UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

TIAGO DA CONCEIÇÃO MOTA

Análise e Proposta de Controladores para Navegação Autônoma de um Robô Inteligente

Prof. Ph.D. Adriano Joaquim de Oliveira Cruz
Orientador

Rio de Janeiro, Maio de 2010

Ficha Catalográfica

Mota, Tiago da Conceição

Análise e Proposta de Controladores para Navegação Autônoma de um Robô Inteligente / Tiago da Conceição Mota. – Rio de Janeiro: UFRJ IM, 2010.

131 f.: il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro. Programa de Pós-Graduação em Informática, Rio de Janeiro, BR–RJ, 2010.

Orientador: Adriano Joaquim de Oliveira Cruz.

1. Inteligência Computacional. 2. Aprendizado de Máquina. 3. Navegação de Robô. 4. Robô Autônomo. I. Cruz, Adriano Joaquim de Oliveira. II. Título.

Análise e Proposta de Controladores para Navegação Autônoma de um Robô Inteligente

Tiago da Conceição Mota

Dissertação de Mestrado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Mestre em Informática.

Aprovado por:

_____

Prof. Ph.D. Adriano Joaquim de Oliveira Cruz (Orientador)

_____

Prof. Ph.D. Antonio Carlos Gay Thomé

_____

Prof. Docteur Josefino Cabral Melo Lima

_____

Profa. D.Sc. Karla Tereza Figueiredo Leite

Rio de Janeiro, Maio de 2010

AGRADECIMENTOS

A minha família, em especial aos ausentes, pois nela buscamos o apoio nas ocasiões mais difíceis. Sem ela certamente não teria alcançado as vitórias de até então.

A minha esposa Fernanda, pela compreensão dos momentos de ausência e pelo incentivo na realização do trabalho.

À Universidade Federal do Rio de Janeiro, seus professores e seus funcionários, pela excelente qualidade de ensino e pelos valores transmitidos.

Em particular, ao professor Adriano Cruz, pela extrema paciência, generosidade e confiança. Seus ensinamentos, não só como professor e orientador mas também como amigo, serão levados comigo pelo resto da vida.

Aos colegas de laboratório, pelo apoio ao longo da confecção do trabalho e pelos momentos de descontração, e com destaque ao amigo Vinícius "Cabessa", pelas produtivas discussões sobre diversas ideias para o trabalho.

Ao Centro de Pesquisas e Desenvolvimento Leopoldo Américo Miguez de Mello (CENPES), em parceria com a COPPETEC, pelo provimento da bolsa que apoiou a realização deste trabalho.

RESUMO

O trabalho de (MORATORI, 2006) teve como produto um robô virtual, capaz de se locomover num ambiente desviando de obstáculos até alcançar seu objetivo de chegar ao outro lado de um corredor. O robô era guiado por um controlador nebuloso, que recebia como entradas os valores lidos por seus sensores. Porém, ao tentarmos introduzir este controlador num robô real, afim de que este se comportasse da mesma forma que o virtual, esbarramos na dificuldade de que alguns de seus sensores não estavam disponíveis no mundo real e sua implementação com sensores mais simples seria complexa e poderia não trazer os resultados esperados.

Neste trabalho, atacamos o mesmo problema de (MORATORI, 2006), porém tentando focar na posterior implementação de um robô real. Desta forma, redefinimos os sensores utilizados pelo robô, além de utilizarmos um esquema de memória e pré-processamento que fornece como entrada para o controlador informações mais relevantes às tomadas de decisão.

Propomos a criação de um novo controlador baseado em Lógica Nebulosa, além de utilizarmos os conhecimentos contidos tanto neste como no antigo controlador para a geração de outros controladores, que empregam outros modelos de Inteligência Computacional e Aprendizado de Máquina.

Por fim, mostramos os resultados de testes com simulações, nas quais utilizamos os diferentes controladores criados durante o trabalho para guiar o robô. Realizamos, ainda, uma análise dos resultados sob diversos aspectos, de forma a decidir qual controlador seria o mais adequado para ser implementado num robô real.

Palavras-chave: Inteligência Computacional, Aprendizado de Máquina, Navegação de Robô, Robô Autônomo.

Building a Controller for Navigation of an Intelligent Autonomous Robot

ABSTRACT

The work described in (MORATORI, 2006) had as product a virtual robot, that was able to move in an environment avoiding obstacles until reaching its objective: the other side of the corridor. The robot was guided by a fuzzy controller, that took as inputs the values read by its sensors. However, when we tried to apply this controller in a real robot, in a way that this one could behave as the virtual one, we ran against the difficulty that some of its sensors were not avaiable on the real world and their implementation using simpler sensors would be complex and could not bring the desired results.

In this work, we tackle the same problem described in (MORATORI, 2006), but trying to focus on the later implementation of a real robot. In this way, besides redefining the sensors used by the robot, we make use of an scheme of memory and pre-processor that provides as input to the controller information that is more relevant to the decision making.

We propose creating a new controller based on Fuzzy Logic, and also to use the knowledge from this controller and the old one to generate other controllers, that employ other models of Computational Intelligence and Machine Learning.

At last, we show results from tests by simulation, in which we use the controllers created during this work to guide the robot. We also analyze the results from several points of view, in order to decide which controller would be more suitable to be implemented on a real robot.

Keywords: Robot Navigation, Autonomous Robot, Computational Intelligence.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ANFIS	*Adaptive Neuro-Fuzzy Inference System*
MLP	*Multi-Layer Perceptron*
RBF	*Radial Basis Function*
RNA	Redes Neurais Artificiais
SVM	*Support Vector Machine*

SUMÁRIO

1 INTRODUÇÃO

Nos dias atuais, é notável a participação de robôs nos mais diversos campos, desde indústrias, passando por setores de serviços e chegando, até mesmo, no ambiente doméstico. As vantagens da utilização de robôs, além do fascínio já de longa data que estes exercem sobre os homens (ASIMOV, 2005), sempre fizeram com que a humanidade estivesse perseguindo a criação de robôs cada vez melhores em diversos aspectos.

Assim, muito se tem pesquisado nas últimas décadas sobre construção de robôs autônomos, em especial inteligentes (CALVO, 2007; HEINEN, 2007; MORATORI et al., 2005; SHI; DUNLAP; COLLINS, 2007). Estes robôs são capazes de desempenhar tarefas que seriam realizadas por humanos, sobretudo relacionadas a reconhecimento de padrões e tomadas de decisão (BEOM; CHO, 1995; NEARCHOU, 1998; XIAO et al., 1997).

Apenas como exemplo, podemos citar a utilização de robôs na execução de tarefas perigosas, em particular na exploração de locais inóspitos, como oleodutos, gasodutos e plataformas de exploração de petróleo.

Acreditamos, então, ser de grande utilidade o aprimoramento da pesquisa referente ao projeto e construção de robôs autônomos inteligentes, e por isso decidimos abordar o problema que descrevemos a seguir.

1.1 Problema

Neste trabalho, nos focamos no problema da locomoção de um robô autônomo de um local de origem a um local destino pré-definido, evitando possíveis obstáculos encontrados no cenário.

Mais especificamente, temos a situação em que o robô se encontra num corredor retangular, inicialmente junto à parede oposta ao único lado onde não há parede impedindo a saída. O objetivo do robô é sair do corredor, ou seja, alcançar o lado deste que não possui parede, evitando os possíveis obstáculos fixos encontrados.

Este problema, colocado desta forma, já foi abordado por (MORATORI, 2006), num estudo sobre um robô virtual autônomo. Naquele trabalho, foi criado para o robô um controlador baseado em Lógica Nebulosa, utilizando determinadas medidas obtidas pelos sensores do robô para a tomada de decisão.

Porém, ao tentarmos inserir este controlador num robô real, esbarramos na dificuldade de que dois dos três sensores do robô virtual (posição no corredor e distância ao obstáculo mais próximo na faixa à frente do robô) não são facilmente implementáveis a partir dos sensores existentes no mundo real. Normalmente, dispomos apenas de sensores de distância (distância ao obstáculo mais próximo na linha apontada pelo sensor) e de bússola (orientação em relação ao polo norte magnético). Por vezes, dispomos também de sensores de posição; no entanto, tais sensores não podem ser utilizados neste trabalho, já que sua precisão não alcança o necessário para que as

tomadas de decisão sejam corretas.

Assim, para que o controlador do robô virtual construído por (MORATORI, 2006) seja utilizado num robô real, devemos escolher novos sensores para o robô, além de adaptar o controlador para que este utilize como entradas as medidas destes novos sensores.

1.2 Soluções Propostas

Primeiramente, a solução proposta neste trabalho passa pela escolha dos sensores que irão compor o novo robô, levando em conta que temos disponíveis apenas sensores de distância e de bússola.

Após esta etapa, uma possível solução seria posicionar sensores de distância de modo a simular o comportamento dos sensores de posição e distância na faixa utilizados em (MORATORI, 2006). Esta solução, porém, foi descartada logo de início, pois teríamos de usar uma grande quantidade de sensores para que a informação fosse suficiente para realizar esse mapeamento satisfatoriamente.

Decidimos, então, que o melhor caminho seria a criação de um novo controlador, o qual tentaria imitar o comportamento do antigo controlador em cada situação, porém utilizando as medidas obtidas através dos novos sensores. Dada a oportunidade, optamos, ainda, por testar diferentes modelos de Inteligência Computacional para a construção deste novo controlador a partir de técnicas de aprendizado.

Além disso, dado que o antigo controlador havia sido construído manualmente, decidimos construir um novo controlador também de forma manual, utilizando Lógica Nebulosa. Mais do que isso, aplicamos também o mesmo conjunto de modelos citado

acima para obtermos novos controladores a partir do conhecimento encontrado no construído manualmente, com o objetivo de compará-los.

A metodologia de trabalho consistiu em construir controladores e realizar simulações para estudar o comportamento do robô virtual em diversos cenários. Para isso, realizamos os seguintes passos:

- Estudo e avaliação do antigo robô virtual;
- Criação de um novo ambiente de simulação e de cenários virtuais;
- Proposição de métodos para a construção dos novos controladores;
- Construção de controladores a partir destes métodos;
- Inclusão, naqueles cenários, do robô virtual guiado por estes controladores; e
- Análise dos resultados das simulações realizadas.

1.3 Organização da Dissertação

No Capítulo 2 detalhamos os modelos e técnicas de Inteligência Computacional e Aprendizado de Máquina que foram utilizados neste trabalho.

O Capítulo 3 descreve o novo robô virtual, seus sensores, seus módulos e seu esquema de movimentação, bem como o ambiente virtual no qual o robô é inserido.

Já no Capítulo 4 são exibidos os detalhes do novo controlador construído manualmente, além de descrevermos o modo como foram utilizadas as técnicas para obtenção de outros controladores a partir deste e do antigo controlador.

O Capítulo 5 mostra os resultados obtidos a partir de simulações e uma análise destes, realizando as comparações entre os controladores.

Por fim, no Capítulo 6 realizamos uma breve análise do trabalho de uma forma geral, comentamos suas contribuições e deixamos indicados alguns possíveis trabalhos futuros.

2 INTELIGÊNCIA COMPUTACIONAL E APRENDIZADO DE MÁQUINA

Neste capítulo, abordamos os modelos e técnicas de Inteligência Computacional e Aprendizado de Máquina que foram utilizados neste trabalho, quais sejam: Lógica Nebulosa, Redes Neurais Artificiais e Máquinas de Vetores de Suporte.

2.1 Lógica Nebulosa

A imprecisão e a incerteza estão presentes sob vários aspectos no cotidiano do ser humano. Termos linguísticos aproximativos como "pouco", "muito", "perto" e "longe" são empregados durante a execução de diversas tarefas do dia-a-dia. Dessa forma, a lógica clássica, apesar de modelar satisfatoriamente boa parte das situações, por vezes deixa algumas lacunas por não permitir a utilização adequada daqueles termos.

A Teoria dos Conjuntos Nebulosos (ZADEH, 1965) tenta, justamente, preencher tais lacunas. Nesta teoria, os elementos passam a pertencer a um conjunto nebuloso com um certo grau de pertinência, o qual indica quão compatível com este conjunto aquele elemento é. São definidas, ainda, operações sobre tais conjuntos nebulosos,

como união, interseção e complemento (DUBOIS; WIDROW, 1980).

A Lógica Nebulosa consiste na aplicação da Teoria dos Conjuntos Nebulosos em um contexto lógico. Em especial, a Lógica Nebulosa pode ser utilizada em processos de inferência nebulosa, os quais são comumente encontrados em controladores para tomadas de decisão (KONG; KOSKO, 1992; MORATORI, 2006).

Nesta seção, descreveremos o funcionamento de sistemas dotados de processos de inferência nebulosa (chamados de sistemas nebulosos), bem como os detalhes relativos a Lógica Nebulosa utilizados naqueles.

2.1.1 Conjuntos e Operações

Na Lógica Clássica de Aristóteles, a pertinência de um elemento a um determinado conjunto é bem definida: ou o elemento pertence ao conjunto, ou ele não pertence. Assim, um conjunto clássico A pode ser definido por uma função de pertinência f_A aplicada a um universo de discurso X, de modo que, para cada elemento $x \in X$, $f_A(x) = 1$ se $x \in A$, e $f_A(x) = 0$ caso contrário.

Porém, quando passamos a trabalhar com imprecisão, torna-se mais complexa a definição da fronteira entre a pertinência ou não de um elemento a um conjunto. Por exemplo, a tarefa de definir a altura mínima para que alguém deixe de ser considerado "não alto" e passe a ser considerado "alto" pode ter severas implicações para um sistema que utilize esta informação para tomar decisões.

A Teoria dos Conjuntos Nebulosos tenta, então, estender a noção de pertinência além do modelo bivalente, de modo que um elemento passa a ter um grau de pertinência em relação a um conjunto. Com isso, podemos caracterizar um conjunto nebuloso

B por um função de pertinência μ_B, como fizemos com a função f_A em relação a A. Porém, para o conjunto nebuloso B, $\mu_B(x)$ (para $x \in X$) pode assumir valores no intervalo $[0, 1]$: $\mu_B(x) = 1$ quando x é totalmente compatível com o conjunto B; $\mu_B(x) = 0$ quando x é totalmente incompatível com B; e $0 < \mu_B(x) < 1$ quando x é parcialmente compatível com B. Neste último caso, $\mu_B(x)$ indica, ainda, o grau de compatibilidade de x em relação a B.

Nas figuras 2.1 e 2.2 podemos visualizar a diferença entre possíveis funções de pertinência para um conjunto clássico e um conjunto nebuloso, ambos representando o conceito "alto".

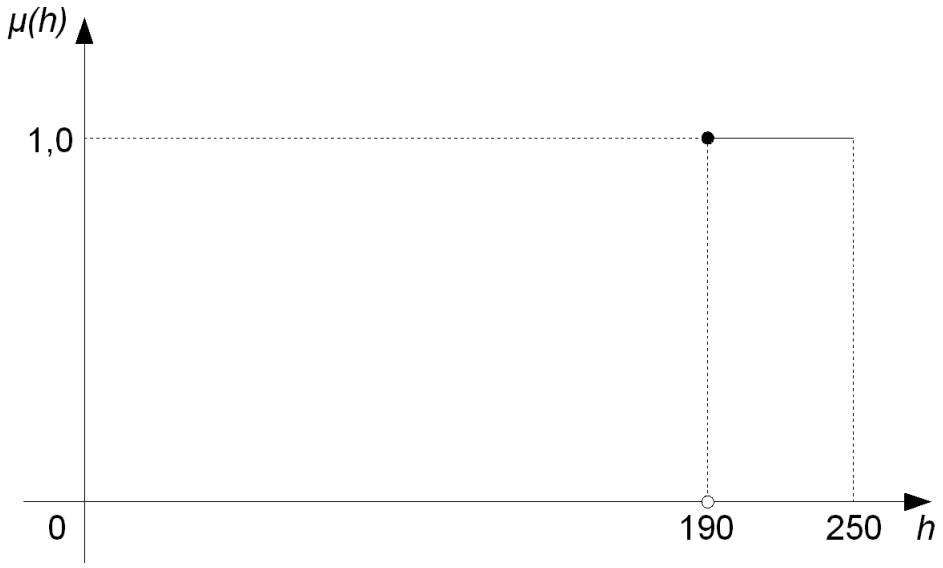


Figura 2.1: Função de pertinência para um conjunto clássico que representa o conceito "alto".

Assim como para os conjuntos clássicos, também dispomos de operações para trabalharmos com conjuntos nebulosos, tais como união, interseção e complemento. Tais operações são construídas utilizando-se as funções de pertinência dos conjuntos envolvidos, de modo a definir a função de pertinência do conjunto resultante da

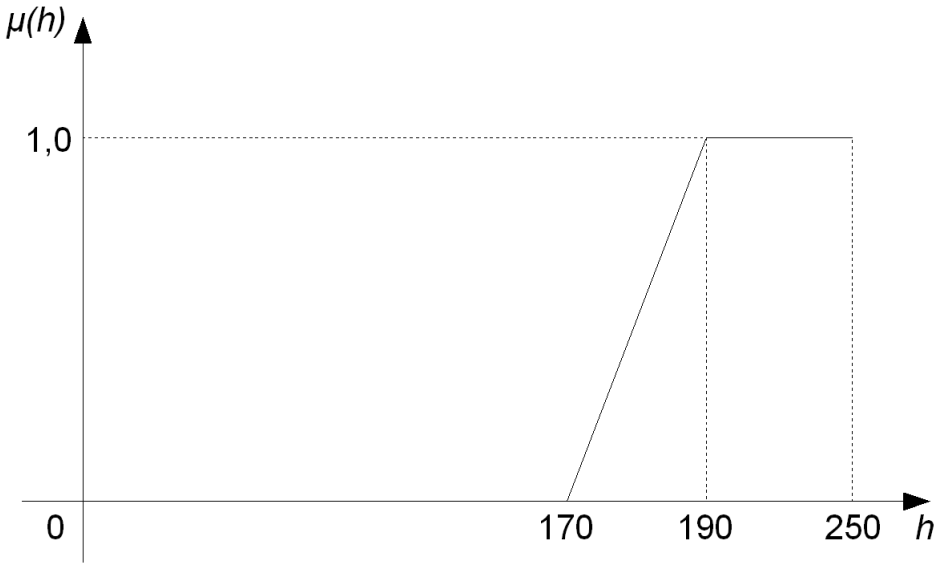


Figura 2.2: Função de pertinência para um conjunto nebuloso que representa o conceito "alto".

operação.

Desta forma, para a operação de união, a qual produz um conjunto $C = A \cup B$, costuma-se definir a função de pertinência μ_C tal que $\mu_C(x) = \max\{\mu_A(x), \mu_B(x)\}$, onde $x \in X$, sendo X o universo de discurso. Já para a interseção, $D = A \cap B$, μ_D é comumente definido como $\mu_D(x) = \min\{\mu_A(x), \mu_B(x)\}$. Por fim, para o complemento de um conjunto A, $E = \bar{A}$, temos $\mu_E(x) = 1 - \mu_A(x)$.

Há, ainda, diversas outras formas de representar as operações sobre conjuntos nebulosos, como pode ser visto em (DUBOIS; WIDROW, 1980).

2.1.2 Variáveis e Regras

Quando atacamos um problema através da abordagem de Lógica Nebulosa, utilizamos o que chamamos de variáveis nebulosas, através das quais representamos as entradas e as saídas do problema em questão. Cada variável nebulosa possui um universo de discurso, sobre o qual temos conjuntos nebulosos. Assim, quando a variável nebulosa assume um determinado valor, passam a estar definidos os graus de compatibilidade da variável em relação a cada conjunto.

A figura 2.3 exemplifica a divisão do universo de discurso da variável nebulosa h (altura), de domínio $[0, 250]$, em 3 conjuntos: "baixo", "médio" e "alto".

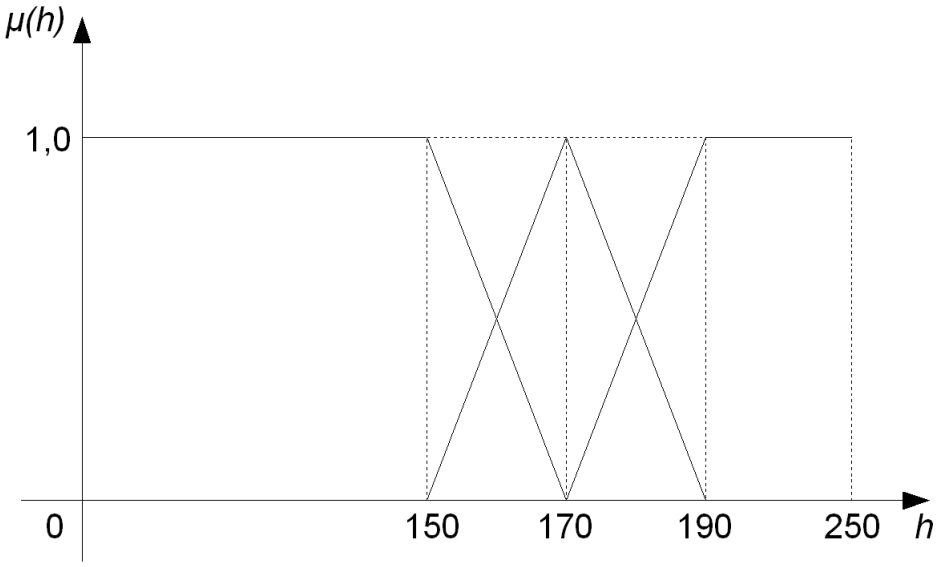


Figura 2.3: Divisão do universo de discurso da variável nebulosa h, que representa o conceito "altura", nos conjuntos "baixo", "médio" e "alto".

Uma vez que tenhamos definido as variáveis de nosso problema, podemos construir as regras que guiarão a lógica de um sistema nebuloso. Em seu formato, uma regra nebulosa é idêntica a uma regra de implicação da Lógica Clássica, constituída

por antecedentes e consequentes, os quais são compostos por cláusulas ligadas por conectivos como "e" e "ou". As cláusulas, por sua vez, possuem o formato "a é A", onde a é uma variável e A é um dos conjuntos existentes sobre o universo de discurso de A.

Os conectivos "e" e "ou" são avaliados da mesma forma que as operações sobre conjuntos nebulosos, representando, respectivamente, a interseção e a união entre conjuntos. Assim, podemos avaliar o grau de uma regra utilizando as mesmas funções comentadas na seção anterior.

Por exemplo, se para nosso problema temos as variáveis nebulosas a, b, c e d, podemos ter em nosso sistema uma regra do tipo "se (a é A) ou ((b é B) e (c é C)), então (d é D)." Neste caso, se $\mu_A(a) = 0{,}5$, $\mu_B(b) = 0{,}8$ e $\mu_C(c) = 0{,}4$, o grau da regra pode ser avaliado como se o antecedente estivesse no formato "$(a \in A) \cup ((b \in B) \cap (c \in C))$". Desta forma, temos que o grau da regra é $\max\{0{,}5; \min\{0{,}8; 0{,}4\}\} = 0{,}5$. É com este grau que, nesta situação, o consequente será levado em conta nas computações realizadas pelo sistema.

2.1.3 Sistemas Nebulosos

Chamamos de sistema nebuloso o sistema computacional que utiliza conjuntos, regras e operadores nebulosos para realizar inferências. Para isto, o sistema passa pelas seguintes etapas:

- Fuzzyficação: mapeamento dos valores numéricos das variáveis de entrada para graus de pertinência em conjuntos nebulosos;
- Processo de inferência a partir das regras nebulosas, analisando seus antecedentes, calculando seus graus e gerando um conjunto para cada variável de

saída; e

- Defuzzyficação: geração de valores numéricos para as variáveis de saída a partir dos conjuntos gerados.

Estas etapas podem ser vistas no esquema de controlador nebuloso da Figura 2.4.

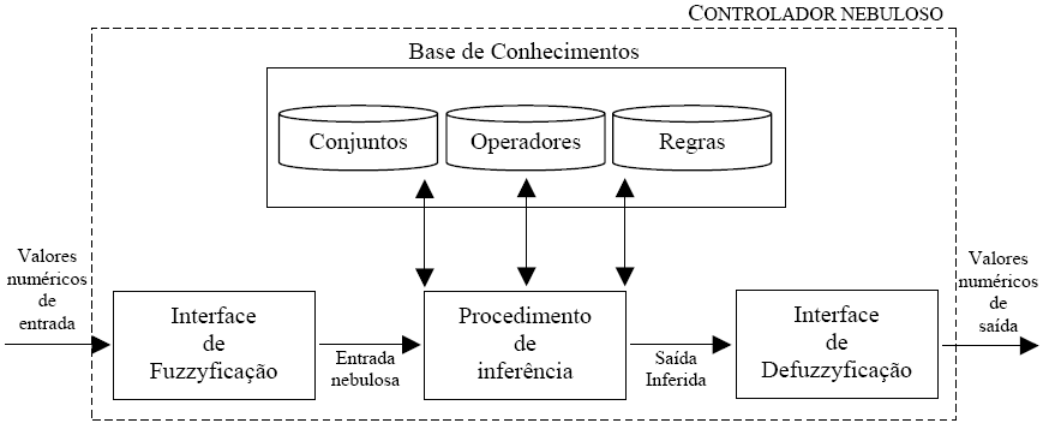


Figura 2.4: Esquema de controlador nebuloso, mostrando as etapas de fuzzyficação, inferência e defuzzyficação. Retirada de (MORATORI, 2006).

Na etapa de fuzzyficação, verificamos os graus de pertinência de cada variável (dado seu valor numérico) em relação a cada conjunto definido no universo de discurso daquela. Desta forma, obtemos as entradas nebulosas para nosso sistema.

Para realizar a inferência, primeiramente analisamos cada regra separadamente, procedendo da seguinte forma:

- Calculamos o grau da regra, utilizando os graus de pertinência das variáveis presentes nas cláusulas dos antecedentes e os operadores entre estas; e

- Para cada variável presente no consequente da regra, geramos um conjunto através do operador de interseção aplicado ao conjunto da cláusula daquela variável, utilizando o grau da regra para realizar um corte neste.

