do robô bípede. Foi utilizado o *software* Matlab® V5.3 R11.1 para geração das funções (extensões **.m**).

Tabela M – Implementação da Rede RBF para os membros inferiores do robô bípede.

```
function [sys,x0,str,ts] = GAMA_RBF(t,x,u,flag)
% RNBR1 implementa uma Rede Neural de Base Radial para o RB-01 (rbipede1.m).
%  Este M-file S-function implementa o seguinte modelo discreto (RNBR):
%
%          |y1(k) |  |N11 N21 ... Nm1| |FBR1|
%          |y2(k) |= |N12 N22 ... Nm2| |FBR2|
%          |   :  | |    :     ||  :|
%        |y14(k)|  |N17 N27 ... Nm7| |FBRm|
%
% Ou, de forma compacta:
%
%    {y(k)} = [NL]{FBR}
```

```
%
%  Copyright (c) Jun, 2002 by João BOsco Gonçalves,All Rights Reserved.
%  $Revision: 1.0 $
%
% ------------------------------------------------------------------------
% Carrega os arquivos:
%    [NL]  parâmetros (pesos) devido a parte não-linear do modelo
%    [CE]  centros da RBF
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Define o método a ser empregado pelo valor da variável "flag"
% ------------------------------------------------------------------------
switch flag,

 % ==========================================================================
 % Inicialização do Modelo
 % ==========================================================================
 case 0,
  [sys,x0,str,ts] = mdlInitializeSizes;



 % ==========================================================================
 % Atualização da Saída do Modelo da RBF
 % ==========================================================================
 case 3,
   sys = mdlOutputs(u);

 % ==========================================================================
 % Casos Não Utilizados
 % ==========================================================================
```

```
  case { 1, 2, 4, 9 },
    sys = [];

  %==========================================================================
  % Flags não-esperados
  %==========================================================================
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end

%------------------------------------------------------------------------
% Impementação dos Métodos, definidos pelo valor da variável "flag"
%------------------------------------------------------------------------

%==========================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%==========================================================================
function [sys,x0,str,ts] = mdlInitializeSizes,

    sizes = simsizes;
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 10;
    sizes.NumInputs      = 60;      % u = [e(20x1) Xmr(20x1) Uref(20x1) x(20x1)]
    sizes.DirFeedthrough = 0;
    sizes.NumSampleTimes = 1;

    sys = simsizes(sizes);
    x0  = [];
    str = [];
```

```
    ts = [0.001 0];

%=============================================================
% mdlOutputs
% Return Return the output vector for the S-function
%=============================================================
function sys = mdlOutputs(u),

load c:\matlabr11\work\controlador_rb\dados_m\P.m -ascii % Sol. da Eq. de Lyapunov
load c:\matlabr11\work\controlador_rb\dados_m\centros.m -ascii  % (centros da RBF)

[L,C] = size(centros);
X   = [1;u(10:60,1)];                    % Vetor de excitação à rede RBF

B   = [zeros(10,10);eye(10,10)]; % Modelo Nominal
e   = u(1:20,1);                 % Erro de Aproximação
PI  = 0.1;

for k=1:10,
  FBR    = basis_func(X,centros',L);
  NLA    = -PI*(FBR')*(e')*P*B(:,k);      % Atualização dos Pesos, Eq.(4.26)
  GAMA(k,1) = FBR*NLA;
end

sys = GAMA;

% ------------------------------------------------------------------
% Implementação dos Métodos utilizados por "mdlUpdate"
% ------------------------------------------------------------------

%=============================================================
```

```
%  Calcula o valor da função base radial para o passo corrente
%       X = [x(1) x(2) x(3) x(4) .... x(N)] -> vetor linha contento padrões de entrada à RBF
% ==========================================================
function [arg] = basis_func(X,CE,L),

D      = max(max(CE));
sg     = D/sqrt(2*L);
arg(1,1) = 1;
for i=1:L,
  arg(1,i+1) = 1/sqrt(sg^2 + norm(X - CE(:,i)));
end
```

ANEXO N

Implementação computacional da Rede RBF

A Tabela N mostra a implementação computacional da Rede RBF projetada para identificar as parcelas não consideradas no modelo analítico do tronco do robô bípede. Foi utilizado o *software* Matlab® V5.3 R11.1 para geração das funções (extensões **.m**).