Tomemos como exemplo um sistema com duas variáveis de entrada (a e b) e uma variável de saída (v), e cuja base de conhecimentos contém, dentre outras, a regra "se (a é A_1) e (b é B_1) então (v é V_1)". O grau da regra, g, é calculado a partir dos graus de pertinência $\mu_{A_1}(a)$ (de a em A_1) e $\mu_{B_1}(b)$ (de b em B_1), utilizando-se o operador "min" (já que o operador lógico da regra trata-se de um "e"). Assim, $g = \min\{\mu_{A_1}(a), \mu_{B_1}(b)\}$.

Em seguida, aplicamos o operador de interseção ao conjunto V_1 do consequente da regra, utilizando g como valor de corte. Neste sentido, o novo conjunto V_1', que será utilizado no cálculo da saída, tem função de pertinência $\mu_{V_1'}(v) = \min\{\mu_{V_1}(v), g\}$.

A Figura 2.5 ilustra os procedimentos descritos acima. Nela, podemos ver o conjunto V_1, de formato triangular, e o conjunto V_1', resultado do corte do primeiro no valor g.

Uma vez tendo processado cada regra, aplicamos, para cada variável de saída, o operador de união sobre os conjuntos gerados no procedimento acima. Os conjuntos obtidos a partir destas uniões são as saídas nebulosas de nosso sistema. A Figura 2.6 mostra um exemplo de saída nebulosa, construída a partir da união dos conjuntos cortados pelos graus das regras do sistema, dentre eles o conjunto V_1' da Figura 2.5.

Por fim, na etapa de defuzzyficação utilizamos algum tipo de procedimento no qual, dado um conjunto resultante do processo de inferência, tentamos extrair um valor numérico que represente a idéia daquele conjunto. Comumente, utilizamos o método do centro de gravidade (CRUZ, 2004), no qual, como sugerido pelo nome, calculamos

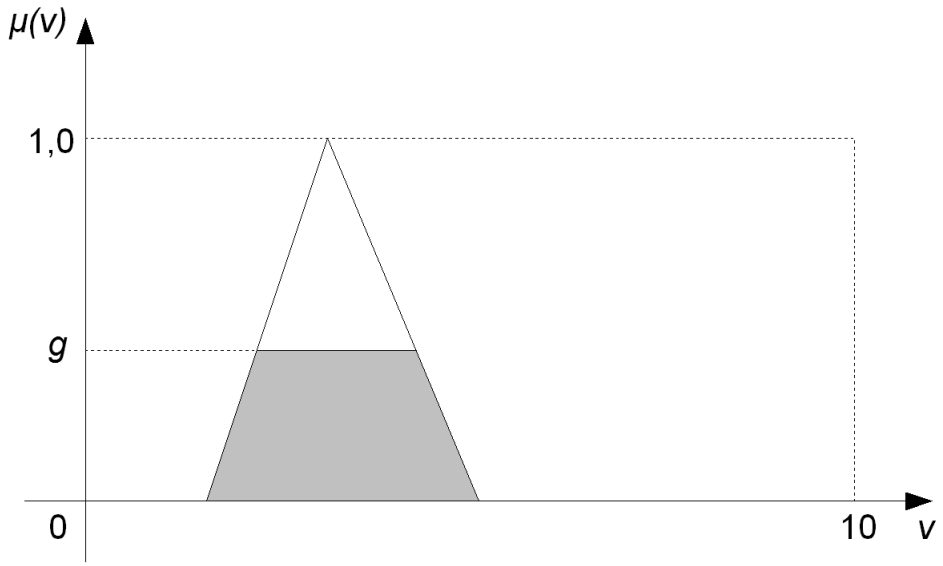


Figura 2.5: Exemplo de corte no processo de inferência, gerando um dos conjuntos nebulosos que serão utilizados para construir a saída nebulosa.

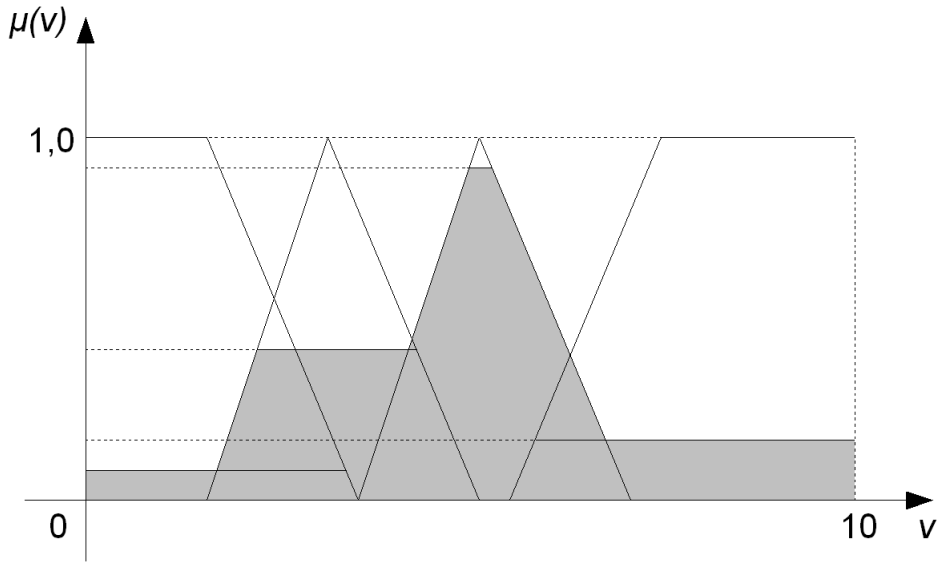


Figura 2.6: Exemplo de saída nebulosa, construída a partir da união dos conjuntos obtidos do processo de inferência.

o centro de gravidade da área do conjunto, obtendo o valor que divide esta em duas regiões de mesma área. Há, ainda, outros métodos de defuzzyficação, como os encontrados em (SANDRI; CORREA, 1999; DRIANKOV; HELLENDOORN; RENFRANK, 1993).

Na Figura 2.7 podemos ver a aplicação do método de centro de gravidade ao exemplo anterior. O ponto marcado corresponde ao centro de massa da saída nebulosa, cujo valor de v correspondente é o que seria dado como saída numérica pelo sistema.

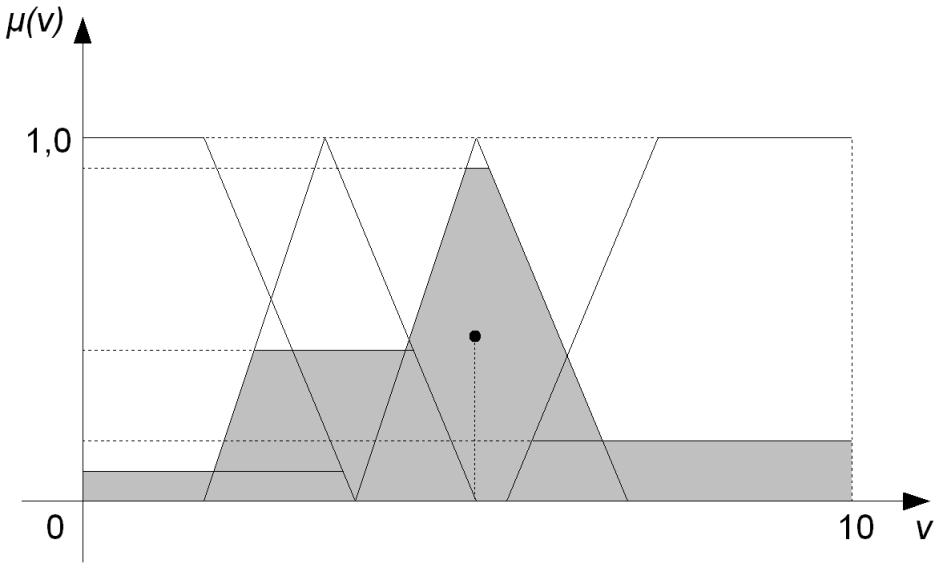


Figura 2.7: Defuzzyficação a partir da aplicação do método de centro de gravidade, o qual corresponde ao ponto marcado na figura.

2.1.4 Aprendizado da Base de Conhecimento

Quando construímos um sistema nebuloso, toda a base de conhecimento (conjuntos, operadores e regras) é normalmente criada por um especialista da tarefa na qual deseja-se empregar tal sistema. Porém, por vezes não temos à disposição um

operador humano que seja capaz de criar manualmente esta base de conhecimento. Nestes casos, temos a possibilidade de utilizar métodos que automatizem o processo de criação, a partir de informações sobre como desejamos que seja o comportamento do sistema nebuloso.

Para o modelo clássico apresentado anteriormente, denominado Mamdani (MAMDANI, 1974), costuma-se empregar o método de Wang (WANG; MENDEL, 1992). Porém, existem outros modelos, como o de Takagi-Sugeno (TAKAGI; SUGENO, 1985), que adequam-se melhor a essa situação.

O modelo de Takagi-Sugeno é semelhante ao de Mamdani, porém os consequentes representam funções das entradas no lugar de cláusulas unidas por operadores. Estas funções são normalmente combinações lineares das entradas, tendo como parâmetros constantes previamente definidas. A saída do sistema, por sua vez, é a média dos resultados das funções de cada regra, ponderados pelos graus calculados para cada regra. Além disso, no esquema de Takagi-Sugeno costuma-se utilizar funções mais suaves para representar os conjuntos das variáveis de entrada, como a gaussiana, ao contrário das funções triangulares e trapezoidais comumente encontradas nos sistemas do tipo Mamdani.

Um dos métodos mais utilizados para a geração de um sistema nebuloso Takagi-Sugeno é o que faz uso da representação do sistema através de ANFIS (do inglês *Adaptive Neuro-Fuzzy Inference System*, Sistema de Inferência Neuro-Fuzzy Adaptativa). O ANFIS é uma forma de representar um sistema nebuloso que utiliza interpolação, como o de Takagi-Sugeno, como uma rede neural artificial.

Como consequência, podemos fazer uso de um algoritmo de treinamento que tenta adaptar os parâmetros do sistema, baseando-se nos erros gerados pelo sistema para um conjunto de dados de amostra e respectivas respostas esperadas. O algoritmo

de treinamento mais comum (JANG, 1993) utiliza o método de Mínimos Quadrados em conjunto com o método de Gradiente Decrescente para, respectivamente, definir os parâmetros dos consequentes e dos antecedentes das regras.

Como este procedimento é um tanto custoso e pode não trazer os resultados esperados, neste trabalho optamos por não utilizá-lo. Neste caso, uma vez que os parâmetros dos antecedentes já estão corretamente definidos (por já termos disponível um controlador Mamdani), podemos utilizar apenas o método do Gradiente Decrescente para adaptação dos parâmetros das funções dos consequentes. Para isso, os cálculos são descritos a seguir.

Sejam

- n: número de entradas do sistema;
- s: número de saídas do sistema;
- x_i: entrada i do sistema ($i = 1, \ldots, n$);
- m_i: número de conjuntos usados na divisão do domínio da entrada i ($i = 1, \ldots, n$);
- μ_{ij}: função de pertinência no j-ésimo conjunto ($j = 1, \ldots, m_i$) da entrada i ($i = 1, \ldots, n$);
- a_{ij}: número de parâmetros da função de pertinência no j-ésimo conjunto ($j = 1, \ldots, m_i$) da entrada i ($i = 1, \ldots, n$);
- p_{ijk}: k-ésimo parâmetro ($k = 1, \ldots, a_{ij}$) da função de pertinência do j-ésimo conjunto ($j = 1, \ldots, m_i$) da entrada i ($i = 1, \ldots, n$);
- r: número de regras do sistema;

- f_{ij}: função de ativação do ij-ésimo consequente do sistema ($i = 1, \ldots, r$, $j = 1, \ldots, s$);

- b_{ij}: número de parâmetros do ij-ésimo consequente do sistema ($i = 1, \ldots, r$, $j = 1, \ldots, s$);

- q_{ijk}: k-ésimo parâmetro ($k = 1, \ldots, b_{ij}$) do ij-ésimo consequente do sistema ($i = 1, \ldots, r$, $j = 1, \ldots, s$);

- y_{1ij}: saída do neurônio ij da camada 1, relativa ao grau de pertinência da entrada i ($i = 1, \ldots n$) ao j-ésimo conjunto desta entrada ($j = 1, \ldots, m_i$);

- y_{2i}: saída do neurônio i da camada 2, relativa ao grau de ativação da i-ésima regra ($i = 1, \ldots, r$);

- w_{ijk}: peso da ligação entre o neurônio i ($i = 1, \ldots, r$) da camada 2 e o neurônio jk ($j = 1, \ldots, n$, $k = 1, \ldots, m_j$) da camada 1, definindo a i-ésima regra do sistema;

- y_{30}: saída do neurônio 0 da camada 3, relativa à soma dos graus de ativação das regras do sistema;

- y_{3ij}: saída do neurônio ij da camada 3, relativa ao consequente ij do sistema ($i = 1, \ldots, s$, $j = 1, \ldots, r$);

- y_{4i}: saída do neurônio i da camada 4, relativa à i-ésima saída do sistema, combinação normalizada dos consequentes ($i = 1, \ldots, s$);

- h_{ij}: erro absoluto do neurônio j da camada i;

- e_{ij}: erro ajustado do neurônio j da camada i; e

- E: performance do sistema para cada amostra de treinamento,

com as seguintes restrições:

$$\begin{aligned} w_{ijk} &\in \{0,1\} \\ \sum_{k=1}^{m_j} w_{ijk} &\in \{0,1\} \\ \sum_{j=1}^{n}\sum_{k=1}^{m_j} w_{ijk} &= 1 \end{aligned}$$

Calculamos os antecedentes e os consequentes do sistema através das seguintes fórmulas:

$$\begin{aligned} net_{1ij} &= \mu_{ij}\left(x_i; p_{ij1}, \ldots, p_{ija_{ij}}\right) \\ y_{1ij} &= net_{1ij} \\ net_{2ij} &= \sum_{k=1}^{m_j} w_{ijk} y_{1jk} \\ y_{2i} &= \prod_{j=1}^{n} net_{2ij} \\ net_{30} &= \sum_{i=1}^{r} y_{2i} \\ y_{30} &= net_{30} \\ net_{3ij1} &= f_{ij}\left(x_1, \ldots, x_n; q_{ij1}, \ldots, q_{ijb_{ij}}\right) \\ net_{3ij2} &= y_{2j} \\ y_{3ij} &= net_{3ij1} \cdot net_{3ij2} \\ net_{4i1} &= \sum_{j=1}^{r} y_{3ij} \\ net_{4i2} &= y_{30} \\ y_{4i} &= \frac{net_{4i1}}{net_{4i2}} \end{aligned}$$

Com isso, as saídas do sistema correspondem aos y_{4i} $(i = 1, \ldots, s)$.

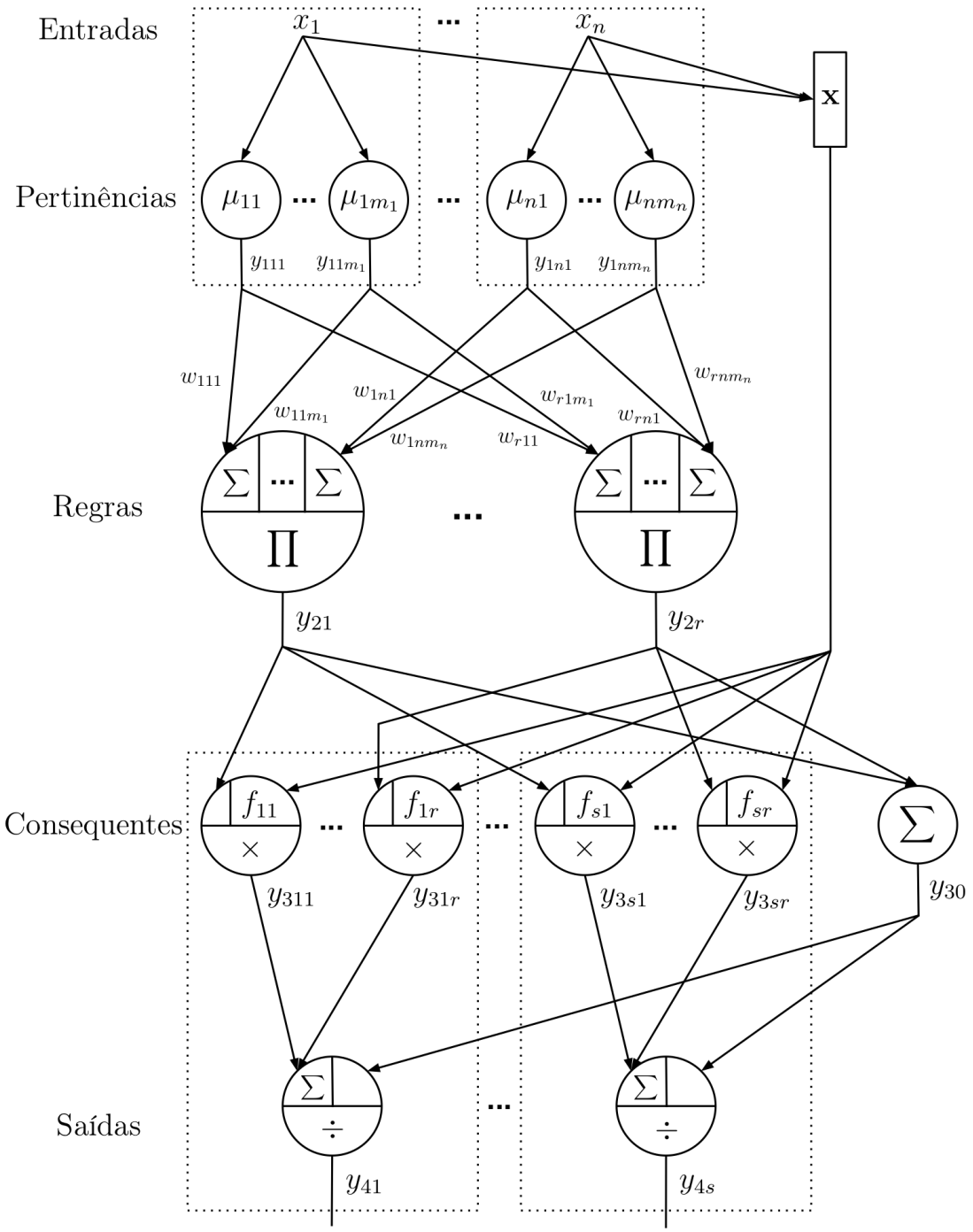


Figura 2.8: Esquema de sistema Takagi-Sugeno para adaptação de parâmetros utilizando-se Gradiente Decrescente.

A Figura 2.8 exibe um esquema dos cálculos descritos acima.

Os parâmetros p_{ijk} e w_{ijk} são facilmente definidos copiando-se as informações já conhecidas do sistema do tipo Mamdani.

Já para adaptarmos os parâmetros q_{ijk} do sistema podemos fazer uso de Gradiente Decrescente, ou seja, realizamos as atualizações na direção oposta à apontada pelo gradiente do erro em relação a cada parâmetro,

$$\nabla E = \left(\frac{\partial E}{\partial q_{111}}, \frac{\partial E}{\partial q_{112}}, \ldots, \frac{\partial E}{\partial q_{11b_{11}}}, \frac{\partial E}{\partial q_{121}}, \ldots, \frac{\partial E}{\partial q_{rsb_{rs}}} \right)$$

de modo a seguirmos a direção de maior decréscimo na função de erro. Assim, para um certo parâmetro q_{ijk}, temos

$$\Delta q_{ijk} \propto -\frac{\partial E}{\partial q_{ijk}}.$$

Realizamos, então, o cálculo na ordem inversa das camadas, isto é,

- Camada 4 (definição dos erros):

$$\begin{aligned} E &= \frac{1}{2} \sum_{i=1}^{s} h_{4i}^2 \\ h_{4i} &= d_i - y_{4i} \\ e_{4i1} &= h_{4i} \cdot \frac{1}{net_{4i2}} \\ e_{4i2} &= h_{4i} \cdot \left(-\frac{net_{4i1}}{net_{4i2}^2} \right) \end{aligned}$$

- Camada 3:

$$
\begin{aligned}
\frac{\partial E}{\partial q_{ijk}} &= \frac{\partial E}{\partial h_{4i}}\frac{\partial h_{4i}}{\partial q_{ijk}} \\
&= \frac{\partial E}{\partial h_{4i}}\frac{\partial h_{4i}}{\partial y_{4i}}\frac{\partial y_{4i}}{\partial q_{ijk}} \\
&= \frac{\partial E}{\partial h_{4i}}\frac{\partial h_{4i}}{\partial y_{4i}}\frac{\partial y_{4i}}{\partial net_{4i1}}\frac{\partial net_{4i1}}{\partial q_{ijk}} \\
&= \frac{\partial E}{\partial h_{4i}}\frac{\partial h_{4i}}{\partial y_{4i}}\frac{\partial y_{4i}}{\partial net_{4i1}}\frac{\partial net_{4i1}}{\partial y_{3ij}}\frac{\partial y_{3ij}}{\partial q_{ijk}} \\
&= \frac{\partial E}{\partial h_{4i}}\frac{\partial h_{4i}}{\partial y_{4i}}\frac{\partial y_{4i}}{\partial net_{4i1}}\frac{\partial net_{4i1}}{\partial y_{3ij}}\frac{\partial y_{3ij}}{\partial net_{3ij1}}\frac{\partial net_{3ij1}}{\partial q_{ijk}} \\
\frac{\partial E}{\partial q_{ijk}} &= (h_{4i})(-1)\left(\frac{1}{net_{4i2}}\right)(1)(net_{3ij2})\left(\frac{\partial f_{ij}}{\partial q_{ijk}}\right) \\
&= -\left(h_{4i}\cdot\frac{1}{net_{4i2}}\right)\cdot net_{3ij2}\cdot\frac{\partial f_{ij}}{\partial q_{ijk}} \\
&= -e_{4i1}\cdot net_{3ij2}\cdot\frac{\partial f_{ij}}{\partial q_{ijk}} \\
h_{3ij} &= e_{4i1} \\
\frac{\partial E}{\partial q_{ijk}} &= -h_{3ij}\cdot net_{3ij2}\cdot\frac{\partial f_{ij}}{\partial q_{ijk}} \\
e_{3ij1} &= h_{3ij}\cdot net_{3ij2} \\
\frac{\partial E}{\partial q_{ijk}} &= -e_{3ij1}\cdot\frac{\partial f_{ij}}{\partial q_{ijk}}
\end{aligned}
$$

Assim,

$$\Delta q_{ijk} = \eta\cdot e_{3ij1}\cdot\frac{\partial f_{ij}}{\partial q_{ijk}},$$

onde η é uma constante denominada taxa de aprendizado.

É interessante notar que o uso de Gradiente Decrescente nesta situação é uma proposta deste trabalho.

2.2 Redes Neurais Artificiais

Redes Neurais Artificiais (RNAs) são modelos computacionais que tentam, de alguma forma, imitar a estrutura e o funcionamento do cérebro humano, com o objetivo de obter as capacidades de aprendizado e generalização que este possui. Tais capacidades são interessantes na solução de problemas complexos, como os de classificação, categorização e previsão, para os quais normalmente não existem soluções por métodos algorítmicos convencionais (GUINGO; RODRIGUES; THOMé, 2002; SILVA, 2002).

Uma RNA é composta por unidades simples de processamento paralelo, chamadas de nodos ou neurônios, que emulam o funcionamento de neurônios biológicos. Estas unidades são usualmente dispostas em uma ou mais camadas e estão interligadas através de conexões. Tais conexões normalmente possuem pesos associados, os quais são os principais responsáveis por armazenar o conhecimento da rede.

Para utilizarmos uma rede neural na solução de um problema, devemos, então, definir os pesos da rede de modo que esta atue satisfatoriamente. O processo normalmente utilizado para a definição dos pesos é denominado aprendizado ou treinamento, pois consiste em apresentar à rede um conjunto de exemplos históricos para que ela aprenda a partir destes exemplos. Para cada exemplo apresentado, a rede gera uma saída, a qual é analisada segundo algum critério, de modo a produzir um retorno em relação à saída gerada. Os retornos das apresentações dos exemplos são, então, utilizados para adaptar os pesos da rede.

Nas próximas seções trataremos, inicialmente, de alguns conceitos relativos aos neurônios biológicos, para, posteriormente, introduzirmos alguns modelos de redes neurais artificiais.

2.2.1 Neurônio Biológico

O neurônio biológico, como pode ser visto na Figura 2.9, é composto basicamente por três partes: corpo celular, axônio e dendritos. Os dendritos são responsáveis por receber os impulsos nervosos de outros neurônios e repassá-los ao corpo celular, onde os impulsos são processados. O processamento dos impulsos recebidos gera novos impulsos, os quais são transmitidos através do axônio e chegam a outros neurônios pelo contato entre esse axônio e os dendritos desses outros neurônios. Este contato é denominado sinapse. As sinapses podem facilitar ou dificultar a transmissão do impulso nervoso, de modo que o impulso gerado pelo corpo celular depende de quão fortes são os impulsos recebidos após estes passarem pelas sinapses.

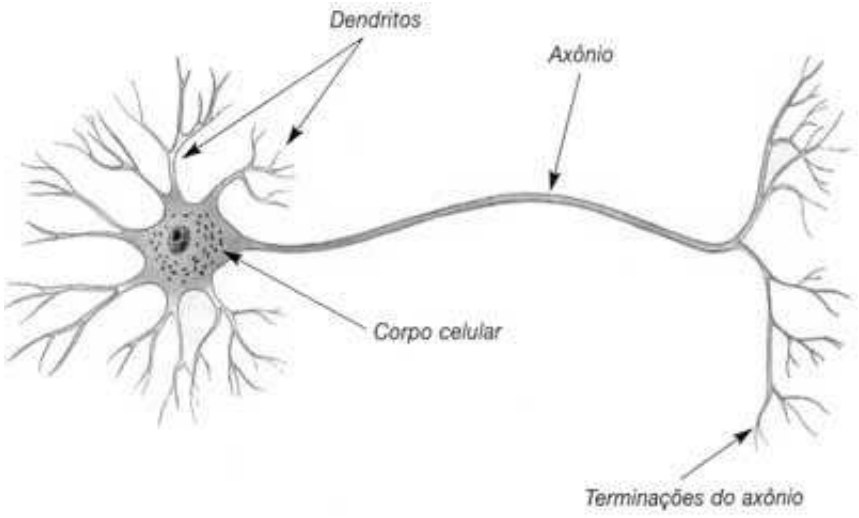


Figura 2.9: Neurônio biológico e seus componentes.

2.2.2 Modelo MCP

Em 1943, Warren McCulloch e Walter Pitts criaram o primeiro modelo artificial para um neurônio biológico (MCCULLOCH; PITTS, 1943). O modelo MCP (de McCulloch-Pitts), como ficou conhecido, representava matematicamente as principais estruturas do neurônio, bem como seu funcionamento.

No modelo MCP, os dendritos são representados por terminais de entrada, responsáveis por receber as entradas, as quais simbolizam os impulsos provenientes de outros neurônios. A cada terminal de entrada está ligado um peso, representando a força da sinapse do dendrito correspondente ao terminal. O processamento realizado pelo corpo celular é representado por uma função das entradas e dos pesos, produzindo a saída, a qual simboliza o impulso gerado pelo neurônio. O axônio é representado por um terminal de saída, responsável por distribuir a saída gerada.

Formalmente, sejam $x_1, \ldots, x_n$ as n entradas recebidas por um neurônio MCP através de seus terminais de entrada, os quais possuem pesos $w_1, \ldots, w_n$. O neurônio terá sua saída ativada apenas quando

$$\sum_{i=1}^{n} w_i x_i \geq \theta, \tag{2.1}$$

ou seja, apenas quando o somatório das entradas ponderadas por seus pesos correspondentes for maior ou igual a um determinado limiar θ, chamado *threshold*. Esta situação pode ser visualizada através da Figura 2.10.

Sejam os vetores $\mathbf{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$ e $\mathbf{w} = (w_1, \ldots, w_n)^{\mathrm{T}}$ e o escalar $b = -\theta$ (denominado *bias*). Podemos, então, considerar a saída y gerada por um neurônio MCP de n entradas como sendo o resultado da aplicação de uma função f, denominada função de propagação, sobre o resultado de uma função *net*, denominada função de

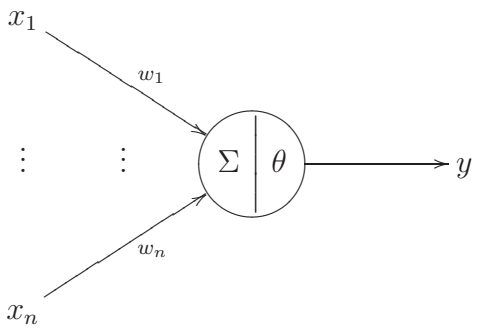


Figura 2.10: Modelo MCP.

ativação, tal que

$$y = f\left(net\right). \tag{2.2}$$

Para isso, devemos ter

$$net = b + \mathbf{w}^{\mathrm{T}}\mathbf{x} \tag{2.3}$$

e

$$f\left(z\right) = \begin{cases} 1 & \text{se } z \geq 0 \\ 0 & \text{se } z < 0 \end{cases}. \tag{2.4}$$

É importante ressaltar que o modelo MCP original foi proposto com pesos não-ajustáveis.

2.2.3 Redes Perceptron

Em 1958, Frank Rosenblatt propôs o modelo conhecido como Perceptron (ROSENBLATT, 1958), o qual consiste em uma rede composta de vários neurônios MCP. Com o modelo, foi proposta, também, uma regra de aprendizado, através da qual os pesos da rede são ajustados.

A topologia do modelo perceptron original consistia em três níveis. O primeiro nível continha as unidades sensoras, responsáveis, apenas, por obter as entradas para o perceptron, de modo que não possuíam pesos. O segundo nível era composto por unidades de associação, cada uma sendo um neurônio MCP sem pesos ajustáveis. O terceiro nível era formado por unidades de resposta, as quais também eram neurônios MCP. Os neurônios deste último nível, porém, possuíam pesos ajustáveis. Dessa forma, o modelo original ficou conhecido como Perceptron de uma única camada, pois apenas o último nível possuía propriedades adaptativas (Figura 2.11).

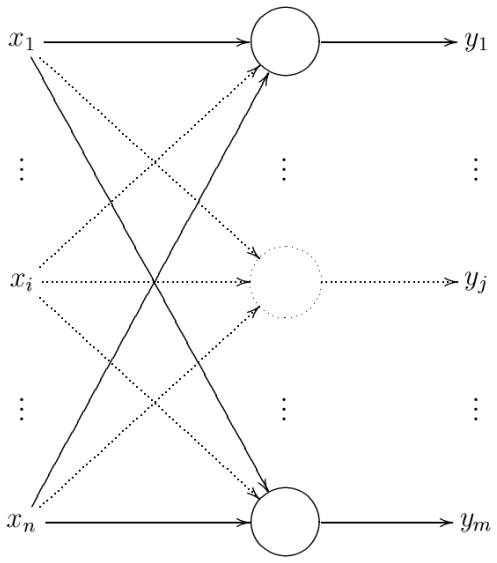


Figura 2.11: Rede Perceptron.

Cada um dos neurônios da única camada do Perceptron gera saídas que independem dos outros neurônios desta camada. Assim, podemos analisar separadamente o processo de aprendizado de cada neurônio.

Para um determinado neurônio com n terminais de entrada, sejam $\mathbf{x}$ um vetor de entradas, $\mathbf{w}^{(k)}$ o vetor de pesos do neurônio na k-ésima iteração do treinamento (com

$k = 0, 1, \ldots$) e $b^{(k)}$ o *bias* do neurônio. Definimos, normalmente de forma aleatória, $\mathbf{w}^{(0)}$ e $b^{(0)}$. Devemos, então, calcular $\Delta\mathbf{w}^{(k)}$ e $\Delta b^{(k)}$, de modo que

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \Delta\mathbf{w}^{(k)} \tag{2.5}$$

$$b^{(k+1)} = b^{(k)} + \Delta b^{(k)}, \tag{2.6}$$

isto é, devemos calcular o incremento no vetor de pesos e no *bias* do neurônio na k-ésima iteração do treinamento, com vistas ao aprendizado da entrada $\mathbf{x}$.

Seja $e^{(k)}$ o erro do neurônio na k-ésima iteração do treinamento para a entrada $\mathbf{x}$, dado por

$$e^{(k)} = d - y^{(k)}, \tag{2.7}$$

onde $y^{(k)}$ é a saída gerada pelo neurônio para o vetor de entradas $\mathbf{x}$, o vetor de pesos $\mathbf{w}^{(k)}$ e o *bias* $b^{(k)}$, através da equação (2.2), e d é a saída desejada para tal entrada. Então, como visto em (BRAGA; CARVALHO; LUDERMIR, 2000), devemos ter

$$\Delta\mathbf{w}^{(k)} = \eta e^{(k)}\mathbf{x} \tag{2.8}$$

$$\Delta b^{(k)} = \eta e^{(k)}, \tag{2.9}$$

onde η é uma constante denominada taxa de aprendizado.

De forma mais genérica, sejam n o número de terminais de entrada, m o número de terminais de saída, p o número de exemplos, $\mathbf{X}_{n\times p}$ as entradas dos exemplos, $\mathbf{W}^{(k)}_{m\times n}$ os pesos na k-ésima iteração do treinamento, $\mathbf{b}^{(k)}_{m\times 1}$ os *bias*, $\mathbf{Y}^{(k)}_{m\times p}$ as saídas geradas para as entradas em $\mathbf{X}$, $\mathbf{D}_{m\times p}$ as saídas esperadas para as entradas em $\mathbf{X}$, $\mathbf{E}^{(k)}_{m\times p}$ os erros para cada uma das saídas, $\Delta\mathbf{W}^{(k)}_{m\times n}$ os incrementos dos pesos, $\Delta\mathbf{b}^{(k)}_{m\times 1}$ os incrementos dos *bias* e $\mathbf{1}_{1\times p}$ um vetor no qual todas as componentes têm valor 1.

Temos, então,

$$\mathbf{net}^{(k)} = \mathbf{b}^{(k)}\mathbf{1} + \mathbf{W}^{(k)}\mathbf{X} \tag{2.10}$$

$$\mathbf{Y}^{(k)} = \mathbf{F}\left(\mathbf{net}^{(k)}\right) \tag{2.11}$$

$$\mathbf{E}^{(k)} = \mathbf{D} - \mathbf{Y}^{(k)} \tag{2.12}$$

$$\Delta\mathbf{W}^{(k)} = \eta\mathbf{E}^{(k)}\mathbf{X}^{\mathrm{T}} \tag{2.13}$$

$$\Delta\mathbf{b}^{(k)} = \eta\mathbf{E}^{(k)}\mathbf{1}^{\mathrm{T}}, \tag{2.14}$$

onde

$$\left(\mathbf{F}\left(\mathbf{Z}\right)\right)_{ij} = f\left(\mathbf{Z}_{ij}\right) \tag{2.15}$$

e f é a mesma função da equação (2.4).

Podemos utilizar o Algoritmo 2.1 para realizar o treinamento de uma rede perceptron, o qual, pelo teorema da convergência (BRAGA; CARVALHO; LUDERMIR, 2000), chega a uma solução em um número finito de passos caso os exemplos formem classes linearmente separáveis (isto é, possam ser separados por hiperplanos no hiperespaço das amostras).

Algoritmo 2.1 Treinamento de redes Perceptron.

inicie $\mathbf{W}$
inicia $\mathbf{b}$
repita
  calcule **net** a partir da equação (2.10)
  calcule $\mathbf{Y}$ a partir das equações (2.15) e (2.11)
  calcule $\mathbf{E}$ a partir da equação (2.12)
  se $\mathbf{E} \neq \mathbf{0}_{m\times p}$ **então**
    calcule $\Delta\mathbf{W}$ a partir da equação (2.13)
    calcule $\Delta\mathbf{b}$ a partir da equação (2.14)
    $\mathbf{W} \leftarrow \mathbf{W} + \Delta\mathbf{W}$
    $\mathbf{b} \leftarrow \mathbf{b} + \Delta\mathbf{b}$
  fim de se
até que $\mathbf{E} = \mathbf{0}_{m\times p}$

2.2.4 Redes Adaline

Proposto por Widrow e Hoff em 1960, o Adaline (WIDROW; HOFF, 1960) possui várias semelhanças em relação ao Perceptron, apesar de terem surgido a partir de áreas distintas e com enfoques distintos. Assim como no Perceptron, o Adaline possui apenas uma camada de neurônios e tem como função de ativação o somatório das entradas ponderadas pelos pesos correspondentes. Por outro lado, sua função de propagação gera saídas binárias em -1 e $+1$.

O grande diferencial do Adaline, porém, está na dedução da fórmula de adaptação dos pesos. Ao contrário do perceptron, no Adaline o erro para um exemplo $\mathbf{x}$ é calculado a partir da função de ativação, isto é,

$$e^{(k)} = d - net^{(k)}. \tag{2.16}$$

Com isso, através da equação

$$G^{(k)} = \frac{\left(e^{(k)}\right)^2}{2}, \tag{2.17}$$

os pesos de um neurônio geram uma superfície de erro. A idéia é caminhar nesta superfície em direção ao ponto de mínimo, utilizando, para isto, a direção oposta à indicada pelo gradiente, ou seja, devemos ter

$$\Delta\mathbf{w}^{(k)} \propto -\nabla G^{(k)}. \tag{2.18}$$

Assim, como descrito em (BRAGA; CARVALHO; LUDERMIR, 2000), obtemos

$$\nabla G^{(k)} = -e^{(k)}\mathbf{x}, \tag{2.19}$$

de modo que, pelas equações (2.18) e (2.19), temos

$$\Delta\mathbf{w}^{(k)} = \eta e^{(k)}\mathbf{x}, \tag{2.20}$$

que é semelhante à equação (2.8). A equação (2.20) é comumente chamada de *regra delta*.

O treinamento de um Adaline pode, então, ser feito através de um algoritmo semelhante ao utilizado para o treinamento do Perceptron, trocando-se o cálculo do erro para que seja utilizada a equação (2.16).

2.2.5 Redes MLP

Tanto o Perceptron quanto o Adaline, descritos nas seções anteriores, possuem algumas limitações. A principal dessas limitações é o fato daqueles modelos resolverem apenas problemas linearmente separáveis, o que restringe enormemente os problemas que podem ser tratados utilizando-se tais modelos.

Isto é devido à utilização de funções de limiar ou lineares, as quais geram superfícies de separação lineares. Além disso, para que problemas mais complexos possam ser resolvidos, faz-se necessário o uso de mais camadas além das camadas de entrada e de saída (CYBENKO, 1989).

Temos, então, as Redes Perceptron Multicamadas, ou MLP (de *Multi-Layer Perceptron*), as quais são compostas por mais de uma camada de perceptrons com pesos ajustáveis e, normalmente, com pelo menos uma camada de perceptrons com função de propagação não-linear. As camadas existentes entre os terminais de entrada e a camada de saída são denominadas camadas intermediárias ou camadas ocultas. Nas Redes MLP, as saídas geradas pelos neurônios de uma camada oculta são utilizadas como entradas para os neurônios da camada seguinte (Figura 2.12).

Os algoritmos de treinamento das seções anteriores utilizam o erro em relação a saí-

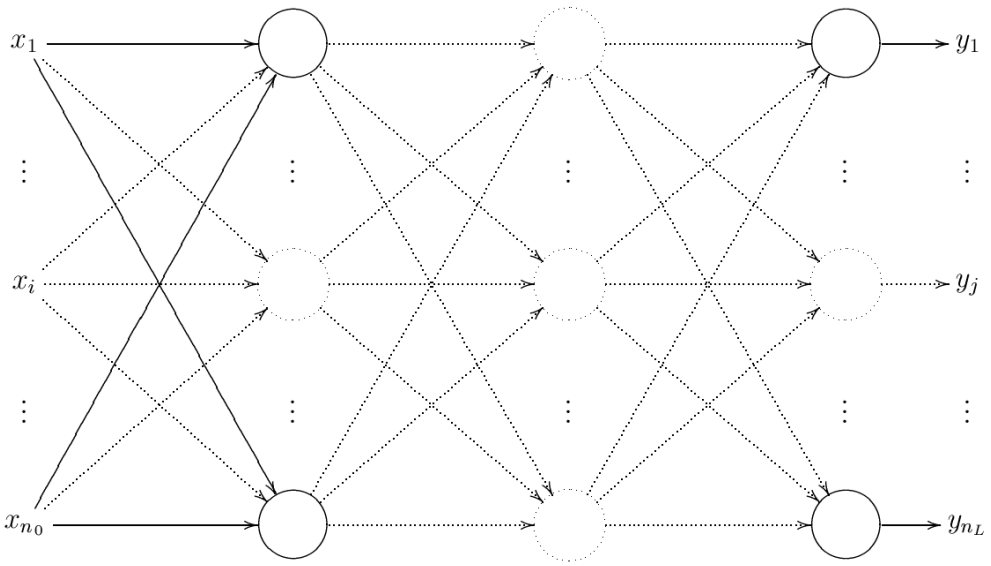


Figura 2.12: Rede MLP.

das esperadas para a atualização dos pesos. O uso de redes com mais de uma camada esbarra, então, na dificuldade em calcular o erro para as camadas intermediárias, já que estas, a princípio, não possuem saídas esperadas definidas.

As limitações das redes de uma única camada, as dificuldades no treinamento das redes de múltiplas camadas e a publicação de Minsky e Papert, em 1969, mostrando uma visão pessimista em relação às RNAs (MINSKY; PAPERT, 1969), fizeram com que a área ficasse estagnada por mais de uma década.

Em 1986, porém, Rumelhart propôs um algoritmo para treinamento de Redes MLP, denominado *Back-propagation* (RUMELHART; HINTON; WILLIAMS, 1986), o qual foi um dos responsáveis pela retomada das pesquisas na área de RNAs.

2.2.5.1 Algoritmo Back-propagation

Cada ciclo do algoritmo *Back-propagation* é composto por duas fases: a fase *forward* e a fase *backward*. A fase *forward* consiste em apresentar à rede as entradas dos exemplos para gerar as saídas de cada neurônio de cada camada. Para isto, percorre-se a rede da camada de entrada para a camada de saída. A fase *backward* consiste em atualizar os pesos dos neurônios de cada camada, o que é feito da camada de saída em direção à camada de entrada. Para isto, utiliza-se o erro em relação às saídas desejadas ou, no caso das camadas intermediárias, a estimativa do erro das saídas geradas, a qual é calculada em função do erro da camada seguinte.

A idéia do *Back-propagation* é a mesma do treinamento do Adaline: utilizar uma função que gera uma superfície de erro e percorrer tal superfície em busca do ponto de mínimo. Utiliza-se, então, o erro médio quadrático

$$e^{(k)} = \sum_{i=1}^{p} \sum_{j=1}^{n_L} \left(\mathbf{T}_{ji} - \left(\mathbf{Y}_L^{(k)} \right)_{ji} \right)^2, \tag{2.21}$$

ou seja, o somatório dos quadrados das diferenças entre as saídas esperadas ($\mathbf{T}$) e as saídas geradas pela rede ($\mathbf{Y}$). Por fim, como no Adaline, tenta-se minimizar a função da equação (2.17).

Porém, devido à não-linearidade da função de propagação utilizada, o desenvolvimento do cálculo do gradiente acaba alcançando resultados ligeiramente diferentes dos obtidos para o perceptron e o Adaline, como será visto a seguir.

Sejam

- $(L+1)$ o número de camadas da rede (a camada 0 é a camada de entrada e a camada L é a camada de saída);

- n_l o número de neurônios na camada l $(0 \leq l \leq L)$;
- p o número de exemplos;
- $\mathbf{P}$ uma matriz $n_0 \times p$ com as entradas dos exemplos;
- $\mathbf{T}$ uma matriz $n_L \times p$ com as saídas esperadas para as entradas em $\mathbf{P}$;
- $\mathbf{X}_l^{(k)}$ uma matriz $n_{l-1} \times p$ com as entradas para os neurônios da camada l $(0 < l \leq L)$ na k-ésima iteração do treinamento;
- $\mathbf{Y}_l^{(k)}$ uma matriz $n_l \times p$ com as saídas dos neurônios da camada l $(0 \leq l \leq L)$;
- $\mathbf{W}_l^{(k)}$ uma matriz $n_l \times n_{l-1}$ com os pesos dos neurônios da camada l $(0 < l \leq L)$, onde $\left(\mathbf{W}_l^{(k)}\right)_{ij}$ é o peso relativo à j-ésima entrada do i-ésimo neurônio;
- $\mathbf{b}_l^{(k)}$ um vetor $n_l \times 1$ com os *bias* dos neurônios da camada l $(0 < l \leq L)$;
- $f_l : \mathbb{R} \to \mathbb{R}$ a função de propagação da camada l $(0 < l \leq L)$;
- $\mathbf{E}_l^{(k)}$ uma matriz $n_l \times p$ com os erros estimados para os neurônios da camada l $(0 < l \leq L)$;
- $\Delta\mathbf{W}_l^{(k)}$ uma matriz $n_l \times n_{l-1}$ com os incrementos dos pesos dos neurônios da camada l $(0 < l \leq L)$;
- $\Delta\mathbf{b}_l^{(k)}$ um vetor $n_l \times 1$ com os incrementos dos *bias* dos neurônios da camada l $(0 < l \leq L)$;
- $\mathbf{1}$ um vetor $1 \times p$ no qual todas as componentes têm valor 1.

Com isto, desenvolvendo-se o cálculo do gradiente, como visto em (BRAGA; CAR-

VALHO; LUDERMIR, 2000), temos

$$\mathbf{X}_l^{(k)} = \mathbf{Y}_{l-1}^{(k)},\ 0 < l \leq L \tag{2.22}$$

$$\mathbf{net}_l^{(k)} = \mathbf{b}_l^{(k)}\mathbf{1} + \mathbf{W}_l^{(k)}\mathbf{X}_l^{(k)},\ 0 < l \leq L \tag{2.23}$$

$$\mathbf{Y}_l^{(k)} = \begin{cases} \mathbf{P} & \text{se } l = 0 \\ \mathbf{F}_l\left(\mathbf{net}_l^{(k)}\right) & \text{se } 0 < l \leq L \end{cases} \tag{2.24}$$

$$\mathbf{E}_l^{(k)} = \begin{cases} \mathbf{F}_l'\left(\mathbf{net}_l^{(k)}\right) \bullet \left(\left(\mathbf{W}_{l+1}^{(k)}\right)^{\mathrm{T}}\mathbf{E}_{l+1}^{(k)}\right) & \text{se } 0 < l < L \\ \mathbf{F}_l'\left(\mathbf{net}_l^{(k)}\right) \bullet \left(\mathbf{T} - \mathbf{Y}_L^{(k)}\right) & \text{se } l = L \end{cases} \tag{2.25}$$

$$\Delta\mathbf{W}_l^{(k)} = \eta\mathbf{E}_l^{(k)}\left(\mathbf{X}_l^{(k)}\right)^{\mathrm{T}},\ 0 < l \leq L \tag{2.26}$$

$$\Delta\mathbf{b}_l^{(k)} = \eta\mathbf{E}_l^{(k)}\mathbf{1}^{\mathrm{T}},\ 0 < l \leq L, \tag{2.27}$$

onde

$$\left(\mathbf{A} \bullet \mathbf{B}\right)_{ij} = \mathbf{A}_{ij} \times \mathbf{B}_{ij} \tag{2.28}$$

$$\left(\mathbf{F}_l\left(\mathbf{Z}\right)\right)_{ij} = f_l\left(\mathbf{Z}_{ij}\right),\ 0 < l \leq L \tag{2.29}$$

$$\left(\mathbf{F}_l'\left(\mathbf{Z}\right)\right)_{ij} = f_l'\left(\mathbf{Z}_{ij}\right),\ 0 < l \leq L. \tag{2.30}$$

A equação (2.26) é comumente chamada de regra delta generalizada.

Podemos utilizar o Algoritmo 2.2 para o treinamento de uma rede neural MLP. Podemos ter como critérios de parada:

- O número de ciclos de treinamento executados, de modo que o algoritmo pára quando este número excede um limite pré-determinado.
- O erro desejado ter sido alcançado, de modo que o algoritmo pára quando o erro para o conjunto de exemplos passar a ser menor ou igual a um determinado valor.

- A validação, de modo que o algoritmo pára quando o erro num conjunto de exemplos auxiliar (denominado conjunto de validação) deixa de decrescer por um número de iterações especificado.

Algoritmo 2.2 *Back-propagation* para treinamento de redes MLP.

para l **de** 1 **até** L **faça**
  inicie $\mathbf{W}_l$
  inicie $\mathbf{b}_l$
fim de para
repita
  $\mathbf{Y}_0 \leftarrow \mathbf{P}$
  para l **de** 1 **até** L **faça**
    $\mathbf{X}_l \leftarrow \mathbf{Y}_{l-1}$
    calcule $\mathbf{net}_l$ a partir da equação (2.23)
    calcule $\mathbf{Y}_l$ a partir da equação (2.24)
  fim de para
  para l **de** L **até** 1 **faça**
    calcule $\mathbf{E}_l$ a partir da equação (2.25)
    calcule $\Delta\mathbf{W}_l$ a partir da equação (2.26)
    calcule $\Delta\mathbf{b}_l$ a partir da equação (2.27)
  fim de para
  para l **de** 1 **até** L **faça**
    $\mathbf{W}_l \leftarrow \mathbf{W}_l + \Delta\mathbf{W}_l$
    $\mathbf{b}_l \leftarrow \mathbf{b}_l + \Delta\mathbf{b}_l$
  fim de para
até que a condição de parada seja alcançada

2.2.5.2 Variações do Back-propagation

O *Back-propagation*, porém, sofre de algumas deficiências. A principal delas é em relação à velocidade de convergência, normalmente baixa devido a mínimos locais e platôs encontrados na superfície de erro durante o treinamento. Para tentar minimizar os efeitos deste fato, costuma-se utilizar algumas técnicas, como a da adição de um termo de *momentum* e o uso de taxa de aprendizado adaptativa.

O uso do termo de *momentum* consiste em alterar ligeiramente a fórmula de atualização dos pesos, de modo a considerar, também, incrementos calculados anteriormente. Para isso, a equação (2.26) é alterada para

$$\Delta \mathbf{W}_l^{(k)} = \alpha \Delta \mathbf{W}_l^{(k-1)} + (1-\alpha)\left(\eta \mathbf{E}_l^{(k)} \left(\mathbf{X}_l^{(k)}\right)^{\mathrm{T}}\right), \tag{2.31}$$

para $k > 1$ e onde α $(0 < \alpha < 1)$ é chamada constante de *momentum*.

A técnica de taxa de aprendizado adaptativa consiste em alterar, durante o treinamento, a taxa de aprendizado utilizada. As alterações são feitas de modo que sucessivas diminuições no erro implicam no aumento da taxa de aprendizado, enquanto sucessivos aumentos do erro levam à diminuição da taxa de aprendizado.

Além destas ténicas, foram propostas outras variações do *Back-propagation*, como o *Quickprop* e o *Rprop* (BRAGA; CARVALHO; LUDERMIR, 2000).

2.2.6 Redes RBF

Outro tipo de rede neural artificial bastante utilizado é a chamada RBF, do inglês *Radial Basis Function* (Função de Base Radial) (BRAGA; CARVALHO; LUDERMIR, 2000). Este tipo de rede possui uma arquitetura semelhante à de uma rede neural MLP, porém comumente com apenas uma camada escondida, na qual as funções de ativação e propagação são bem definidas.

Ao contrário do produto escalar utilizado nas redes MLP, a função de ativação da camada intermediária das redes RBF é a distância, isto é,

$$net = \frac{||\mathbf{w_i} - \mathbf{x}||}{\sigma_i} = \frac{\sqrt{\sum_{j=1}^{n}(w_{ij} - x_j)^2}}{\sigma_i},$$

onde $\mathbf{w_i}$, o vetor de pesos do i-ésimo neurônio da camada intermediária, corresponde ao centro da gaussiana deste neurônio, e σ_i corresponde à largura desta mesma gaussiana.

Já a função de propagação utilizada na camada escondida é a chamada base radial, que dá nome ao modelo.

Na camada de saída, costuma-se utilizar a função linear como função de propagação. Porém, nada impede que sejam utilizadas outras funções, como a tangente hiperbólica utilizada nas redes MLP.