Tabela N – Implementação da Rede RBF para o tronco do robô bípede.

```
function [sys,x0,str,ts] = GAMARBFTR(t,x,u,flag)
% RNBR1 implementa uma Rede Neural de Base Radial para o RB-01 (rbipede1.m).
%  Este M-file S-function implementa o seguinte modelo discreto (RNBR):
%
%          |y1(k) |  |N11 N21 ... Nm1| |FBR1|
%          |y2(k) |= |N12 N22 ... Nm2| |FBR2|
%          |   :  | |    :     | | :|
%         |y14(k)|  |N17 N27 ... Nm7| |FBRm|
%
% Ou, de forma compacta:
%
%    {y(k)} = [NL]{FBR}
%
```

```
%   Copyright (c) Jun, 2002 by João BOsco Gonçalves,All Rights Reserved.
%   $Revision: 1.0 $
%
% ------------------------------------------------------------------------
% Carrega os arquivos:
%    [NL]  parâmetros (pesos) devido a parte não-linear do modelo
%    [CE]  centros da RBF
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Define o método a ser empregado pelo valor da variável "flag"
% ------------------------------------------------------------------------
switch flag,

 % ==========================================================================
 % Inicialização do Modelo
 % ==========================================================================
 case 0,
  [sys,x0,str,ts] = mdlInitializeSizes;

% ==========================================================================
% Atualização da Saída do Modelo da RBF
% ==========================================================================
case 3,
   sys = mdlOutputs(u);

% ==========================================================================
% Casos Não Utilizados
% ==========================================================================
case { 1, 2, 4, 9 },
  sys = [];
```

```
% ==============================================================
% Flags não-esperados
% ==============================================================
otherwise
  error(['unhandled flag = ',num2str(flag)]);
end

% ---------------------------------------------------------------------------
% Impementação dos Métodos, definidos pelo valor da variável "flag"
% ---------------------------------------------------------------------------

% ==============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
% ==============================================================================
function [sys,x0,str,ts] = mdlInitializeSizes,

    sizes = simsizes;
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 2;
    sizes.NumInputs      = 12;     % u = [e(4x1) Xmr(4x1) x(4x1)]
    sizes.DirFeedthrough = 0;
    sizes.NumSampleTimes = 1;

    sys = simsizes(sizes);
    x0  = [];
    str = [];
    ts  = [0.001 0];

%==============================================================================
```

```
% mdlOutputs
% Return Return the output vector for the S-function
%=============================================================================
function sys = mdlOutputs(u),

load c:\matlabr11\work\controlador_rb\dados_m\PTR.m  -ascii  % Sol. da Eq. de Lyapunov
load c:\matlabr11\work\controlador_rb\dados_m\centrostr.m -ascii    % (centros da RBF)

[L,C] = size(centrostr);
X    = [1;u(4:12,1)];                  % Vetor de excitação à rede RBF

B    = [zeros(2,2);eye(2,2)];       % MOdelo Nominal
e    = u(1:4,1);                       % Erro de Aproximação
PI   = 0.1;

for k=1:2,
  FBR     = basis_func(X,centrostr',L);
  NLA     = -PI*(FBR')*(e')*PTR*B(:,k);      % Atualização dos Pesos, Eq.(4.26)
  GAMA(k,1) =  FBR*NLA;
end

sys = GAMA;

% -----------------------------------------------------------------------------
% Implementação dos Métodos utilizados por "mdlUpdate"
% -----------------------------------------------------------------------------

%=============================================================================
%  Calcula o valor da funcção base radial para o passo corrente
%       X = [x(1) x(2) x(3) x(4) .... x(N)] -> vetor linha contento padrões de entrada da RBF
%=============================================================================
```

```
function [arg] = basis_func(X,CE,L),


arg(1,1) = 1;
for i=1:L,
  D       = max(max(max(CE(:,i))));
  sg      = D/sqrt(2*L);
  arg(1,i+1) = 1/sqrt(sg^2 + norm(X - CE(:,i)));
end
```