A adaptação dos parâmetros (pesos e *bias*) da camada de saída ocorre como no *Back-propagation*, através de Gradiente Decrescente. Já a definição dos parâmetros da camada escondida, isto é, os centros e as larguras das gaussianas, pode ser feita manualmente, a partir de amostras do conjunto de dados ou de outras amostras significativas para o problema.

2.2.7 Funções de Propagação

Por fim, listamos nesta seção algumas funções de propagação que podem ser utilizadas na construção de redes neurais. Estas funções podem ser vistas na Figura 2.13 e possuem as seguintes definições:

- Degrau (Figura 2.13(a)):

$$f(x) = \begin{cases} 1 & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases};$$

- Linear pura (Figura 2.13(b)):

$$f(x) = x;$$

- Sigmoide logística (Figura 2.13(c)):

$$f(x) = \frac{1}{1 + e^{-x}};$$

- Tangente hiperbólica logística (Figura 2.13(d)):

$$f(x) = \frac{2}{1 + e^{-2x}} - 1;$$

- Base radial (Figura 2.13(e)):

$$f(x) = e^{-x^2};$$

- Linear saturada (Figura 2.13(f)):

$$f(x) = \begin{cases} 0 & \text{se } x \leq 0 \\ x & \text{se } 0 < x \leq 1 \\ 1 & \text{se } x > 1 \end{cases};$$

2.3 Máquinas de Vetores de Suporte

As Máquinas de Vetores Suporte (SVM, do inglês *Support Vector Machine*) (BOSER; GUYON; VAPNIK, 1992) são classificadores binários que vêm ganhando espaço nos últimos anos devido às propriedades de generalização de seu modelo (MOTA; THOMé, 2009; USHIZIMA; LORENA; CARVALHO, 2005).

Para obter uma boa generalização, um classificador deve ter sua capacidade ajustada para que esta não seja insuficiente (caso em que o classificador não consegue aprender os dados utilizados no treinamento) ou grande demais (caso em que o classificador apresenta pequeno erro para os dados de treinamento, mas apenas para estes). A principal característica do SVM é, então, a de tentar ser um modelo auto-ajustável, visando à busca pela capacidade ideal para o problema a ser tratado.

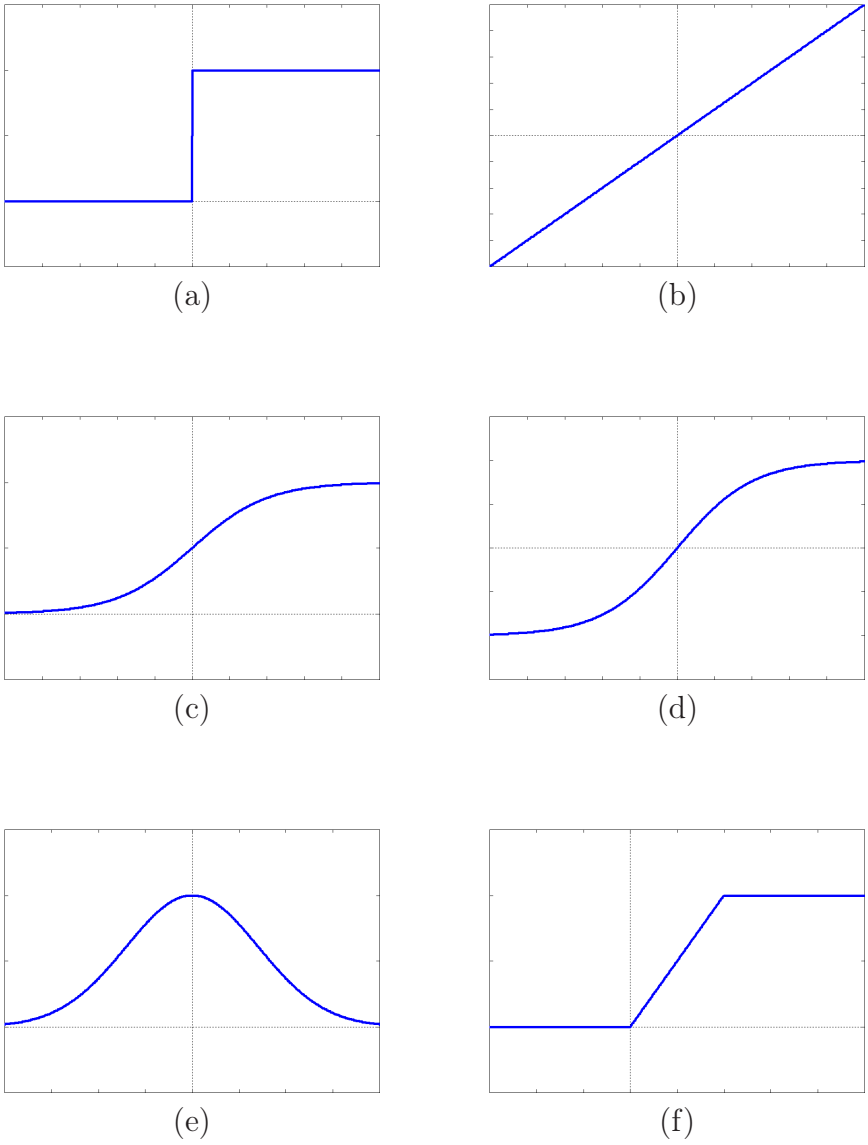


Figura 2.13: Gráficos das funções de propagação.

Para isto, seu algoritmo de treinamento calcula a superfície de separação para a qual a menor distância desta a qualquer ponto representante dos dados de treinamento seja a maior possível. Com isto, a função de decisão passa a ser composta pelos dados de treinamento mais relevantes à classificação desejada, sendo a relevância ajustada automaticamente pelo treinamento.

A Seção 2.3.1 introduz a função de decisão utilizada pelo SVM. A Seção 2.3.2 apresenta a derivação das fórmulas para o treinamento do SVM. A Seção 2.3.3 generaliza o treinamento para o caso não-separável, enquanto a Seção 2.3.4 faz o mesmo para introduzir separação não-linear.

2.3.1 Classificação Utilizando SVM

A superfície utilizada pelo modelo SVM para realizar a separação é o hiperplano. Sejam, então, $\mathbf{w}$ e b, respectivamente o vetor normal ao hiperplano e seu deslocamento em relação à origem. Assim, temos que a função de decisão para uma entrada $\mathbf{x}$ é

$$d(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b,$$

e

$$\mathbf{x} \in \begin{cases} A & \text{se } d(\mathbf{x}) > 0 \\ B & \text{caso contrário} \end{cases}.$$

Como pode ser visto na Figura 2.14, a distância (com sinal) de $\mathbf{x}$ em relação ao hiperplano é

$$\frac{d(\mathbf{x})}{\|\mathbf{w}\|}. \tag{2.32}$$

Assim, $d(\mathbf{x_1})$ e $d(\mathbf{x_2})$ têm sinais opostos se, e somente se, $\mathbf{x_1}$ e $\mathbf{x_2}$ estão em lados opostos do hiperplano.

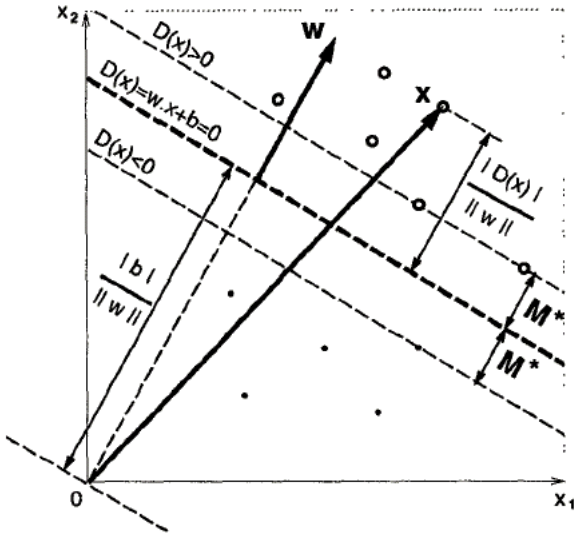


Figura 2.14: Exemplo de hiperplano de separação (em duas dimensões), distâncias e margem. Retirada de (BOSER; GUYON; VAPNIK, 1992).

2.3.2 Treinamento

Para o treinamento, sejam as amostras $\mathbf{x_1}, \ldots, \mathbf{x_p}$ e suas respectivas classificações desejadas $y_1, \ldots, y_p$, onde

$$y_i = \begin{cases} +1 & \text{se } \mathbf{x_i} \in A \\ -1 & \text{se } \mathbf{x_i} \in B \end{cases}, \quad \text{para } i = 1, \ldots, p.$$

Supondo que as amostras são linearmente separáveis, há pelo menos um par de $\mathbf{w}$ e b tal que

$$d(\mathbf{x_i}) \geq +1, \quad \forall\, i, y_i = +1, \tag{2.33}$$

$$d(\mathbf{x_i}) \leq -1, \quad \forall\, i, y_i = -1, \tag{2.34}$$

e, além disto, tal que pelo menos um representate de cada classe transforma a respectiva inequação acima em uma equação.

Podemos reunir estas inequações em uma só, multiplicando y_i nos dois lados de cada uma, de modo que

$$y_i d(\mathbf{x_i}) \geq 1, \quad \text{para } i = 1, \ldots, p. \tag{2.35}$$

Podemos fazer o mesmo com (2.32), obtendo a distância (sem sinal) de $\mathbf{x}$ ao hiperplano:

$$\frac{y_i d(\mathbf{x_i})}{\|\mathbf{w}\|}, \quad \text{para } i = 1, \ldots, p. \tag{2.36}$$

Alterando a inequação (2.35) para que o lado esquerdo fique igual a (2.36), temos

$$\frac{y_i d(\mathbf{x_i})}{\|\mathbf{w}\|} \geq \frac{1}{\|\mathbf{w}\|} = M, \quad \text{para } i = 1, \ldots, p,$$

onde M é a *margem*, isto é, a menor distância ao hiperplano de separação de pelo menos um representante de cada classe.

Argumenta-se que a melhor generalização de classificação é obtida quando M é máximo. Então, temos de maximizar M, sujeito às restrições da inequação (2.35). Porém, maximizar M equivale a minimizar $\|\mathbf{w}\|$. Obtemos, com isto, o seguinte problema de programação quadrática:

$$\begin{aligned} &\min \ \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{sujeito a} \\ &\qquad y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1 \geq 0, \quad \text{para } i = 1, \ldots, p. \end{aligned}$$

Poderíamos resolver este problema utilizando métodos conhecidos, encontrando o vetor $\mathbf{w}^*$ e o escalar b^* soluções. Porém, podemos desenvolver este problema de modo a obter mais informações sobre a solução. Além disto, tal desenvolvimento será fundamental para os casos tratados nas seções seguintes.

Podemos (LUENBERGER, 1984) introduzir multiplicadores de Lagrange α_i ($i = 1, \ldots, p$) não negativos, um para cada restrição, e inseri-las na função objetivo,

formando o seguinte Lagrangiano:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{p} \alpha_i (y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1)$$

sujeito a

$$\alpha_i \geq 0, \quad \text{para } i = 1, \ldots, p,$$

onde $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_p)^{\mathrm{T}}$.

A superfície formada por L no espaço de $\mathbf{w}$, b e $\boldsymbol{\alpha}$ tem forma de cela, de modo que o mínimo em relação a $\mathbf{w}$ e b (primal) corresponde ao máximo em relação a $\boldsymbol{\alpha}$ (dual).

Sabe-se que, no ponto solução, as seguintes condições devem ser satisfeitas:

$$\left.\frac{\partial L}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^*} = \mathbf{w}^* - \sum_{i=1}^{p} \alpha_i y_i \mathbf{x_i} = 0,$$
$$\left.\frac{\partial L}{\partial b}\right|_{b=b^*} = -\sum_{i=1}^{p} \alpha_i y_i = 0,$$

de modo que

$$\mathbf{w}^* = \sum_{i=1}^{p} \alpha_i y_i \mathbf{x_i}. \tag{2.37}$$

Substituindo estes resultados no Lagrangiano L, obtemos um novo Lagrangiano

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{p} \alpha_i - \frac{1}{2}\mathbf{w}^* \cdot \mathbf{w}^*,$$

e, substituindo (2.37) novamente,

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{p} \alpha_i - \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \mathbf{x_i} \cdot \mathbf{x_j}.$$

Com isto, temos o seguinte problema de programação quadrática:

$$\max \ \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}\boldsymbol{\alpha}$$

sujeito a

$$\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{y} = 0$$
$$\boldsymbol{\alpha} \geq \mathbf{0},$$

onde $\mathbf{y} = (y_1, \ldots, y_p)^{\mathrm{T}}$, $\mathbf{0} = (0, \ldots, 0)$ e $\mathbf{1} = (1, \ldots, 1)$ têm dimensão p e $\mathbf{H}_{p \times p}$ é tal que

$$\mathbf{H}_{ij} = y_i y_j \mathbf{x_i} \cdot \mathbf{x_j}, \quad \text{para } i = 1, \ldots, p, \ j = 1, \ldots, p.$$

Após resolver este problema por métodos usuais (LUENBERGER, 1984), podemos encontrar $\mathbf{w}$ a partir da equação (2.37).

Além disto, uma condição é imposta pelo Teorema de Kühn-Tucker, a qual relaciona o multiplicador de Lagrange com sua respectiva restrição:

$$\alpha_i^*(y_i(\mathbf{x_i} \cdot \mathbf{w}^* + b^*) - 1) = 0, \quad \text{para } i = 1, \ldots, p. \tag{2.38}$$

Se $\alpha_i^* \neq 0$, então

$$y_i(\mathbf{x_i} \cdot \mathbf{w}^* + b^*) - 1 = 0, \quad \text{para } i = 1, \ldots, p,$$

ou seja,

$$y_i d(\mathbf{x_i}) = 1, \quad \text{para } i = 1, \ldots, p.$$

Deste modo, $\mathbf{x_i}$ é uma das amostras para as quais a inequação (2.35) se torna uma equação, sendo, então, chamada *vetor suporte*, o qual dá nome ao modelo.

O valor de b pode ser calculado a partir da equação (2.38), utilizando as amostras para as quais $\alpha_i \neq 0$.

2.3.3 Caso Não-Separável

Na seção anterior, supomos que as amostras de treinamento eram separáveis. Caso não fossem separáveis, o método divergiria, com sua função objetivo crescendo arbitrariamente.

Para contornar isto, podemos fazer uma pequena alteração na formulação do problema. Inserimos variáveis de folga δ_i $(i = 1, \ldots, p)$ nas inequações (2.33) e (2.34), de modo que obtemos

$$\begin{aligned} d(\mathbf{x_i}) &\geq +1 - \delta_i, \quad \forall\ i, y_i = +1, \\ d(\mathbf{x_i}) &\leq -1 + \delta_i, \quad \forall\ i, y_i = -1, \\ \delta_i &\geq 0, \quad \forall\ i = 1, \ldots, p. \end{aligned}$$

Reescrevendo, temos

$$y_i d(\mathbf{x_i}) \geq 1 - \delta_i, \quad \text{para } i = 1, \ldots, p.$$

Como o nome sugere, as variáveis de folga dão alguma liberdade para que algumas amostras não respeitem as inequações (2.33) e (2.34). Porém, devemos tentar minimizar a quantidade de variáveis que se utilizam de sua "folga", além de minimizar, também, o tamanho desta.

Para isto, podemos somar à função objetivo a ser minimizada uma penalização com o seguinte formato:

$$C \sum_{i=1}^{p} \delta_i,$$

onde C indica a "força" da penalização, sendo maior quanto maior for seu valor.

Assim, temos um novo problema de minimização quadrática:

$$\begin{aligned} &\min\ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{p} \delta_i \\ &\text{sujeito a} \\ &\qquad y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1 + \delta_i \geq 0, \quad \text{para } i = 1, \ldots, p \\ &\qquad \delta_i \geq 0, \quad \text{para } i = 1, \ldots, p. \end{aligned}$$

Do mesmo modo que procedemos na seção anterior, podemos introduzir multiplicadores de Lagrange λ_i $(i = 1, \ldots, p)$ para as novas restrições sobre as variáveis de

folga δ_i, obtendo o Lagrangiano

$$L(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\lambda}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{p}\delta_i - \sum_{i=1}^{p}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}+b)-1) - \sum_{i=1}^{p}\lambda_i\delta_i$$

sujeito a

$$\begin{aligned}\alpha_i &\geq 0, \quad \text{para } i = 1, \ldots, p\\ \lambda_i &\geq 0, \quad \text{para } i = 1, \ldots, p,\end{aligned}$$

onde $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_p)^{\mathrm{T}}$ e $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)^{\mathrm{T}}$.

Ainda como na seção anterior, podemos desenvolver este Lagrangiano, utilizar as condições sobre as derivadas parciais, montar um novo Lagrangiano e obter um novo problema de programação quadrática:

$$\begin{aligned}&\max\ \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}\boldsymbol{\alpha}\\ &\text{sujeito a}\\ &\quad \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{y} = 0\\ &\quad \mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{c},\end{aligned}$$

onde $\mathbf{c} = (C, \ldots, C)$ tem dimensão p.

É interessante notar que a única alteração no problema final foi a limitação em C para o valor dos α_i $(i = 1, \ldots, p)$.

2.3.4 Separação Não-Linear

Podemos generalizar os métodos das seções anteriores para que a separação passe a ser não-linear. Porém, se fizermos isso explicitamente, alterando a função de decisão, teremos de alterar toda a formulação obtida até então.

Ao invés disto, podemos transformar os vetores do espaço de entrada para um espaço de atributo, através de uma função não-linear

$$\boldsymbol{\phi} : \mathbb{R}^r \to \mathbb{R}^N,$$

onde r é a dimensão do espaço de entrada e N é a dimensão do espaço de atributo, normalmente com $N > r$. O hiperplano continua sendo a superfície de separação; porém, este é traçado no espaço de atributo.

Com isto, podemos utilizar os mesmos métodos das seções anteriores, apenas substituindo as ocorrências das amostras $\mathbf{x_i}$ por suas imagens $\boldsymbol{\phi}(\mathbf{x_i})$ $(i = 1, \ldots, p)$.

Porém, para tirarmos mais proveito desta modificação, não trabalharemos explicitamente com a função $\boldsymbol{\phi}$.

Como todas as ocorrências das amostras estão na forma de produtos $\boldsymbol{\phi}(\mathbf{x_i}) \cdot \boldsymbol{\phi}(\mathbf{x_j})$, podemos utilizar o "truque do *kernel*" (BURGES, 1998): substituímos estes produtos por uma função de *kernel* $K(\mathbf{x_i}, \mathbf{x_j})$. Esta função indica como calcular o produto interno no espaço de atributo, porém, sem explicitar a função $\boldsymbol{\phi}$, a qual pode assumir diversas formas e dimensões, para as quais seria impraticável o uso direto desta função.

Alguns exemplos de funções de *kernel* são:

$$\begin{aligned} K(\mathbf{u}, \mathbf{v}) &= (\mathbf{u} \cdot \mathbf{v} + 1)^p, \\ K(\mathbf{u}, \mathbf{v}) &= e^{-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}}, \\ K(\mathbf{u}, \mathbf{v}) &= \tanh(\kappa \mathbf{u} \cdot \mathbf{v} - \nu). \end{aligned}$$

A alteração no treinamento é mínima:

$$\begin{aligned} \mathbf{H}_{ij} &= y_i y_j \boldsymbol{\phi}(\mathbf{x_i}) \cdot \boldsymbol{\phi}(\mathbf{x_j}) \\ &= y_i y_j K(\mathbf{x_i}, \mathbf{x_j}), \quad \text{para } i = 1, \ldots, p,\ j = 1, \ldots, p \end{aligned}$$

O cálculo de $\mathbf{w}^*$ também sofre pequena alteração:

$$\mathbf{w}^* = \sum_{i=1}^{p} \alpha_i y_i \boldsymbol{\phi}(\mathbf{x_i}).$$

Porém, como nesta formulação $\mathbf{w}^*$ é um vetor do espaço de atributo, não podemos utilizá-lo diretamente.

Para contonar esta dificuldade, substituímos o cálculo de $\mathbf{w}^*$ na nova função de decisão

$$d(\mathbf{x}) = \mathbf{w}^* \cdot \boldsymbol{\phi}(\mathbf{x}) + b,$$

obtendo

$$d(\mathbf{x}) = \sum_{i=1}^{p} \alpha_i y_i \boldsymbol{\phi}(\mathbf{x_i}) \cdot \boldsymbol{\phi}(\mathbf{x}) + b.$$

Substituindo o produto escalar pela função de *kernel*, temos

$$d(\mathbf{x}) = \sum_{i=1}^{p} \alpha_i y_i K(\mathbf{x_i}, \mathbf{x}) + b.$$

Com isto, a decisão sobre a classificação de uma entrada $\mathbf{x}$ depende, apenas, dos vetores suporte $\mathbf{x_i}$, pois são aqueles para os quais seus α_i são não nulos.

Além disto, como notado em (CORTES; VAPNIK, 1995), a estrutura da decisão de um SVM é semelhante à de uma rede neural *feedforward* com q neurônios na camada intermediária, onde q é a quantidade de vetores suporte. O j-ésimo "neurônio da camada intermediária" avalia a função de *kernel* sobre a entrada $\mathbf{x}$ e seu vetor suporte correspondente $\mathbf{x_{i_j}}$, enquanto o "neurônio da camada de saída" realiza a soma dos resultados das aplicações da função de *kernel*, ponderados pelos α_{i_j}, que funcionam como pesos.

Em particular, quando a função de *kernel* utilizada é a da tangente hiperbólica, o SVM se comporta exatamente como uma rede neural MLP, na qual as funções de

propagação das camadas intermediária e de saída são, respectivamente, a tangente hiperbólica e a linear. Além disto, neste caso os vetores suporte $\mathbf{x_{i_j}}$ funcionam como os pesos da camada intermediária, enquanto os α_{i_j} fazem a parte dos pesos da camada de saída.

Podemos notar, então, que o treinamento do SVM busca ajustar a capacidade do modelo ao problema em questão, selecionando, para isso, a quantidade adequada de vetores suporte.

2.3.5 Regressão

As mesmas idéias utilizadas na construção de máquinas de vetores de suporte para problemas de classificação podem ser empregadas na adaptação daquelas a problemas de regressão (DRUCKER et al., 1997). Nestes, desejamos obter uma função $d(\mathbf{x})$ que seja o mais próxima possível de uma função $f(\mathbf{x})$ desconhecida, para a qual dispomos de apenas algumas amostras.

Para isso, uma máquina de vetores de suporte para regressão fornece uma função

$$d(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b,$$

semelhante à função utilizada para classificação. As diferenças em relação a esta surgem no treinamento, principalmente no tratamento dado às amostras para definição da margem.

As amostras, como na classificação, são compostas por pares $(\mathbf{x_i}, y_i)$ $(i = 1, \ldots, p)$. Porém, ao invés de estarem limitados ao conjunto $\{-1, +1\}$, os y_i podem assumir quaisquer valores em $\mathbb{R}$.

Já a margem é determinada por um valor ε, definindo o erro máximo permitido

para cada amostra em relação à função $d(\mathbf{x})$. Isto é, para cada amostra $(\mathbf{x_i}, y_i)$ $(i = 1, \ldots, p)$, devemos ter,

$$d(\mathbf{x_i}) - \varepsilon \leq y_i \leq d(\mathbf{x_i}) + \varepsilon.$$

Isto, somado ao fato de querermos minimizar a norma de $\mathbf{w}$ (SMOLA; SCHöLKOPF, 2004), nos leva a um primeiro problema de programação quadrática:

$$\begin{aligned}
&\min \ \frac{1}{2}\|\mathbf{w}\|^2 \\
&\text{sujeito a} \\
&\qquad y_i - \mathbf{w} \cdot \mathbf{x} - b \leq \varepsilon, \quad \text{para } i = 1, \ldots, p \\
&\qquad \mathbf{w} \cdot \mathbf{x} + b - y_i \leq \varepsilon, \quad \text{para } i = 1, \ldots, p.
\end{aligned}$$

Naturalmente, em problemas de regressão mais complexos torna-se difícil atender às restrições do problema de programação quadrática, ao menos que aumentemos a margem ε de forma que o erro torna-se impraticável. Netes casos, podemos proceder como na Seção 2.3.3, introduzindo variáveis de folga para permitir que o erro para algumas amostras seja maior que ε, dando maior flexibilidade à solução do problema.

No caso de máquinas de vetores de suporte para regressão, as variáveis de folga para cada amostra $(\mathbf{x_i}, y_i)$ vêm em pares (δ_i^+, δ_i^-): δ_i^+ representa a folga acima de $d(\mathbf{x_i})$, enquanto δ_i^-, a folga abaixo de $d(\mathbf{x_i})$. Ou seja,

$$d(\mathbf{x_i}) - \varepsilon - \delta_i^- \leq y_i \leq d(\mathbf{x_i}) + \varepsilon + \delta_i^+,$$

para $i = 1, \ldots, p$. Como uma amostra não pode estar ao mesmo tempo acima e abaixo de $d(\mathbf{x_i})$, δ_i^+ e δ_i^- não podem ser, simultaneamente, maiores que zero.

Assim, temos um novo problema de programação quadrática

$$\begin{aligned}
&\min\ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{p}\left(\delta_i^+ + \delta_i^-\right)\\
&\text{sujeito a}\\
&\qquad y_i - \mathbf{w}\cdot\mathbf{x} - b \le \varepsilon + \delta_i^+, \quad \text{para } i = 1,\ldots,p\\
&\qquad \mathbf{w}\cdot\mathbf{x} + b - y_i \le \varepsilon + \delta_i^-, \quad \text{para } i = 1,\ldots,p\\
&\qquad \delta_i^+ \ge 0, \quad \text{para } i = 1,\ldots,p\\
&\qquad \delta_i^- \ge 0, \quad \text{para } i = 1,\ldots,p,
\end{aligned}$$

no qual desejamos minimizar, além da norma de $\mathbf{w}$, a soma das folgas utilizadas. Mais uma vez, C tem o papel de dar peso à penalização por uso de folga.

Também como na Seção 2.3.3, podemos introduzir multiplicadores de Lagrange, desenvolver o Lagrangiano, utilizar as condições sobre as derivadas parciais, montar um novo Lagrangiano e, finalmente, obter um novo problema de programação quadrática, dual do original:

$$\begin{aligned}
&\max\ \left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right)^{\mathrm{T}}\mathbf{y} - \left(\boldsymbol{\alpha}^+ + \boldsymbol{\alpha}^-\right)^{\mathrm{T}}\boldsymbol{\varepsilon} - \frac{1}{2}\left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right)^{\mathrm{T}}\mathbf{H}\left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right)\\
&\text{sujeito a}\\
&\qquad \left(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-\right)^{\mathrm{T}}\mathbf{1} = 0\\
&\qquad \mathbf{0} \le \boldsymbol{\alpha}^+, \boldsymbol{\alpha}^- \le \mathbf{c},
\end{aligned}$$

onde $\boldsymbol{\alpha}^+ = (\alpha_1^+,\ldots,\alpha_p^+)$, $\boldsymbol{\alpha}^- = (\alpha_1^-,\ldots,\alpha_p^-)$, $\boldsymbol{\varepsilon} = (\varepsilon,\ldots,\varepsilon)$ têm dimensão p e $\mathbf{H}_{p\times p}$ é tal que

$$\mathbf{H}_{ij} = \mathbf{x_i}\cdot\mathbf{x_j}, \quad \text{para } i = 1,\ldots,p,\ j = 1,\ldots,p.$$

Além disso, como anteriormente, temos

$$\mathbf{w} = \sum_{i=1}^{p}\left(\alpha_i^+ - \alpha_i^-\right)\mathbf{x_i},$$

o que nos leva a uma nova função

$$d(\mathbf{x}) = \sum_{i=1}^{p}\left(\alpha_i^+ - \alpha_i^-\right)\mathbf{x_i}\cdot\mathbf{x} + b,$$

dependente dos vetores de suporte $\mathbf{x_i}$ para os quais $\left(\alpha_i^+ - \alpha_i^-\right) \neq 0$.

Por fim, podemos proceder como na Seção 2.3.4, introduzindo funções de *kernel*. Com isso, temos

$$\mathbf{H}_{ij} = K(\mathbf{x_i}, \mathbf{x_j}), \quad \text{para } i = 1, \ldots, p, \ j = 1, \ldots, p$$

e

$$d(\mathbf{x}) = \sum_{i=1}^{p} \left(\alpha_i^+ - \alpha_i^-\right) K(\mathbf{x_i}, \mathbf{x}) + b.$$

Podemos, ainda, após o treinamento, definir $\alpha_i = \alpha_i^+ - \alpha_i^-$, de modo que a função

$$d(\mathbf{x}) = \sum_{i=1}^{p} \alpha_i K(\mathbf{x_i}, \mathbf{x}) + b$$

fique idêntica à função de decisão das máquinas de vetores de suporte para classificação.

3 AMBIENTE E ROBÔ VIRTUAIS

Apesar do objetivo do trabalho ser a construção de um controlador que esteja pronto para ser utilizado num robô autônomo real, devemos, numa primeira fase, realizar testes em um ambiente virtual. Optamos por esta abordagem porque, apesar das simplificações que temos num modelo virtual, o desenvolvimento e os testes dos métodos num ambiente deste tipo são muito mais simples, principalmente pela facilidade em se automatizar diversas etapas do processo.

Além disso, podemos focar os esforços justamente no desenvolvimento e nos testes dos modelos e métodos, enquanto num ambiente real teríamos de nos preocupar, também, com diversos detalhes fora desse escopo, os quais podem ser deixados para uma próxima fase, de implantação do controlador num robô real.

Assim, temos um ambiente virtual, no qual inserimos um robô virtual, de modo a realizar simulações para o desenvolvimento do controlador do robô autônomo.

3.1 Ambiente Virtual

O ambiente virtual consiste, basicamente, num corredor no qual estão dispostos alguns obstáculos, como pode ser visto na Figura 3.1. Como, para fins deste trabalho, podemos ignorar a altura dos objetos, podemos considerar o ambiente como visto de cima. Com isso, levamos em conta apenas as componentes horizontal (eixo x) e vertical (eixo y) dos objetos do ambiente.

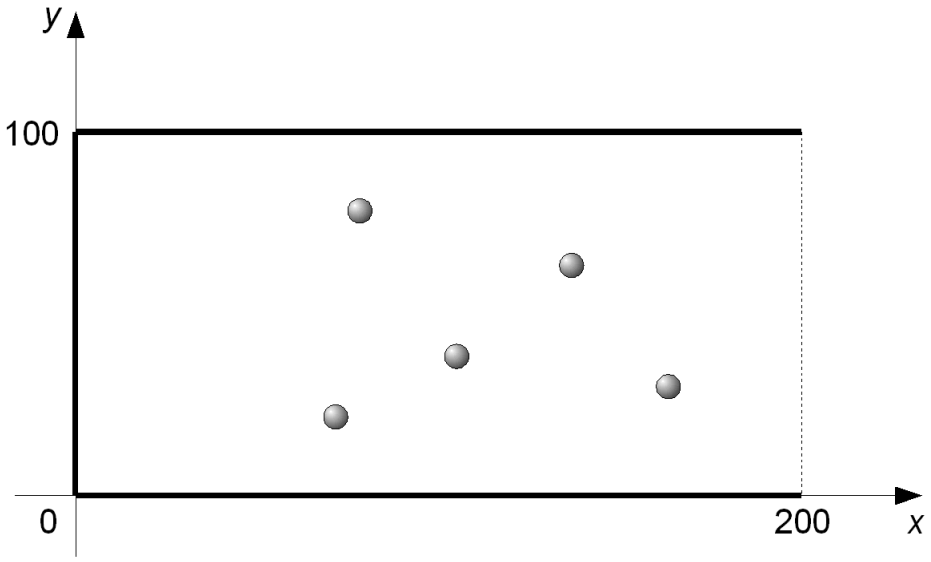


Figura 3.1: Ambiente virtual, composto por corredor e obstáculos.

O corredor é retangular, tendo 200 unidades de medida em seu comprimento e 100 unidades de medida em sua largura. O canto inferior esquerdo do ambiente corresponde à coordenada $(0, 0)$, enquanto o canto superior direito, à coordenada $(200, 100)$. Além disso, o corredor é delimitado por paredes horizontais nas partes superior e inferior, e por uma parede vertical na lateral esquerda.

Os obstáculos, por sua vez, têm formato circular, com 3 unidades de medida de raio.

As posições dos obstáculos são definidas no início da simulação, e permanecem fixas até o final desta.

3.2 Robô Virtual

O robô, assim como os obstáculos, tem formato circular, com 6 unidades de medida de raio. Ao contrário dos obstáculos, porém, o robô pode perceber certos detalhes do ambiente e se movimentar, fazendo-o para alcançar seu objetivo.

A posição do robô em cada instante t é dada por uma tripla $\left(x_r^{(t)}, y_r^{(t)}, \phi^{(t)}\right)$, onde $\left(x_r^{(t)}, y_r^{(t)}\right)$ refere-se à coordenada do robô no plano do ambiente e $\phi^{(t)}$ corresponde ao ângulo da parte frontal do robô com a horizontal. A Figura 3.2 mostra o robô inserido no ambiente virtual.

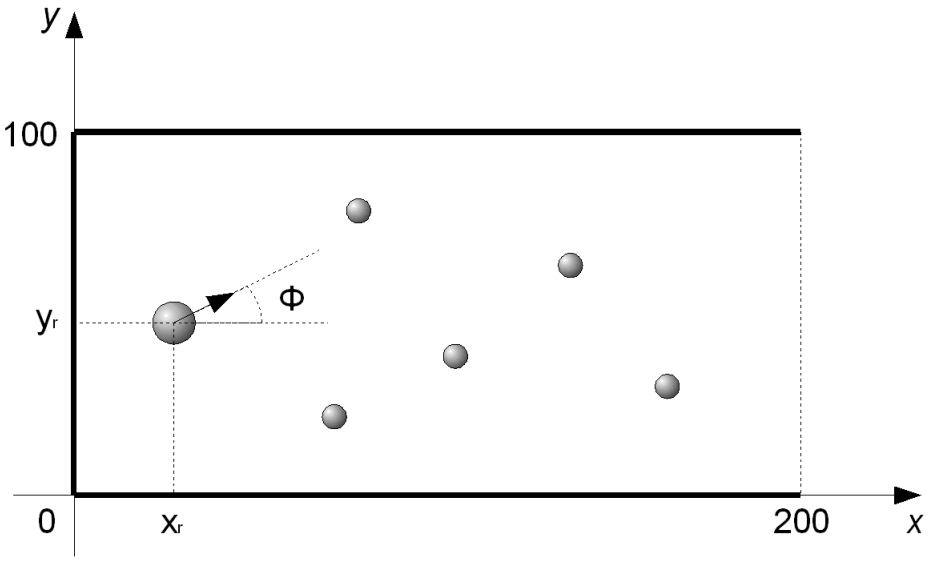


Figura 3.2: Robô virtual inserido no ambiente virtual. A seta indica o direcionamento da parte frontal do robô.

Inicialmente, o robô é colocado junto à parede esquerda, isto é, numa posição em que $x_r^0 = 6$. Neste caso, y_r^0 pode variar entre 6 e 94, bem como ϕ^0 pode assumir valores entre -90^o e 90^o. A partir daí, o robô deve se movimentar para atingir seu objetivo: alcançar o lado direito do corredor, isto é, fazer com que $x_r \geq 194$, sem colidir com as paredes e os obstáculos encontrados no ambiente. A simulação é abortada e dizemos que o robô não obteve sucesso caso este colida com uma das paredes ou com um dos obstáculos.

Assim, o robô é composto de sensores, módulo decisor e módulo de movimentação. O módulo decisor do robô é o responsável por tratar a percepção do ambiente, obtida através dos sensores, e tomar decisões quanto à movimentação, como pode ser visto no esquema da Figura 3.3.

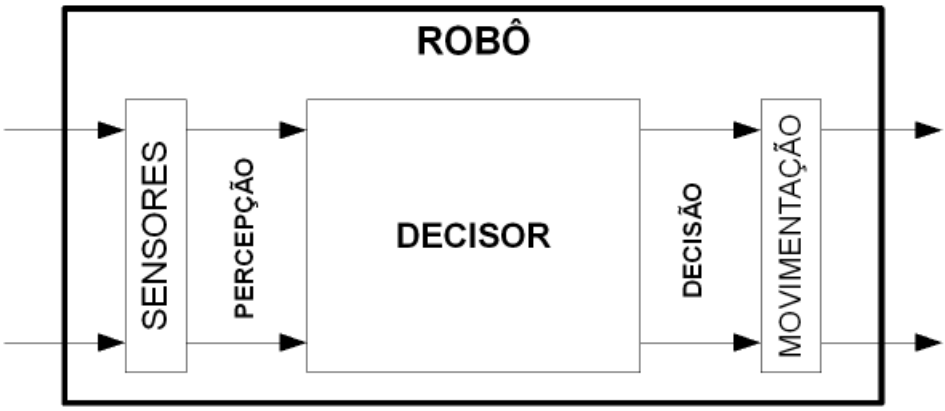


Figura 3.3: Esquema de funcionamento do robô virtual, composto por módulo decisor, sensores e módulo de movimentação.

3.2.1 Sensores

O robô não possui, inicialmente, informações sobre as posições dos obstáculos no ambiente, nem sobre qual é sua posição a cada momento. Assim, para que não colida

com obstáculos e paredes, o robô deve possuir sensores através dos quais perceba informações sobre o ambiente.

Para montar o robô real, temos à disposição os seguintes sensores:

- Sensor de distância: informa a distância percorrida pelo ultrassom até a colisão com algum obstáculo (incluindo parede), na direção apontada pelo sensor; e
- Sensor de bússola: informa a direção para a qual o sensor aponta, através da medição do campo magnético da Terra.

Como o objetivo é que o controlador construído neste trabalho esteja pronto para ser introduzido num robô real, simulamos os sensores listados acima no robô virtual, ao contrário dos utilizados em (MORATORI, 2006).

Os três sensores de distância disponíveis são posicionados no centro do robô, apontando para as direções a $\alpha_1^\star = 15^\text{o}$, $\alpha_2^\star = 0^\text{o}$ e $\alpha_3^\star = -15^\text{o}$ em relação à frente do robô. Logo, os ângulos dos sensores em relação à horizontal do ambiente no instante de simulação t são $\alpha_1^{(t)} = \alpha_1^\star + \phi^{(t)}$, $\alpha_2^{(t)} = \alpha_2^\star + \phi^{(t)}$ e $\alpha_3^{(t)} = \alpha_3^\star + \phi^{(t)}$.

Para cada sensor i ($i = 1, 2, 3$), definimos uma semirreta partindo do centro do robô virtual e com ângulo $\alpha_i^{(t)}$, e calculamos suas possíveis interseções com obstáculos (representados por círculos) e paredes (representadas por segmentos de reta). A distância à mais próxima destas interseções corresponde à distância que seria medida pelo sensor no robô real.

Para calcular os pontos de interseção do raio emitido pelo sensor com obstáculos e paredes, podemos utilizar parametrização. Com isso, a semirreta do sensor i é dada

por:

$$x_i^{(t)}(s) = x_r^{(t)} + s\cos\alpha_i^{(t)}, \quad (3.1)$$

$$y_i^{(t)}(s) = y_r^{(t)} + s\sin\alpha_i^{(t)}, \quad (3.2)$$

onde s ($s \geq 0$) é o parâmetro livre.

Para as paredes, inicialmente desconsideramos os limites dos segmentos de reta (isto é, consideramo-os como sendo retas), calculamos as interseções e verificamos se estas estão dentro dos limites dos segmentos. Assim, para as paredes horizontais superior ($y_p = 100$) e inferior ($y_p = 0$), igualamos $y_i^{(t)}(s)$ a y_p e utilizamos a Equação (3.2) para obter o valor de s, ou seja,

$$\begin{aligned} y_i^{(t)}(s) &= y_p \\ y_r^{(t)} + s\sin\alpha_i^{(t)} &= y_p \\ s\sin\alpha_i^{(t)} &= y_p - y_r^{(t)} \\ s &= \frac{y_p - y_r^{(t)}}{\sin\alpha_i^{(t)}}. \end{aligned}$$

Se $\sin\alpha_i^{(t)} \neq 0$ (a semirreta não é horizontal) e $s \geq 0$, temos $x_p = x_i^{(t)}(s)$ para o valor de s calculado acima. Se $0 \leq x_p \leq 200$, (x_p, y_p) é o ponto de interseção entre a semirreta do sensor e o segmento de reta que representa a parede em questão.

A Figura 3.4 mostra um exemplo no qual identificamos a interseção entre o sensor 2 com a parede horizontal de $y_p = 100$.

De modo análogo, para a parede vertical esquerda ($x_p = 0$) utilizamos a Equação

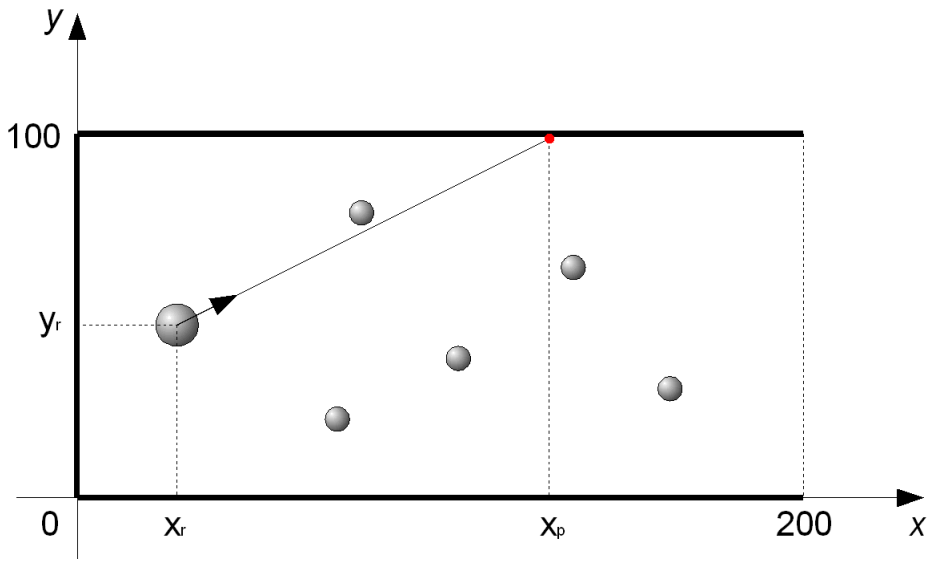


Figura 3.4: Exemplo de interseção entre o sensor 2 ($\alpha_2^\star = 0^\text{o}$) e a parede horizontal de $y_p = 100$.

(3.1) para obter o valor de s:

$$\begin{aligned} x_i^{(t)}(s) &= x_p \\ x_r^{(t)} + s\cos\alpha_i^{(t)} &= x_p \\ s\cos\alpha_i^{(t)} &= x_p - x_r^{(t)} \\ s &= \frac{x_p - x_r^{(t)}}{\cos\alpha_i^{(t)}}. \end{aligned}$$

Se $\cos\alpha_i^{(t)} \neq 0$ (a semirreta não é vertical) e $s \geq 0$, temos $y_p = y_i^{(t)}(s)$ a partir do valor de s calculado acima. Se $0 \leq y_p \leq 100$, (x_p, y_p) é o ponto de interseção entre a semirreta do sensor e o segmento de reta que representa a parede.

Já para o cálculo das interseções com os círculos dos obstáculos, seja (x_{o_j}, y_{o_j}) o centro do j-ésimo obstáculo ($j = 1, \ldots, n_o$, onde n_o é o número de obstáculos). Este

obstáculo pode ser representado da seguinte forma:

$$\left(x - x_{o_j}\right)^2 + \left(y - y_{o_j}\right)^2 - r_{o_j}^2 = 0, \tag{3.3}$$

onde $r_{o_j} = 3$ é o raio do obstáculo. Unindo (3.3) a (3.1) e (3.2) através de $x = x_i^{(t)}(s)$ e $y = y_i^{(t)}(s)$, temos

$$\begin{aligned}
\left(x - x_{o_j}\right)^2 + \left(y - y_{o_j}\right)^2 - r_{o_j}^2 &= 0 \\
\left(x_i^{(t)}(s) - x_{o_j}\right)^2 + \left(y_i^{(t)}(s) - y_{o_j}\right)^2 - r_{o_j}^2 &= 0 \\
\left(x_r^{(t)} - x_{o_j} + s\cos\alpha_i^{(t)}\right)^2 + \left(y_r^{(t)} - y_{o_j} + s\sin\alpha_i^{(t)}\right)^2 - r_{o_j}^2 &= 0 \\
\left(x_r^{(t)} - x_{o_j}\right)^2 + 2\left(x_r^{(t)} - x_{o_j}\right)\left(s\cos\alpha_i^{(t)}\right) + \left(s\cos\alpha_i^{(t)}\right)^2 + & \\
+ \left(y_r^{(t)} - y_{o_j}\right)^2 + 2\left(y_r^{(t)} - y_{o_j}\right)\left(s\sin\alpha_i^{(t)}\right) + \left(s\sin\alpha_i^{(t)}\right)^2 - r_{o_j}^2 &= 0 \\
s^2 + 2\left(\left(x_r^{(t)} - x_{o_j}\right)\cos\alpha_i^{(t)} + \left(y_r^{(t)} - y_{o_j}\right)\sin\alpha_i^{(t)}\right)s + & \\
+ \left(\left(x_r^{(t)} - x_{o_j}\right)^2 + \left(y_r^{(t)} - y_{o_j}\right)^2 - r_{o_j}^2\right) &= 0.
\end{aligned}$$

Resolvemos, então, a equação acima, a qual pode ter de 0 a 2 soluções. Caso haja soluções reais tais que $s \geq 0$, escolhemos aquela em que s é o menor possível (ou seja, aquela para a qual o ponto de interseção é o mais próximo possível do centro do robô). Neste caso, o ponto de interseção entre a semirreta do sensor e o círculo do obstáculo é $\left(x_i^{(t)}(s), y_i^{(t)}(s)\right)$, para o valor de s encontrado.

Na Figura 3.5 vemos outro exemplo de interseção, desta vez entre o sensor 3 e um dos objetos do ambiente.

O sensor de bússola pode ser facilmente inserido na simulação, uma vez que o módulo simulador conhece o ângulo $\phi^{(t)}$ do robô com a horizontal. Como já discutido no início do capítulo, para simplificar a implementação e abstrair alguns detalhes técnicos o sensor de bússola simulado informa ao robô justamente o ângulo $\phi^{(t)}$. Num robô real, porém, o valor informado seria da medição do campo magnético

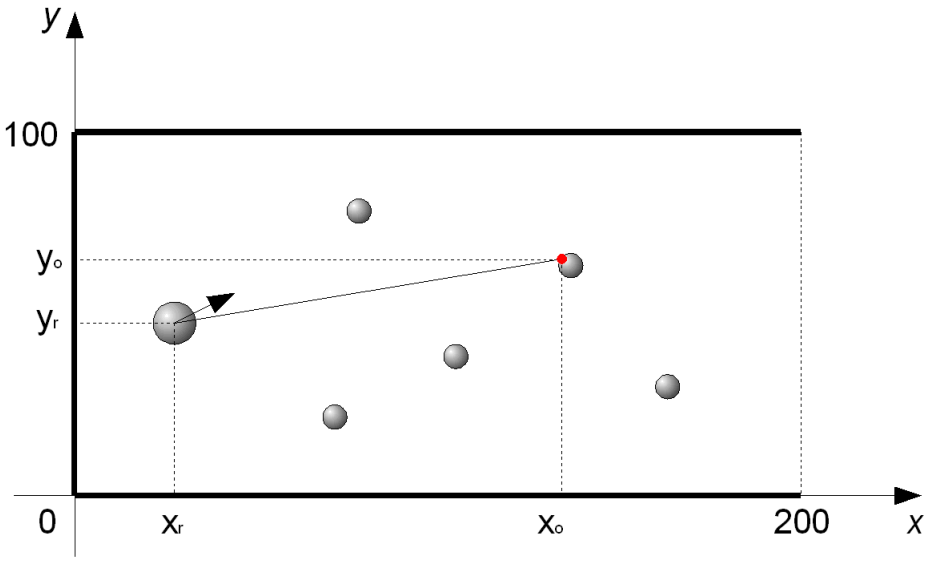


Figura 3.5: Exemplo de interseção entre o sensor 3 ($\alpha_3^\star = -15^o$) e um dos objetos do ambiente.

da Terra, e este valor teria de passar por algum tipo de pré-processamento que o transformasse no ângulo $\phi^{(t)}$.

3.2.2 Movimentação

O robô real possui um sistema de rodas e aceleração similar ao utilizado por (MORATORI, 2006), o qual, por sua vez, é baseado no modelo usado por (KONG; KOSKO, 1992).

Neste sistema, o robô possui duas rodas traseiras, responsáveis pela tração, e uma roda dianteira que pode se mover em torno de um eixo, responsável por direcionar o robô. Este deve, então, a cada instante, ajustar o ângulo da roda dianteira em relação à parte frontal e determinar a tração nas rodas traseiras, de modo a definir

o quanto e em qual direção o robô se movimentará.

Para simular o comportamento deste sistema, podemos recorrer às equações utilizadas por (KONG; KOSKO, 1992), quais sejam:

$$\begin{aligned} \phi^{(t+1)} &= \phi^{(t)} + \theta^{(t)}, \\ x_r^{(t+1)} &= x_r^{(t)} + \Delta^{(t)} \cos \phi^{(t+1)}, \\ y_r^{(t+1)} &= y_r^{(t)} + \Delta^{(t)} \sin \phi^{(t+1)}, \end{aligned}$$

onde $\theta^{(t)}$ e $\Delta^{(t)}$ são, respectivamente, o ângulo da roda dianteira e a velocidade do robô no instante t, ambos calculados pelo controlador do robô. Num robô real, a velocidade num trecho do trajeto seria resultado da tração das rodas traseiras.

3.2.3 Módulo Decisor

Como dito anteriormente, o módulo decisor é o responsável por tomar as decisões quanto à movimentação do robô, baseando-se, para isso, na leitura dos sensores. Assim, o módulo decisor é o que contém o controlador, parte principal do estudo deste trabalho. Além disso, o módulo decisor inclui, ainda, um módulo pré-processador, responsável por receber as informações dos sensores, tratá-las e repassar ao módulo controlador informações mais relevantes. A Figura 3.6 mostra um esquema do funcionamento do módulo decisor.

3.2.3.1 Pré-processamento

O módulo decisor recebe como entradas os valores lidos pelos sensores, quais sejam:

- $d_{s_1}^{(t)}$, distância alcançada pelo sensor de distância direcionado com ângulo $\alpha_1^\star$

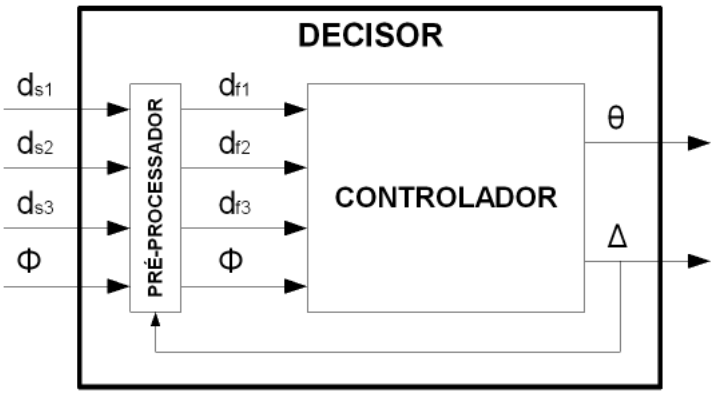


Figura 3.6: Módulo decisor, composto por pré-processador e módulo controlador.

em relação à parte frontal do robô;

- $d_{s_2}^{(t)}$, distância alcançada pelo sensor de distância direcionado com ângulo $\alpha_2^\star$ em relação à parte frontal do robô;
- $d_{s_3}^{(t)}$, distância alcançada pelo sensor de distância direcionado com ângulo $\alpha_3^\star$ em relação à parte frontal do robô; e
- $\phi^{(t)}$, ângulo do robô em relação à horizontal do ambiente.

Consideramos, no entanto, que podemos utilizar algum processamento, além do já requerido pelo módulo controlador, para que este receba outras entradas, mais relevantes para a tomada de decisão. O módulo pré-processador é o responsável por esta tarefa.

Tendo em vista que os obstáculos e as paredes têm posições fixas durante toda a simulação, podemos armazenar em memória os pontos já conhecidos pelos sensores. Os pontos são armazenados através de coordenadas polares, ou seja, a memória contém diversos pares (d, α), onde d é a distância do ponto ao centro do robô e α

é o ângulo que a parte frontal deste faz com a linha que liga seu centro ao ponto em questão. São armazenados, ainda, os valores de $\phi^{(t)}$ e $\Delta^{(t)}$ a cada instante, para serem utilizados na atualização dos pontos armazenados.

No instante $t = 0$, os pontos $\left(d_{s_i}^{(t)}, \alpha_i^{\star}\right)$ $(i = 1, 2, 3)$ são todos inseridos em memória.

Já para os instantes $t > 0$, o primeiro passo realizado pelo pré-processador é a atualização dos pontos armazenados, que deve refletir as mudanças nas posições dos pontos em relação ao robô, já que este se moveu. Para isso, $\left(d_{m_j}^{(t-1)}, \alpha_{m_j}^{(t-1)}\right)$, o j-ésimo ponto armazenado em memória no instante $t-1$ $(j = 1, \ldots, n_m^{(t-1)}$, onde $n_m^{(t-1)}$ é o número de pontos armazenados no instante $t-1)$, é primeiramente atualizado quanto à diferença entre $\phi^{(t)}$ e $\phi^{(t-1)}$, ou seja,

$$\begin{aligned} d_{m_j}^{(+)} &= d_{m_j}^{(t-1)}, \\ \alpha_{m_j}^{(+)} &= \alpha_{m_j}^{(t-1)} - (\phi^{(t)} - \phi^{(t-1)}). \end{aligned}$$

Após isto, este ponto é transformado para coordenadas cartesianas, isto é,

$$\begin{aligned} x_{m_j}^{(+)} &= d_{m_j}^{(+)} \cos \alpha_{m_j}^{(+)}, \\ y_{m_j}^{(+)} &= d_{m_j}^{(+)} \sin \alpha_{m_j}^{(+)}. \end{aligned}$$

É interessante notar que $\left(x_{m_j}^{(+)}, y_{m_j}^{(+)}\right)$ são coordenadas relativas ao centro e à parte frontal do robô. Assim, o novo posicionamento, para o instante t, é calculado da seguinte forma:

$$\begin{aligned} x_{m_j}^{(t)} &= x_{m_j}^{(+)} - \Delta^{(t-1)}, \\ y_{m_j}^{(t)} &= y_{m_j}^{(+)}. \end{aligned}$$

Um novo ponto $\left(d_{s_i}^{(t)}, \alpha_i^{\star}\right)$ $(i = 1, 2, 3)$ só é armazenado em memória se esta inserção representar alguma relevância. Isto é feito calculando-se as coordenadas cartesianas relativas a este ponto e, depois, calculando-se a distância a cada um dos pontos

já em memória. Se, para todos os pontos, a distância for maior que 3 unidades, consideramos que o ponto traz informação relevante sobre o ambiente e o inserimos em memória.

Por fim, transformamos novamente todos os pontos, desta vez das coordenadas cartesianas $\left(x_{m_j}^{(t)}, y_{m_j}^{(t)}\right)$ para as coordenadas polares $\left(d_{m_j}^{(t)}, \alpha_{m_j}^{(t)}\right)$.

3.2.3.2 Entradas do Controlador

Com a lista de pontos atualizada, refletindo o que é conhecido do ambiente pelo robô no instante t, podemos extrair informações mais relevantes para a tomada de decisão do módulo controlador. Estas informações, que servirão de entradas para o módulo controlador, são as seguintes:

- $d_{f_1}^{(t)}$, distância mínima dentre os pontos conhecidos pelo robô que estão localizados entre 15º e 105º em relação à parte frontal, isto é,

$$d_{f_1}^{(t)} = \min_{15^o \leq \alpha_{m_j}^{(t)} \leq 105^o} \left\{ d_{m_j}^{(t)} \right\};$$

- $d_{f_2}^{(t)}$, distância mínima dentre os pontos conhecidos pelo robô que estão localizados entre -15^o e 15º em relação à parte frontal;
- $d_{f_3}^{(t)}$, distância mínima dentre os pontos conhecidos pelo robô que estão localizados entre -105^o e -15^o em relação à parte frontal; e
- $\phi^{(t)}$, ângulo do robô em relação à horizontal do ambiente.

A Figura 3.7 exemplifica o posicionamento do robô, os pontos do ambiente conhecidos por este e os pontos mais próximos em cada faixa.

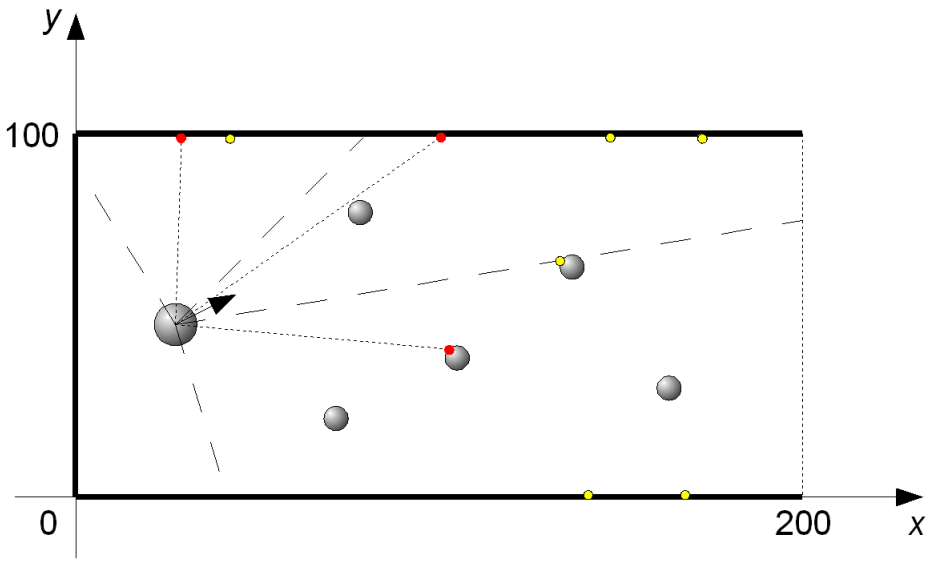


Figura 3.7: Exemplo de pontos do ambiente conhecidos pelo robô e pontos mais próximos em cada faixa. As faixas estão marcadas pelas linhas tracejadas. Em cada uma delas podemos ver um ponto vermelho, correspondente ao ponto mais próximo naquela faixa, tendo uma linha pontilhada ligando-o ao robô, indicando a distância utilizada como entrada do controlador. Os pontos em amarelo correspondem aos demais pontos conhecidos pelo robô.

O uso das distâncias $d_{f_1}^{(t)}$, $d_{f_2}^{(t)}$ e $d_{f_3}^{(t)}$ tem por objetivo informar o quão perto está o objeto mais próximo conhecido à esquerda, à frente e à direita do robô, respectivamente. Já o ângulo $\phi^{(t)}$ indica a direção para a qual o robô está se dirigindo naquele instante.

3.2.3.3 Saídas

O módulo decisor fornece como saída a decisão tomada pelo módulo controlador, com os valores de $\theta^{(t)}$ e $\Delta^{(t)}$ que devem ser utilizados para a movimentação do robô (Seção 3.2.2).

É interessante notar que o valor de $\Delta^{(t)}$ é enviado, também, ao módulo pré-processador, de modo a ser utilizado na atualização dos pontos conhecidos no próximo instante de tempo $t + 1$.

4 CONSTRUÇÃO DE CONTROLADORES

Com a definição de novas entradas (Seção 3.2.3.1), as regras utilizadas no sistema nebuloso do antigo controlador (MORATORI, 2006) não podem ser utilizadas para a construção do novo controlador. Isto ocorre porque não é possível fazer um mapeamento de entradas do antigo para o novo controlador: além de não haver uma entrada correspondente à y_r do antigo controlador, há grande diferença entre a medida de distância do antigo controlador e as distâncias d_f do novo controlador.

Assim, neste capítulo investigamos modelos e técnicas para a construção de novos controladores, utilizando as entradas definidas neste trabalho.

4.1 Construção Manual

O primeiro método que podemos utilizar para construir um novo controlador é o mesmo utilizado em (MORATORI, 2006): contruir manualmente um sistema nebuloso do tipo Mamdani (Seção 2.1.3). Dentre os modelos a serem testados neste trabalho, julgamos que este é o mais adequado a ser construído manualmente, já que as regras e os conjuntos são mais facilmente definidos através de suas descrições

quase textuais.

O sistema nebuloso do tipo Mamdani construído neste trabalho tem como variáveis de entrada e de saída, respectivamente, as entradas e saídas do controlador, descritas na Seção 3.2.3.2. Assim, os valores das entradas do controlador são repassados diretamente como valores das variáveis de entrada do sistema, bem como os valores de suas variáveis de saída, para as saídas do controlador.

Tanto os parâmetros dos conjuntos das variáveis quanto as regras que regem o sistema foram determinados a partir de testes preliminares. Isto é, os parâmetros e as regras foram sendo alterados com base na avaliação manual do desempenho do robô em alguns cenários aleatórios, até que aqueles atingissem uma configuração que trouxesse resultados satisfatórios.

O domínio $[0, 500]$, igual para as três variáveis de distância d_{f_1}, d_{f_2} e d_{f_3}, foi dividido em 4 conjuntos nebulosos, também iguais para tais variáveis: "Muito Perto" (MP), "Perto" (Pe), "Médio" (Me) e "Longe" (Lo). Os parâmetros dos conjuntos podem ser vistos na Tabela 4.1, enquanto a representação gráfica pode ser vista na Figura 4.1.

Tabela 4.1: Parâmetros dos conjuntos nebulosos das variáveis de distância d_{f_1}, d_{f_2} e d_{f_3}.

Conjunto	**Formato**	**Parâmetros**
Muito Perto	Trapézio	$[0, 0, 10, 15]$
Perto	Triângulo	$[10, 20, 30]$
Médio	Triângulo	$[20, 50, 75]$
Longe	Trapézio	$[50, 75, 500, 500]$

Já para a variável de ângulo do robô ϕ, de domínio $[-180, 180]$, temos 5 conjuntos nebulosos, quais sejam: "Baixo-Trás" (BT), "Baixo" (Ba), "Frente" (Fr), "Cima" (Ci) e "Cima-Trás" (CT). Seus parâmetros e seus gráficos podem ser vistos, respectiva-

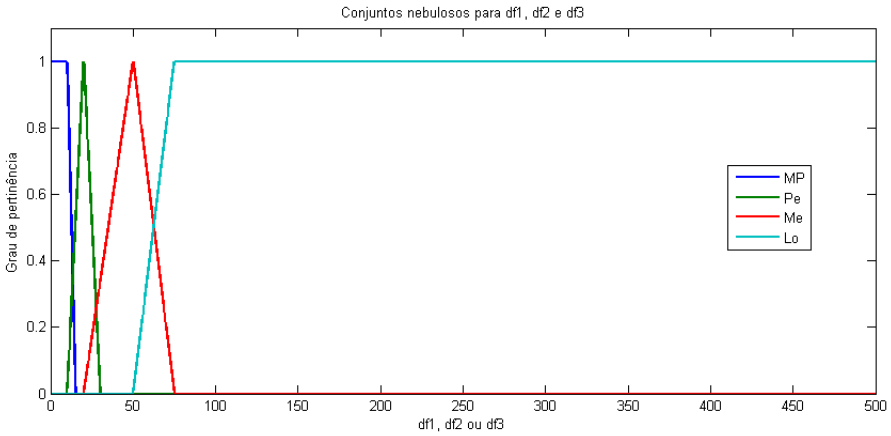


Figura 4.1: Funções de pertinência dos conjuntos nebulosos para d_{f_1}, d_{f_2} e d_{f_3}.

mente, na Tabela 4.2 e na Figura 4.2.

Tabela 4.2: Parâmetros dos conjuntos nebulosos da variável de ângulo do robô ϕ.

Conjunto	**Formato**	**Parâmetros**
Baixo-Trás	Triângulo	$[-210, -180, -120]$
Baixo	Triângulo	$[-180, -90, -0]$
Frente	Triângulo	$[-60, 0, 60]$
Cima	Triângulo	$[0, 90, 180]$
Cima-Trás	Triângulo	$[120, 180, 210]$

A variável de saída de ângulo da roda θ teve seu domínio, $[-30, 30]$, dividido em 7 conjuntos nebulosos: "Muito Direita" (MD), "Direita" (Di), "Pouco Direita" (PD), "Zero" (Ze), "Pouco Esquerda" (PE), "Esquerda" (Es) e "Muito Esquerda" (ME). A Tabela 4.3 apresenta os parâmetros destes conjuntos, enquanto a Figura 4.3, o gráfico dos mesmos.

Por fim, o domínio da variável de velocidade Δ, $[0, 10]$, foi dividido em 5 conjuntos: "Muito Pequena" (MP), "Pequena" (Pe), "Média" (Me), "Grande" (Gr) e "Muito

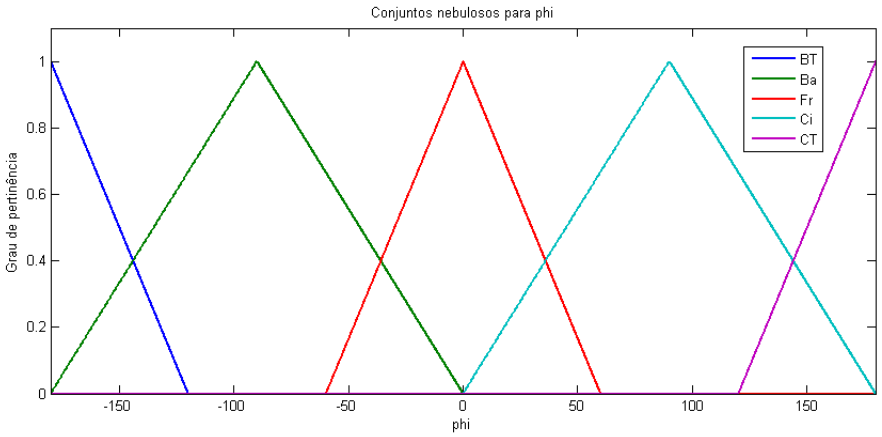


Figura 4.2: Funções de pertinência dos conjuntos nebulosos para ϕ.

Tabela 4.3: Parâmetros dos conjuntos nebulosos da variável de ângulo da roda θ.

Conjunto	**Formato**	**Parâmetros**
Muito Direita	Trapézio	$[-30; -30; -20; -12{,}5]$
Direita	Triângulo	$[-17{,}5; -10; -5]$
Pouco Direita	Triângulo	$[-10; -5; 0]$
Zero	Triângulo	$[-5; 0; 5]$
Pouco Esquerda	Triângulo	$[0; 5; 10]$
Esquerda	Triângulo	$[5; 10; 17{,}5]$
Muito Esquerda	Trapézio	$[12{,}5; 20; 30; 30]$

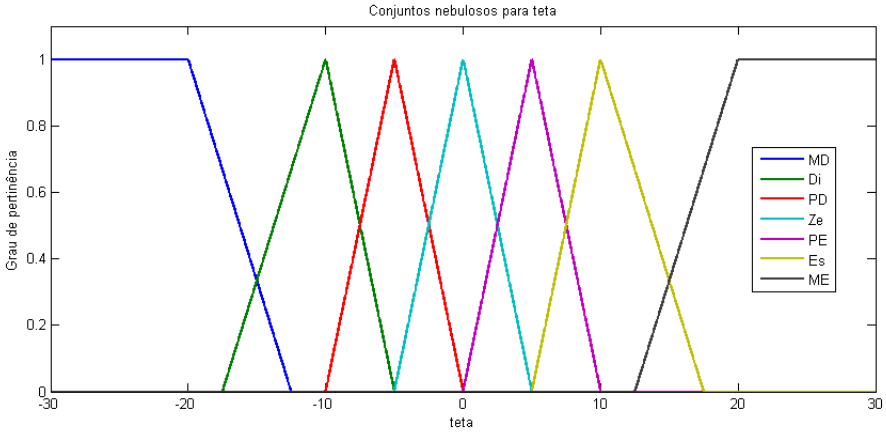


Figura 4.3: Funções de pertinência dos conjuntos nebulosos para θ.

Grande" (MG). Na Tabela 4.4 podem ser vistos os parâmetros dos conjuntos nebulosos para esta variável; a Figura 4.4, por sua vez, apresenta a representação gráfica destes.

Tabela 4.4: Parâmetros dos conjuntos nebulosos da variável de velocidade Δ.

Conjunto	**Formato**	**Parâmetros**
Muito Pequena	Trapézio	$[0; 0; 0{,}1; 0{,}5]$
Pequena	Triângulo	$[0; 0{,}5; 1]$
Média	Triângulo	$[0{,}5; 1{,}3; 2]$
Grange	Triângulo	$[1{,}3; 3; 5]$
Muito Grande	Trapézio	$[4; 5; 10; 10]$

Além das variáveis e de seus conjuntos, o sistema nebuloso construído é composto por 200 regras, divididas em 3 grupos: "frente" (64 regras), "lados" (128 regras) e "trás" (8 regras). A seguir, descrevemos estes 3 grupos e apresentamos as regras através de tabelas.

As tabelas 4.5 a 4.16 possuem o mesmo formato, fixando as entradas d_{f_2} e ϕ e

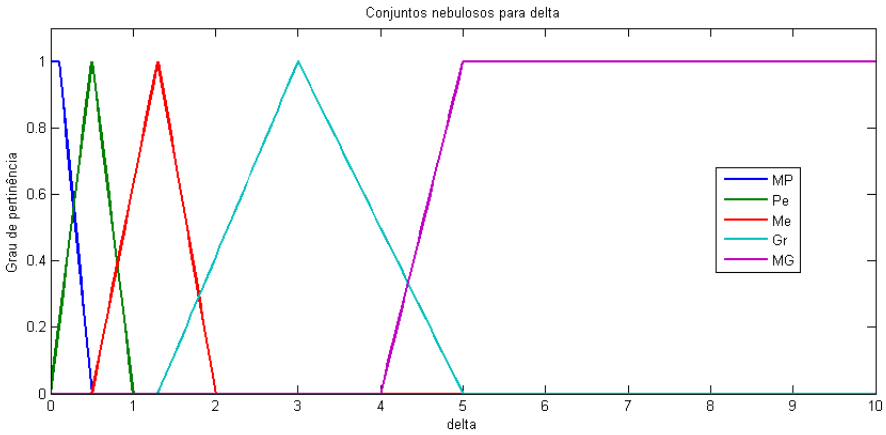


Figura 4.4: Funções de pertinência dos conjuntos nebulosos para Δ.

indicando as saídas θ e Δ a partir das diferentes entradas para d_{f_1} e d_{f_3}. Assim, na Tabela 4.11, por exemplo, uma das regras exibidas é "se (d_{f_1} é Lo) e (d_{f_2} é Me) e (d_{f_3} é Pe) e (ϕ é Ba) então (θ é Es) e (Δ é Gr)". Todas as regras criadas seguem este mesmo formato, isto é, tanto nos antecedentes quanto nos consequentes as cláusulas estão ligadas pelo operador lógico "e".

O grupo de regras "frente" compreende 64 regras para as quais temos o termo "ϕ é Fr". Ou seja, são as regras que tratam do caso em que o robô tem um ângulo em relação à horizontal nulo ou bem pequeno, de modo que está se encaminhando para seu objetivo de alcançar o lado direito do corredor.

De modo geral, parte dessas regras faz o robô reagir à presença de obstáculos, diminuindo a velocidade e contornando-os, enquanto outra parte aumenta a velocidade e mantém o ângulo do robô perto de zero quando não há obstáculos nas proximidades.

As tabelas 4.5, 4.6, 4.7 e 4.8 contêm os consequentes das regras deste grupo, com a variação dos termos para a variável d_{f_2}.

Tabela 4.5: Consequentes das regras onde d_{f_2} é MP e ϕ é Fr.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	ME, MP	Di, MP	Di, MP	Di, MP
Pe	Es, MP	Es, MP	Di, MP	Di, MP
Me	Es, MP	Es, MP	Es, MP	Di, MP
Lo	Es, MP	Es, MP	Es, MP	Es, MP

Tabela 4.6: Consequentes das regras onde d_{f_2} é Pe e ϕ é Fr.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	Es, MP	Di, Pe	Di, Pe	Di, Pe
Pe	Es, Pe	Es, Pe	Di, Pe	PD, Pe
Me	Es, Pe	Es, Pe	PE, Pe	PD, Pe
Lo	Es, Pe	PE, Pe	PE, Pe	PE, Pe

Tabela 4.7: Consequentes das regras onde d_{f_2} é Me e ϕ é Fr.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	Ze, Me	Di, Me	Di, Me	Di, Me
Pe	Es, Me	Ze, Gr	Ze, Gr	Ze, Gr
Me	Es, Me	Ze, Gr	Ze, Gr	Ze, Gr
Lo	Es, Me	Ze, Gr	Ze, Gr	Ze, MG

Tabela 4.8: Consequentes das regras onde d_{f_2} é Lo e ϕ é Fr.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	Ze, Me	Di, Me	Di, Me	Di, Me
Pe	Es, Me	Ze, Gr	Ze, Gr	Ze, Gr
Me	Es, Me	Ze, Gr	Ze, MG	Ze, MG
Lo	Es, Me	Ze, Gr	Ze, MG	Ze, MG

O grupo de regras "lados" é composto por 128 regras: 64 regras para as quais temos o termo "ϕ é Ba" e 64 regras nas quais está presente o termo "ϕ é Ci".

Assim como as regras do grupo "frente", as regras do grupo "lados" tentam desviar o robô dos obstáculos e levá-lo o mais rápido possível a seu objetivo. Porém, estas regras ainda apresentam o diferencial de tentar corrigir a trajetória do robô, fazendo-o girar à esquerda quando está direcionado para baixo e à direita quando está direcionado para cima.

As tabelas 4.9, 4.10, 4.11 e 4.12 contêm os consequentes das regras deste grupo para as quais temos "ϕ é Ba", com a variação dos termos para a variável d_{f_2}. Já as tabelas 4.13, 4.14, 4.15 e 4.16 apresentam as mesmas informações, porém relativas às regras para as quais temos "ϕ é Ci".

É interessante notar que, neste grupo, os consequentes em relação à variável θ são simétricos. Isto é, para iguais conjuntos de d_{f_1}, d_{f_2} e d_{f_3}, o consequente da regra onde "ϕ é Ba" é o oposto daquele da regra onde "ϕ é Ci".

Por fim, o grupo "trás" possui 8 regras: 4 para os casos com o termo "ϕ é BT" e 4 com o termo "ϕ é CT". O objetivo destas regras é simplesmente fazer com que o robô dê meia-volta e siga a direção correta, tendo mínima preocupação com obstáculos.

Tabela 4.9: Consequentes das regras onde d_{f_2} é MP e ϕ é Ba.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	ME, MP	MD, MP	MD, MP	MD, MP
Pe	ME, MP	ME, MP	MD, MP	MD, MP
Me	Es, MP	Es, MP	Es, MP	Es, MP
Lo	Es, MP	Es, MP	Es, MP	Es, MP

Tabela 4.10: Consequentes das regras onde d_{f_2} é Pe e ϕ é Ba.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	ME, MP	Di, Pe	Di, Pe	Di, Pe
Pe	Es, Pe	Es, Pe	Es, Pe	Es, Pe
Me	Es, Pe	Es, Pe	PE, Pe	PE, Pe
Lo	Es, Pe	PE, Pe	PE, Pe	PE, Pe

Tabela 4.11: Consequentes das regras onde d_{f_2} é Me e ϕ é Ba.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	Ze, Me	Ze, Me	Ze, Me	Ze, Me
Pe	PE, Me	PE, Me	PE, Me	PE, Me
Me	Es, Me	Es, Gr	Es, Gr	Es, Gr
Lo	Es, Me	Es, Gr	Es, Gr	Es, Gr

Tabela 4.12: Consequentes das regras onde d_{f_2} é Lo e ϕ é Ba.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	Ze, Me	Ze, Me	Ze, Me	Ze, Me
Pe	PE, Me	PE, Me	PE, Me	PE, Me
Me	Es, Me	Es, Gr	Es, Gr	Es, Gr
Lo	Es, Me	Es, Gr	Es, Gr	Es, MG

Tabela 4.13: Consequentes das regras onde d_{f_2} é MP e ϕ é Ci.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	MD, MP	MD, MP	Di, MP	Di, MP
Pe	ME, MP	MD, MP	Di, MP	Di, MP
Me	ME, MP	ME, MP	Di, MP	Di, MP
Lo	ME, MP	ME, MP	Di, MP	Di, MP

Tabela 4.14: Consequentes das regras onde d_{f_2} é Pe e ϕ é Ci.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	MD, MP	Di, Pe	Di, Pe	Di, Pe
Pe	Es, Pe	Di, Pe	Di, Pe	PD, Pe
Me	Es, Pe	Di, Pe	PD, Pe	PD, Pe
Lo	Es, Pe	Di, Pe	PD, Pe	PD, Pe

Tabela 4.15: Consequentes das regras onde d_{f_2} é Me e ϕ é Ci.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	Ze, Me	PD, Me	Di, Me	Di, Me
Pe	Ze, Me	PD, Me	Di, Gr	Di, Gr
Me	Ze, Me	PD, Me	Di, Gr	Di, Gr
Lo	Ze, Me	PD, Me	Di, Gr	Di, Gr

Tabela 4.16: Consequentes das regras onde d_{f_2} é Lo e ϕ é Ci.

	d_{f_3} é			
d_{f_1} é	MP	Pe	Me	Lo
MP	Ze, Me	PD, Me	Di, Me	Di, Me
Pe	Ze, Me	PD, Me	Di, Gr	Di, Gr
Me	Ze, Me	PD, Me	Di, Gr	Di, Gr
Lo	Ze, Me	PD, Me	Di, Gr	Di, MG

As regras para "ϕ é BT" verificam apenas a variável d_{f_1}, de modo a corrigir a rota do robô fazendo-o virar à esquerda. De forma análoga, as regras para "ϕ é CT" examinam apenas a variável d_{f_3}. As regras podem ser vistas nas tabelas 4.17 e 4.18.

Tabela 4.17: Regras onde ϕ é BT.

Se		então	
d_{f_1} é	ϕ é	θ é	Δ é
MP	BT	MD	MP
Pe	BT	Di	Pe
Me	BT	ME	Me
Lo	BT	ME	Me

Tabela 4.18: Regras onde ϕ é CT.

Se		então	
d_{f_3} é	ϕ é	θ é	Δ é
MP	CT	ME	MP
Pe	CT	Es	Pe
Me	CT	Es	Me
Lo	CT	Es	Me

4.2 Construção a Partir de Exemplos

Além do já discutido sistema nebuloso do tipo Mamdani, podemos utilizar outros modelos e técnicas para a construção de controladores que resolvam o problema proposto. Mais especificamente, trabalhamos com os modelos e técnicas apresentados no Capítulo 2, quais sejam:

- Sistema nebuloso do tipo Sugeno, treinado através de Gradiente Decrescente;
- Rede neural artificial do tipo MLP, treinada através de Gradiente Decrescente (*Back-propagation*);
- Rede neural artificial do tipo RBF, treinada através de Gradiente Decrescente; e
- Máquina de vetores de suporte para regressão, com definição de parâmetros a partir da resolução do problema de programação quadrática.

Em todos estes modelos, as técnicas para a definição de seus parâmetros utilizam conjuntos de exemplos, os quais contêm exemplos de entradas para o controlador e as saídas esperadas para estas. Na seção a seguir, descrevemos os métodos utilizados para a obteção destes conjuntos de exemplos.

4.2.1 Obtenção de Exemplos

Uma das motivações para utilizar estes outros modelos é tentar aproveitar o conhecimento anterior já disponível no antigo controlador de (MORATORI, 2006), extraindo-o e representando-o justamente em um conjunto de exemplos.

Outra motivação é o formato da superfície de saída do novo controlador, por ter sido construído a partir de um sistema nebuloso do tipo Mamdani. Apesar de na maioria dos casos a resposta dada pelo sistema ser satisfatória, para algumas entradas bem próximas as saídas podem ser bem diferentes, isto é, a superfície de saída possui algumas variações bruscas. Esta situação pode levar a casos em que o controlador alterne entre saídas bem distintas, fazendo com que o robô fique praticamente estagnado e não reaja de maneira adequada a obstáculos.

Um exemplo deste caso pode ser visto na simulação apresentada na Figura 4.5, em que o robô não reage corretamente à presença do obstáculo, pois as regras que estão sendo ativadas produzem como resposta final um valor próximo a zero para θ. Quando o robô fica ainda mais próximo do obstáculo, outras regras são ativadas e o robô inicia sua reação. Contudo, a distância já é pequena o suficiente para que o robô não evite a colisão.

Podemos, então, tentar extrair um conjunto de exemplos deste novo controlador e utilizá-lo para o treinamento de outros modelos, com a intenção de que estes tenham em sua saída uma superfície semelhante à daquele, porém com variações mais suaves.

Assim, temos disponíveis dois conjuntos de exemplos, um criado a partir do antigo controlador, e outro criado a partir do novo controlador. As seções a seguir detalham a criação destes conjuntos.

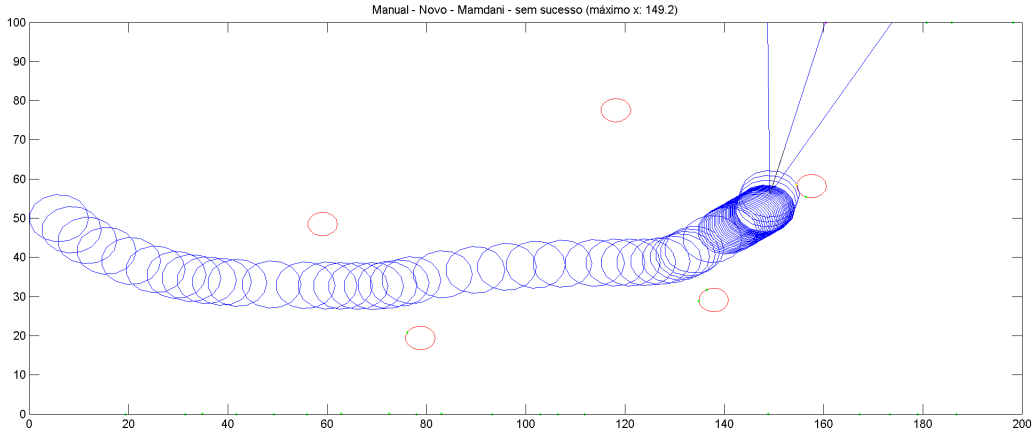


Figura 4.5: Exemplo de colisão do robô controlado pelo novo Sistema Nebuloso Mamdani.

4.2.1.1 Antigo Controlador

Para o antigo controlador, temos a dificuldade de que este não possui as mesmas entradas do novo controlador. Com isso, não é possível fazer a correspondência entre entradas do antigo e do novo controlador.

A solução utilizada para gerar entradas para o novo controlador a partir de entradas do antigo é realizar diversas simulações nais quais o robô possui sensores tanto do antigo quanto do novo controlador. Assim, para cada iteração da simulação, tomamos a decisão baseados nas entradas dos sensores antigos e utilizando o controlador correspondente, porém armazenamos em nosso conjunto de amostras as entradas dos novos sensores e as saídas geradas.

4.2.1.2 Novo Controlador

Para o novo controlador, basta termos um conjunto de entradas e apresentá-las àquele para gerarmos as saídas. Para obter um bom conjunto de entradas, selecionamos valores representativos para cada uma e combinamo-os em todas as possibilidades.

5 IMPLEMENTAÇÃO, TESTES E RESULTADOS

Neste capítulo, descrevemos a implementação dos modelos, detalhamos os métodos e parâmetros utilizados durante os treinamentos, mostramos como foram realizados os testes, apresentamos os resultados e realizamos uma análise destes.

5.1 Implementação

Para realizarmos os testes das abordagens propostas neste trabalho, utilizamos a plataforma MATLAB [1] em sua versão 7.0, em um computador com processador *Core 2 Duo* da Intel de 1,87 GHz e com 2 GB de memória RAM.

Fizemos uso do pacote de Lógica Nebulosa que acompanha a plataforma, além de implementarmos o algoritmo de Gradiente Decrescente mostrado na Seção 2.1.4. Utilizamos, também, o pacote de Redes Neurais Artificiais disponível no ambiente nos treinamentos de redes MLP, bem como adaptamos algumas funções do pacote para o treinamento de redes RBF através de Gradiente Decrescente. Além disso, utilizamos o pacote de Máquinas de Vetores de Suporte de (CANU; GRANDVALET;

[1] http://www.mathworks.com/

RAKOTOMAMONJY, 2003).

Foi criado um módulo de simulação de robôs, sobre o qual implementamos o ambiente, os sensores e a movimentação do robô, como descrito no Capítulo 3.

Utilizando o pacote de Lógica Nebulosa, criamos manualmente um sistema nebuloso para o novo controlador pelas regras e conjuntos mostrados na Seção 4.1, além de montarmos o antigo sistema nebuloso descrito em (MORATORI, 2006) para fins de comparação.

5.2 Treinamentos

Visando a comparar os diferentes modelos a serem utilizados no trabalho e experimentar suas capacidades de generalização, optamos por construir dois conjuntos de dados de treinamento, com diferentes quantidades de amostras. Além disso, com o objetivo de auxiliar os treinamentos, foi consutruído, também, um conjunto de dados de validação.

As entradas dos conjuntos de dados correspondem a todas as combinações dos valores mostrados na Tabela 5.1. Estes valores foram escolhidos de acordo com o descrito na Seção 4.2.1.2, isto é, com a intenção de termos valores representativos para cada uma das 4 entradas (d_1, d_2, d_3 e ϕ). Deste modo, o Conjunto de Dados 1 (CD1) possui 1512 amostras ($6 \times 6 \times 6 \times 7$), o Conjunto de Dados 2 (CD2), 3087 amostras ($7 \times 7 \times 7 \times 9$), e o Conjunto de Dados de Validação (CDV), 135 amostras ($3 \times 3 \times 3 \times 5$).

A partir das entradas destes conjuntos de dados, geramos suas saídas utilizando os métodos de obtenção descritos na Seção 4.2, ou seja, a partir do antigo e do novo controlador. Assim, combinando os dois conjuntos de dados (CD1 e CD2) com os

Tabela 5.1: Valores base de entrada para criação dos conjuntos de dados.

Entrada	d_1, d_2 e d_3	ϕ
CD1	10, 20, 30, 50, 80, 250	-135, -90, -45, 0, 45, 90, 135
CD2	5, 10, 20, 30, 50, 80, 250	-135, -90, -60, -30, 0, 30, 60, 90, 135
CDV	15, 40, 100	-120, -45, 0, 45, 120

dois métodos de obtenção (a partir do antigo e do novo controlador), geramos quatro conjuntos de treinamento diferentes, a serem utilizados na definição dos parâmetros dos modelos.

Para cada modelo a ser treinado, utilizamos os mesmos parâmetros de treinamento tanto para os conjuntos de dados gerados a partir do antigo controlador quanto para os gerados a partir do novo controlador. Os parâmetros específicos de cada modelo são descritos a seguir.

Para o treinamento do Sistema Nebuloso Sugeno utilizando Gradiente Decrescente foram realizados 10 treinamentos para cada conjunto de dados, com número máximo de 500 iterações. Como não sabíamos que valor deveríamos ter idealmente para o erro quadrático médio ao final de um treinamento, em metade deles o erro quadrático médio desejado era de 10^{-2}, enquanto na outra metade era de 10^{-3}. Em ambos os casos fizemos uso de validação, com número máximo de falhas igual a 10. Além disso, a arquitetura da rede correspondente ao sistema era tal que a função dos consequentes utilizada foi a tangente hiperbólica, gerando saídas entre -1 e $+1$. Os parâmetros dos antecedentes das regras foram ajustados de modo a ter comportamento idêntico ao do sistema Mamdani.

No treinamento da Rede Neural MLP através de *Back-propagation* realizamos um total de 15 treinamentos para cada conjunto de dados, com número máximo de 1500 iterações, erro quadrático médio desejado de 10^{-2} e número máximo de falhas de

validação igual a 100. Quanto à quantidade de neurônios na camada escondida, a arquitetura das redes foi diferente para cada grupo de 5 treinamentos, de modo que os 3 grupos eram compostos por redes com 200, 250 e 300 neurônios na camada escondida (valores estabelecidos a partir de testes preliminares). Quanto às funções de propagação, foi utilizada a tangente hiperbólica em todas as camadas.

Já para o treinamento da Rede Neural RBF por Gradiente Decrescente foram realizados 10 treinamentos para cada conjunto de dados, com número máximo de 1000 iterações. Em metade deles o erro quadrático médio desejado era de 10^{-2} e o número máximo de falhas por validação foi de 50, enquanto na outra metade os valores utilizados foram 10^{-3} e 100, respectivamente. A função de propagação utilizada na camada de saída foi a tangente hiperbólica, e a camada intermediária era composta por 200 neurônios (valor estabelecido a partir de testes preliminares), cujos parâmetros (centros e larguras de suas gaussinas) foram definidos visando a imitar os antecedentes das regras do sistema Mamdani.

A Tabela 5.2 mostra os parâmetros comuns aos três modelos apresentados acima.

Tabela 5.2: Parâmetros de treinamento comuns aos modelos Sistema Nebuloso Sugeno, Rede Neural MLP e Rede Neural RBF.

Parâmetro	**Valor**
Taxa de *momentum*	0,75
Taxa de aprendizado inicial	0,1
Fator de aumento da taxa de aprendizado	1,05
Fator de diminuição da taxa de aprendizado	0,70

Para a adaptação dos parâmetros da Máquina de Vetores de Suporte, o valor de ϵ utilizado nos 12 treinamentos foi 10^{-2}. Em todos estes, utilizamos a função de *kernel* gaussiana, e os parâmetros sendo as combinações dos valores encontrados na Tabela 5.3.

Tabela 5.3: Parâmetros para Máquina de Vetores de Suporte.

Parâmetro	**Valores**
C	10^{-2}, 10^{-1}, 10^{0}, 10^{1}
σ	0,025, 0,1, 0,5

5.3 Testes e Resultados

Após o treinamento das instâncias de cada modelo, realizamos simulações para avaliar o desempenho de cada uma como controladora do robô do problema.

Para isso, geramos aleatoriamente diversos cenários (configurações de disposição dos obstáculos no ambiente), dentre os quais selecionamos 100 que consideramos serem representativos para uma satisfatória avaliação dos controladores. Para cada cenário, o robô foi colocado em 25 posições iniciais diferentes, correspondentes à combinação dos valores mostrados na Tabela 5.4, de modo que cada controlador passou por 2500 simulações.

Tabela 5.4: Valores iniciais para as simulações de teste.

Variável	**Valores**
y	10, 30, 50, 70, 90
ϕ	-60, -30, 0, 30, 60

Uma vez realizadas as simulações, das quais armazenamos diversas informações (posições do robô em cada instante, situação ao final da simulação etc.), devemos escolher métricas que nos possibilitem comparar os modelos propostos. Assim, foram utilizadas neste trabalho as seguintes métricas:

- Número de simulações com sucesso (o robô atingiu seu objetivo);

- Número de passos realizados pelo robô, para as simulações com sucesso (quão rápido o robô atinge o objetivo);
- Média das somas dos valores absolutos de θ, para as simulações com sucesso (quão suaves são as variações de ângulo do robô); e
- Média dos valores máximos de x_r, para as simulações sem sucesso (quão perto do objetivo o robô chegou).

Os resultados para estas quatro métricas podem ser vistos nas tabelas 5.5, 5.6, 5.7 e 5.8. Em cada tabela, temos uma seção para o antigo controlador Mamdani, outra para o novo controlador deste mesmo tipo, e outras quatro para cada um dos conjuntos de treinamento utilizados, relativos às quatro combinações de conjuntos de dados e métodos de obtenção (como descrito na Seção 5.2). Para cada modelo em cada seção, observamos os testes realizados com as instâncias obtidas para este modelo e calculamos as seguintes estatísticas: máximo (ou mínimo, de acordo com a métrica), média e desvio padrão da métrica a ser analisada.

5.4 Análise

Observando os resultados da Tabela 5.5, correspondente à métrica que podemos considerar a mais importante dentre as quatro, podemos perceber que o novo controlador Mamdani foi superior ao antigo. Além disso, vemos que o modelo Sugeno é o claro vencedor. Apesar de não ser o melhor para o método de obtenção das amostras a partir do antigo controlador, os resultados deste caso são tão inferiores em relação aos do outro método que podem ser desprezados.

Por outro lado, para os conjuntos formados por amostras obtidas a partir do novo controlador, o melhor resultado para o modelo Sugeno proporcionou um aumento

Tabela 5.5: Número de simulações nas quais o robô terminou com sucesso. As colunas "Máximo", "Média" e "Desvio padrão" correspondem a estas estatísticas para os valores da métrica em questão obtidos pelos controladores gerados a partir dos diversos treinamentos de cada tupla (conjunto de dados, método de obtenção, modelo). Os resultados em negrito correspondem aos melhores para cada par de conjunto de dados e método de obtenção, enquanto os resultados sublinhados, ao melhor no geral.

Conjunto de dados	**Método de obtenção**	**Modelo**	**Máximo**	**Média**	**Desvio padrão**
—	Antigo	Mamdani	1937	—	—
—	Novo	Mamdani	2242	—	—
CD1	Antigo	Sugeno	552	**496,9**	23,6
		MLP	552	334,1	218,0
		RBF	486	435,9	33,0
		SVM	**677**	266,9	247,8
CD1	Novo	Sugeno	**2356**	**2260,5**	76,0
		MLP	990	689,7	373,9
		RBF	2205	2135,3	34,4
		SVM	1683	759,9	627,4
CD2	Antigo	Sugeno	574	469,1	88,5
		MLP	755	308,4	293,9
		RBF	**982**	**780,9**	225,4
		SVM	468	203,5	194,3
CD2	Novo	Sugeno	**2351**	**2199,4**	108,7
		MLP	1122	120,3	308,1
		RBF	2165	2024,0	390,7
		SVM	1926	874,5	693,5

Tabela 5.6: Número de passos realizados pelo robô, para as simulações terminadas com sucesso. As colunas "Mínimo", "Média" e "Desvio padrão" correspondem a estas estatísticas para os valores da métrica em questão obtidos pelos controladores gerados a partir dos diversos treinamentos de cada tupla (conjunto de dados, método de obtenção, modelo). Os resultados em negrito correspondem aos melhores para cada par de conjunto de dados e método de obtenção, enquanto os resultados sublinhados, ao melhor no geral.

Conjunto de dados	**Método de obtenção**	**Modelo**	**Mínimo**	**Média**	**Desvio padrão**
—	Antigo	Mamdani	70,2	—	—
—	Novo	Mamdani	59,9	—	—
CD1	Antigo	Sugeno	**34,4**	36,5	1,4
		MLP	65,2	78,2	5,7
		RBF	**34,4**	**34,5**	0,1
		SVM	39,7	64,9	20,7
CD1	Novo	Sugeno	65,4	69,5	2,9
		MLP	93,0	98,3	3,8
		RBF	**61,5**	**62,6**	0,7
		SVM	62,1	100,2	31,1
CD2	Antigo	Sugeno	40,1	**42,9**	3,1
		MLP	**38,6**	88,4	19,3
		RBF	43,7	46,3	1,7
		SVM	44,0	53,1	7,4
CD2	Novo	Sugeno	64,9	69,6	4,0
		MLP	113,4	129,2	11,4
		RBF	**21,5**	**61,2**	14,0
		SVM	66,9	110,4	39,0

Tabela 5.7: Média das somas dos valores absolutos de θ, para as simulações terminadas com sucesso. As colunas "Mínimo", "Média" e "Desvio padrão" correspondem a estas estatísticas para os valores da métrica em questão obtidos pelos controladores gerados a partir dos diversos treinamentos de cada tupla (conjunto de dados, método de obtenção, modelo). Os resultados em negrito correspondem aos melhores para cada par de conjunto de dados e método de obtenção, enquanto os resultados sublinhados, ao melhor no geral.

Conjunto de dados	**Método de obtenção**	**Modelo**	**Mínimo**	**Média**	**Desvio padrão**
—	Antigo	Mamdani	616,6	—	—
—	Novo	Mamdani	175,2	—	—
CD1	Antigo	Sugeno	87,2	94,5	4,9
		MLP	**43,2**	**50,8**	4,0
		RBF	84,6	88,3	3,5
		SVM	44,1	67,2	37,7
CD1	Novo	Sugeno	167,9	202,3	27,4
		MLP	68,3	**80,7**	12,2
		RBF	157,4	161,4	2,3
		SVM	**62,2**	138,8	60,8
CD2	Antigo	Sugeno	78,2	90,1	7,4
		MLP	**32,0**	**50,4**	28,5
		RBF	64,6	73,0	4,2
		SVM	41,1	84,9	36,3
CD2	Novo	Sugeno	169,5	200,5	25,6
		MLP	84,5	**108,1**	15,9
		RBF	**65,2**	151,5	30,4
		SVM	83,8	153,7	42,8

Tabela 5.8: Média dos valores máximos de x_r, para as simulações terminadas sem sucesso. As colunas "Máximo", "Média" e "Desvio padrão" correspondem a estas estatísticas para os valores da métrica em questão obtidos pelos controladores gerados a partir dos diversos treinamentos de cada tupla (conjunto de dados, método de obtenção, modelo). Os resultados em negrito correspondem aos melhores para cada par de conjunto de dados e método de obtenção, enquanto os resultados sublinhados, ao melhor no geral.

Conjunto de dados	**Método de obtenção**	**Modelo**	**Máximo**	**Média**	**Desvio padrão**
—	Antigo	Mamdani	108,7	—	—
—	Novo	Mamdani	95,7	—	—
CD1	Antigo	Sugeno	89,0	77,0	7,5
		MLP	84,6	62,3	34,8
		RBF	85,0	**79,7**	2,4
		SVM	**93,5**	67,7	21,4
CD1	Novo	Sugeno	101,1	92,1	8,3
		MLP	92,8	71,6	27,7
		RBF	100,4	**97,2**	1,8
		SVM	**109,5**	80,3	18,7
CD2	Antigo	Sugeno	90,3	84,6	8,2
		MLP	**94,8**	60,2	37,1
		RBF	91,8	**85,8**	3,6
		SVM	84,9	54,2	23,9
CD2	Novo	Sugeno	103,0	**95,3**	5,9
		MLP	97,4	42,0	30,8
		RBF	99,1	94,6	4,4
		SVM	**115,9**	85,3	18,4

de 5% de simulações bem sucedidas em relação ao próprio novo controlador Mamdani construído manualmente. E, pela média e pelo desvio padrão, boa parte dos outros controladores para o modelo Sugeno obtiveram resultados melhores que o controlador Mamdani.

Ainda para esta tabela, podemos dizer que o modelo RBF obteve resultados interessantes.

Já para a Tabela 5.6, o modelo RBF se saiu melhor no geral. Mesmo nos casos em que não é o melhor do par de conjunto de dados e método de obtenção, seus resultados são comparáveis aos melhores. Também no geral, o modelo Sugeno obteve resultados bem próximos aos do modelo RBF.

Situação semelhante ocorre quando analisamos a Tabela 5.7: os melhores resultados são do modelo MLP, enquanto o modelo SVM obteve resultados próximos na coluna "Mínimo". Na coluna "Média", contudo, os resultados para o modelo SVM não são tão bons quanto seus correspondentes mínimos.

Por fim, pelas médias e desvios padrão exibidos na Tabela 5.8, podemos dizer que o modelo RBF atingiu os melhores resultados para a métrica correspondente.

Levando em conta os resultados apresentados, podemos ver que o novo controlador Mamdani, construído manualmente, é superior ao antigo controlador em vários aspectos.

No que diz respeito a número de simulações terminadas com sucesso, o uso do novo controlador representa uma melhora de cerca de 16% em relação ao antigo. Além disso, o número de passos para antigir o objetivo e os ângulos das rodas utilizados na navegação do robô são, em média, menores, de modo que este atinge

seu objetivo mais rapidamente e com maior suavidade. Além disso, se o robô real for tal que ângulos maiores para as rodas correspondem a maior gasto de energia, o robô controlado pelo novo sistema Mamdani ainda teria a vantagem de economizar mais energia em relação ao antigo, o que pode ser um aspecto importante em alguns casos (HUNG; LIU; CHANG, 2007).

Uma explicação para os melhores resultados está na utilização dos novos sensores e do pré-processador, mais adequados para a tarefa de evitar obstáculos e paredes, em conjunto com a nova base de regras, refinada para esta tarefa.

Porém, ao final deste trabalho escolheríamos, sem dúvida, o modelo Sugeno. Este é, de longe, o modelo mais bem sucedido, com uma melhora de 21% em relação ao antigo controlador, além de melhoras no quesito ângulos das rodas tanto em relação ao antigo controlador quanto ao novo Mamdani.

O número médio de passos para terminar a simulação, no entanto, é ligeiramente superior ao do novo controlador Mamdani. Isto ocorre porque o controlador do modelo Sugeno dá, em média, passos menores que os dos outros controladores, o que claramente auxilia ao desviar de obstáculos, mas que torna necessária a utilização de mais passos para que o robô alcance o objetivo.

É importante lembrar que os conjuntos utilizados nos antecedentes do modelo Sugeno são idênticos ao do novo Mamdani, a partir do qual foram obtidos os dados para treinamentos dos modelos. Isto provavelmente ajudou o método de Gradiente Decrescente a ser extremamente bem sucedido não só em adaptar o Sugeno para dar respostas semelhantes às do novo Mamdani, mas também a alcançar um de nossos objetivos: fazer com que a superfície de saída do modelo fosse mais suave em relação ao modelo Mamdani (Seção 4.2.1). O melhor desempenho no geral do modelo Sugeno em relação aos outros modelos se deve, provavelmente, a todos estes fatores.

Outro modelo promissor é o de rede neural do tipo RBF. Apesar de seus resultados para a métrica número de simulações com sucesso ser inferior aos dos demais modelos, os resultados para as outras métricas mostram que, quando o robô controlado pelo modelo RBF alcança seu objetivo, o faz de maneira mais eficiente, com menor número de passos e menores ângulos de rodas. Resta saber se esta propriedade permaneceria ao refinarmos o modelo visando a aumentar o número de sucessos.

Baseados nas colunas "Máximo" e "Mínimo" das tabelas de resultados, podemos considerar que o modelo SVM apresentou resultados razoáveis. Porém, olhando para as colunas "Média", isto não ocorre. Observando os desvios padrão, uma possível explicação é que, apesar de variarmos os parâmetros do treinamento dentro de um intervalo não muito grande, a variação é suficiente para produzir instâncias bastante diferentes uma das outras. Isto pode ser resolvido com uma melhor seleção dos parâmetros de treinamento deste modelo.

Já o modelo MLP não convergiu para boas instâncias em nenhum dos casos. Isto pode ser um indicativo de que este modelo não é o mais apropriado para este problema, tanto com os parâmetros utilizados neste trabalho quanto com outros possíveis parâmetros. Mesmo na única métrica em que a rede MLP foi o melhor modelo, podemos explicar estes resultados pelo fato de que este modelo quase sempre produz ângulos de saída próximos de zero, de modo que não é bem sucedido em boa parte das tentativas de desvio de obstáculos.

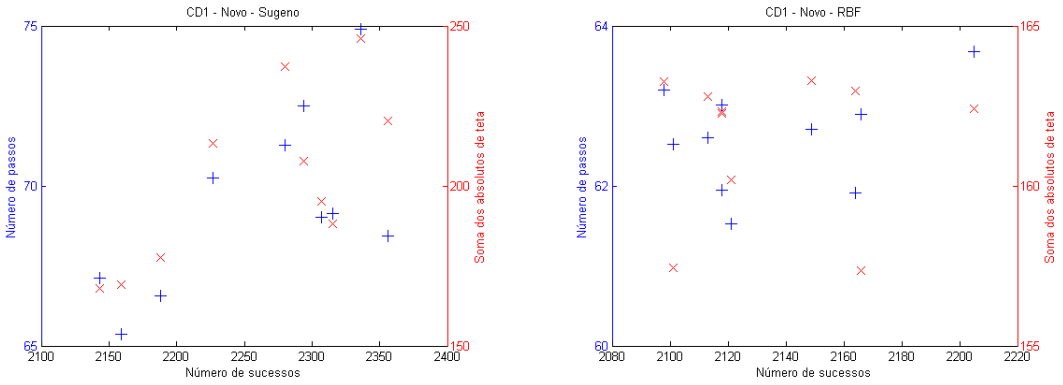
Comparando os modelos Sugeno e RBF com os modelos MLP e SVM, os primeiros foram claramente superiores em relação aos últimos para o problema atacado. Uma explicação bastante intuitiva para isso é que os métodos de treinamento para Sugeno e RBF apenas precisam adaptar os parâmetros de sua última camada, enquanto para MLP e SVM a adaptação dos parâmetros é bem mais complexa.

A comparação entre conjuntos de dados é interessante. Apesar do Conjunto de Dados 2 prover mais amostras (e, consequentemente, mais dados para adaptação dos parâmetros dos modelos) em relação ao Conjunto de Dados 1, os resultados obtidos neste são, em geral, melhores que os obtidos naqueles. Há duas explicações principais para isso. A primeira, mais clara, é que a adaptação dos parâmetros é mais fácil quando há menos amostras. A segunda é que, para o problema em questão, as interpolações realizadas para as amostras do Conjunto de Dados 1 acabam trazendo resultados melhores, talvez porque nem todas as amostras do Conjunto de Dados 2 sejam boas para treinamento.

Apesar do novo controlador Mamdani ser melhor que o antigo controlador, a diferença de resultados entre este e aquele não é tão grande quanto a diferença de resultados alcançados utilizando-se os dois diferentes métodos de obtenção de amostras. Podemos explicar esta situação observando que, no caso das amostras obtidas a partir do novo controlador, estas são extraídas diretamente, pela simples aplicação das entradas sobre o modelo para gerar saídas. Enquanto isso, tivemos de utilizar diversos artifícios para obter amostras a partir do antigo controlador, adaptando os novos sensores a este em simulações, o que pode ter afetado a qualidade dos dados obtidos. Além disso, não há garantia de que os dados obtidos neste último método são compatíveis com os novos sensores, isto é, que haja um mapeamento razoável de entradas do antigo controlador para entradas do novo controlador.

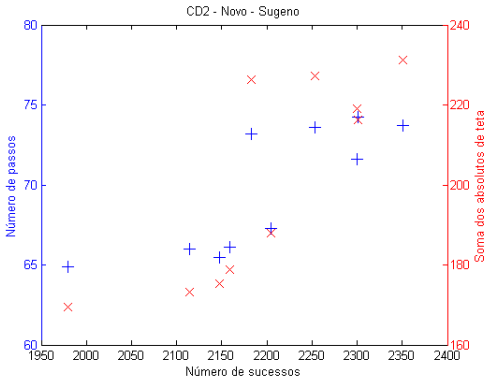
Uma outra comparação interessante que podemos fazer é entre os modelos Sugeno e RBF, levando-se em conta as três métricas relacionadas a sucesso: número de sucessos, número de passos e soma dos valores absolutos de θ. Na Figura 5.1 são exibidas as relações entre a primeira métrica e as outras duas.

Claramente, podemos ver que a correlação entre o número de passos e as outras duas métricas é maior para o modelo Sugeno, sendo quase nula para o modelo RBF. Isto

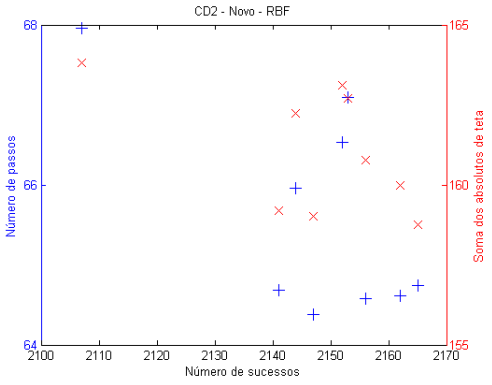


(a)

(b)



(c)



(d)

Figura 5.1: Relações entre a métrica número de sucessos e as métricas número de passos e soma dos valores absolutos de θ. Na figura (d) foram retirados os pontos relativos a um dos controladores, cujo treinamento não convergiu.

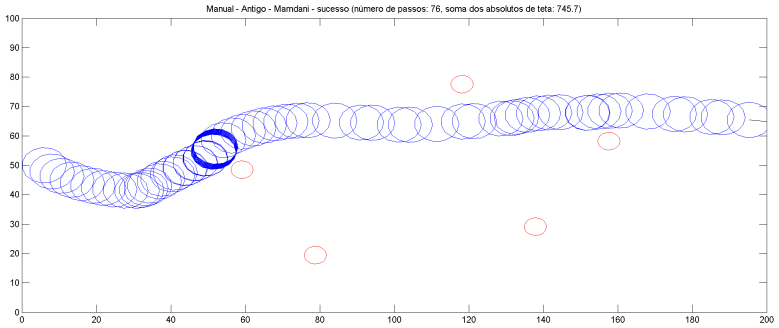
pode ser um indicativo de que o modelo RBF é mais bem sucedido em manter as métricas num certo patamar mesmo quando o refinamos e conseguimos aumentar o número de sucessos. Porém, só poderemos ter certeza disto com mais testes e com resultados em que o número de sucessos para o modelo RBF seja semelhante ao do modelo Sugeno.

Para finalizar a seção de análise, selecionamos seis controladores para comparar seus desempenhos em algumas simulações: os controladores Mamdani construídos manualmente, tanto o antigo quanto o novo; e os melhores controladores (para a métrica número de sucessos) dos modelos Sugeno e RBF, para ambos os conjuntos de dados, com método de obtenção a partir do novo controlador. Informações e métricas para estes controladores podem ser vistas na Tabela 5.9. As posições dos obstáculos nos dois cenários utilizados como exemplos são exibidas na Tabela 5.10.

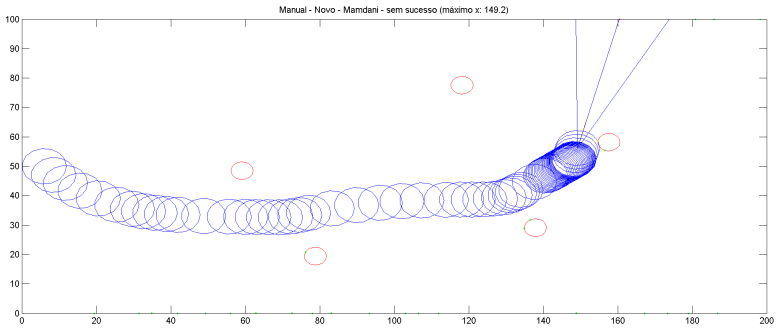
Tabela 5.9: Informações e métricas sobre os controladores selecionados para comparação de desempenho.

Conjunto de dados	**Método de obtenção**	**Modelo**	**Sucessos**	**Passos**	**Soma** ϕ	**Máximo** x_r
—	Antigo	Mamdani	1937	70,2	616,6	108,7
—	Novo	Mamdani	2242	59,9	175,2	95,7
CD1	Novo	Sugeno	2356	68,4	220,4	94,5
CD1	Novo	RBF	2205	63,7	162,4	95,7
CD2	Novo	Sugeno	2351	73,7	231,3	87,0
CD2	Novo	RBF	2165	64,7	158,8	95,0

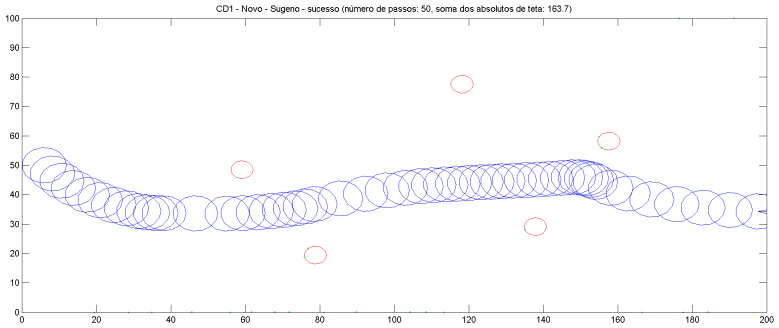
No primeiro exemplo, mostrado nas figuras 5.2 e 5.3, temos um cenário em que o novo controlador Mamdani falha devido a suas propriedades (Seção 4.2.1). Neste caso, podemos ver que tanto os controladores Sugeno quanto os RBF foram bem sucedidos em seus treinamentos, de modo que a interpolação das amostras colhidas a partir do novo Mamdani levou a respostas melhores.



(a)

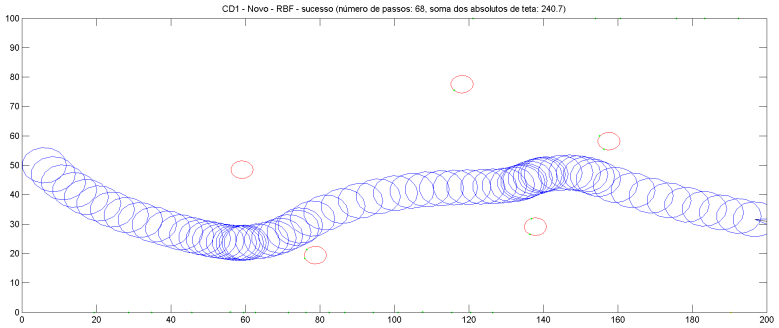


(b)

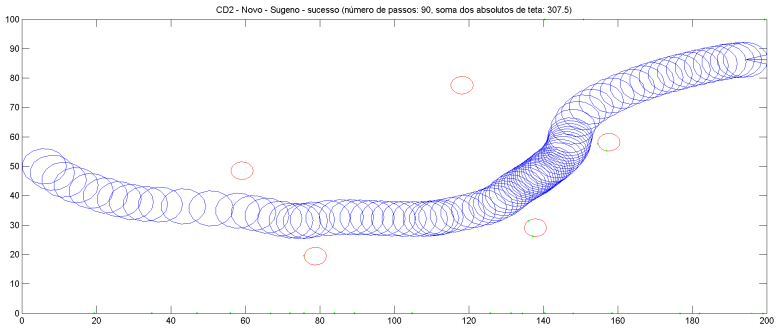


(c)

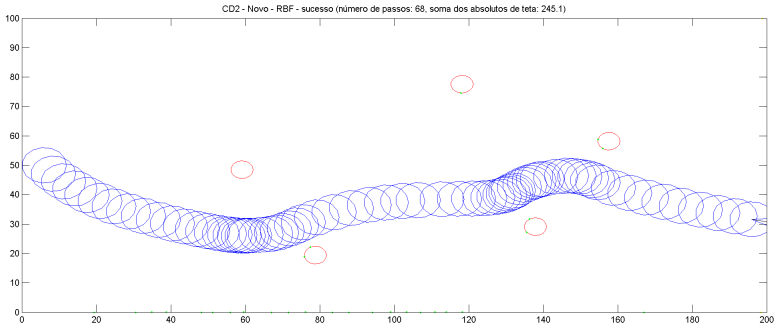
Figura 5.2: Simulações utilizando os controladores selecionados, com posição inicial $x_r = 6$, $y_r = 50$ e $\phi = -60$, no cenário 1.



(a)



(b)



(c)

Figura 5.3: Simulações utilizando os controladores selecionados, com posição inicial $x_r = 6$, $y_r = 50$ e $\phi = -60$, no cenário 1.

Tabela 5.10: Posições dos obstáculos nos dois cenários usados como exemplo.

Cenário	**Obstáculo**	x_o	y_o
1	1	59,10	48,50
	2	78,80	19,40
	3	118,20	77,60
	4	137,90	29,10
	5	157,60	58,20
2	1	128,31	25,87
	2	71,43	26,05
	3	55,65	47,52
	4	39,60	34,77
	5	183,81	47,07

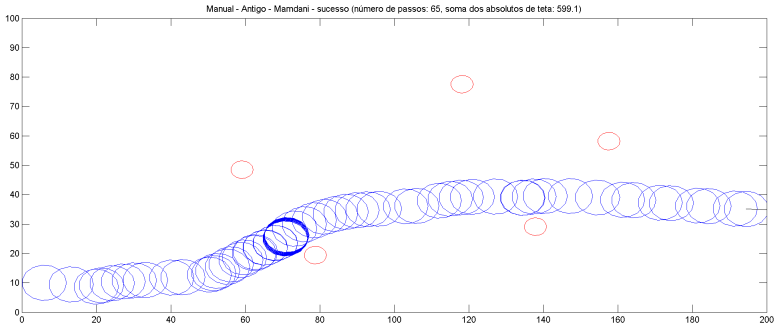
Já nas figuras 5.4 e 5.5 vemos um exemplo de um caso geral, que pode ser usado para explicar alguns dos resultados apresentados anteriormente. Todos os seis controladores terminaram suas simulações.

O antigo controlador Mamdani teve algumas dificuldades na ultrapassagem de um dos obstáculos, durante a qual teve de variar o ângulo das rodas em um intervalo grande, indo de valores negativos a positivos, como pode ser visto na Figura 5.6. Desta forma, a soma dos valores absolutos de θ terminou bastante alta, mesmo com uma quantidade mediana de passos.

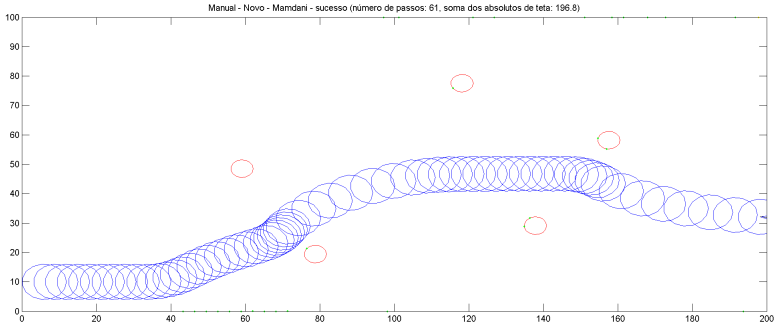
Os controladores Sugeno, apesar de terem alcançado o objetivo, o fizeram tomando um caminho diferente dos outros. Podemos ver que isto aconteceu tanto porque os controladores não corrigiram a rota do robô após o desvio do primeiro obstáculo (para que este ficasse com ϕ próximo a zero), como também porque estes controladores parecem ser mais sensíveis em seus desvios, fazendo com que por vezes tomem caminhos menos simples. Este comportamento explica seus números relativamente altos para as métricas número de passos e soma dos valores absolutos de θ.

Por sua vez, os controladores RBF foram bem sucedidos em suas simulações, realizando a navegação do robô de forma mais direta ao objetivo e mais suave que os demais.

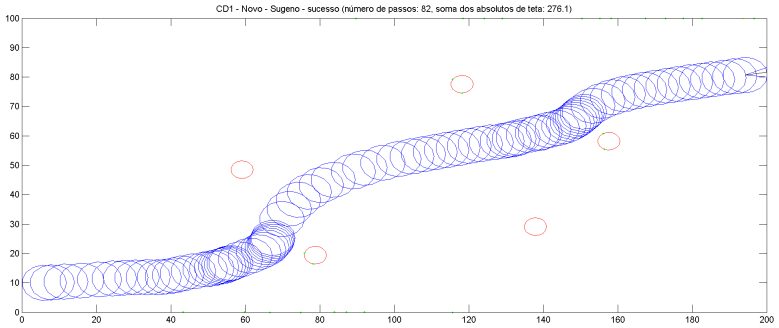
Por fim, nas simulações das figuras 5.7 e 5.8 mostramos um caso em que o excesso de suavidade do modelo RBF acaba fazendo com que seus robôs colidam com obstáculos. Os demais controladores se comportaram de maneira semelhante ao que se comportaram nas simulações exibidas anteriormente.



(a)

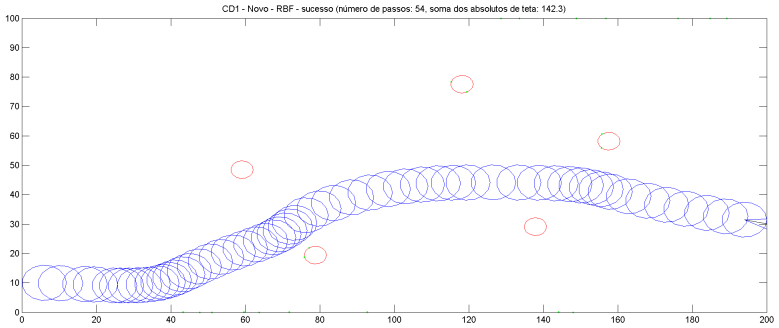


(b)

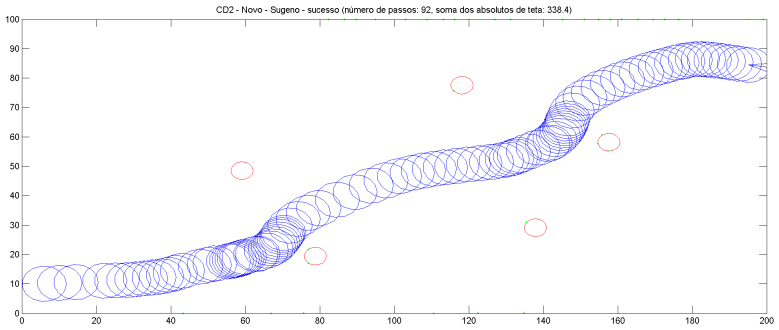


(c)

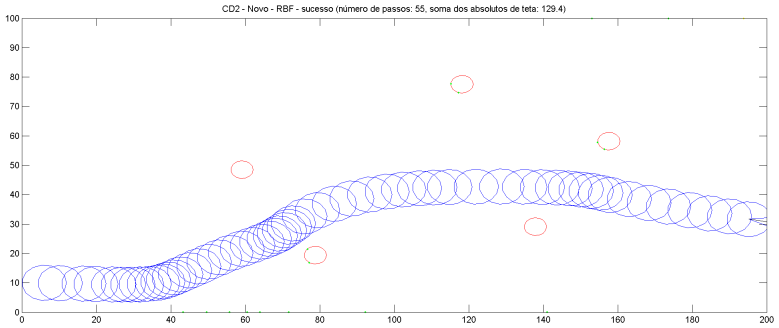
Figura 5.4: Simulações utilizando os controladores selecionados, com posição inicial $x_r = 6$, $y_r = 10$ e $\phi = 0$, no cenário 1.



(a)

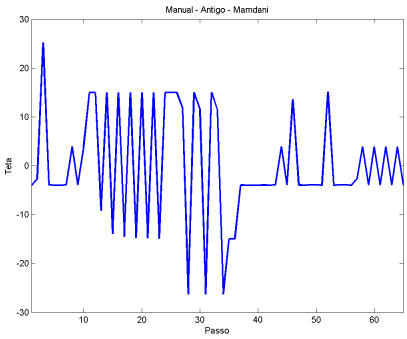


(b)

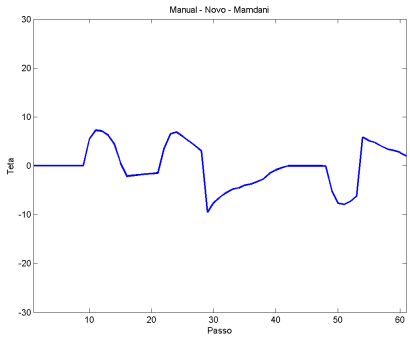


(c)

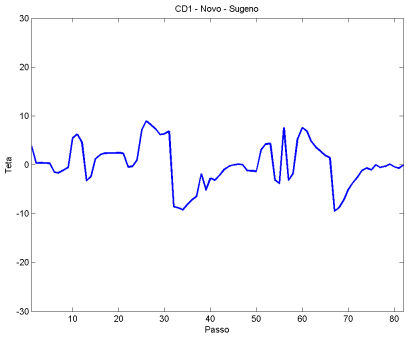
Figura 5.5: Simulações utilizando os controladores selecionados, com posição inicial $x_r = 6$, $y_r = 10$ e $\phi = 0$, no cenário 1.



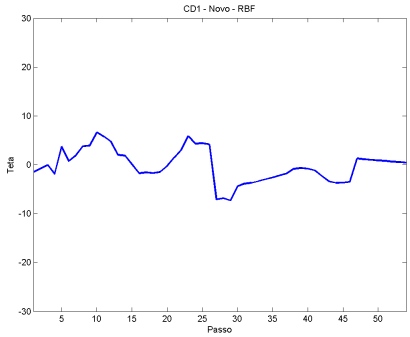
(5.4-a)



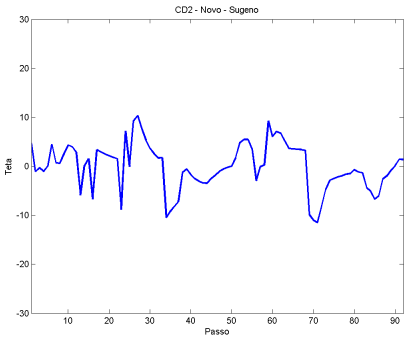
(5.4-b)



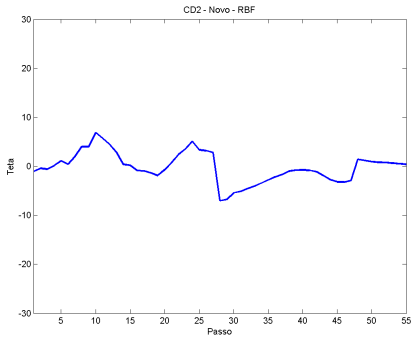
(5.4-c)



(5.5-a)

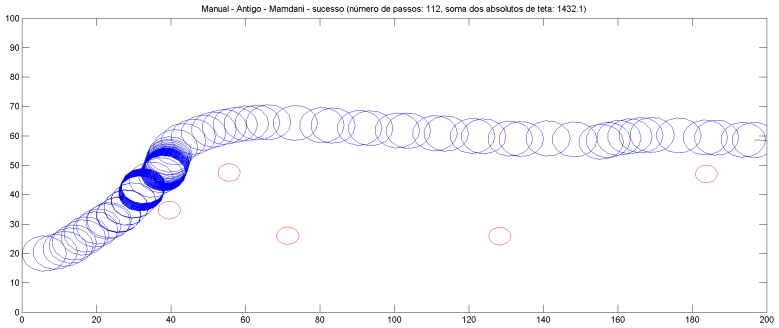


(5.5-b)

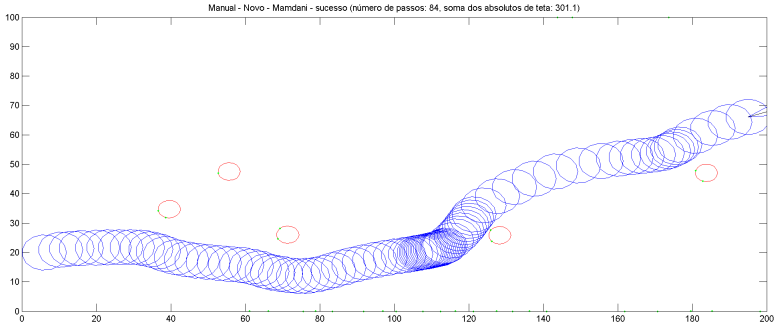


(5.5-c)

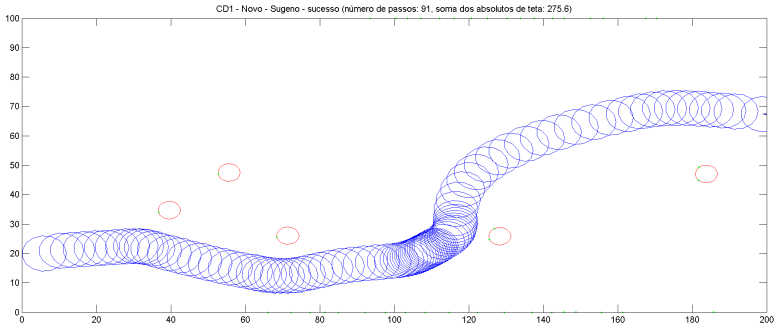
Figura 5.6: Valores de θ para as respectivas simulações das figuras 5.4 e 5.5.



(a)

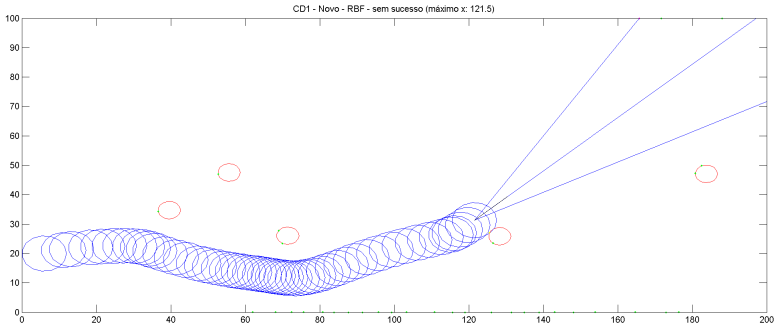


(b)

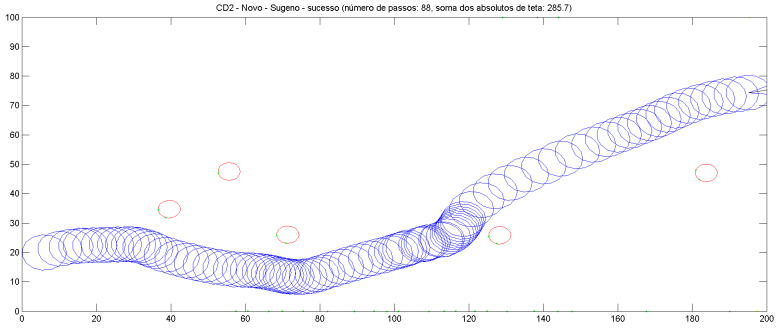


(c)

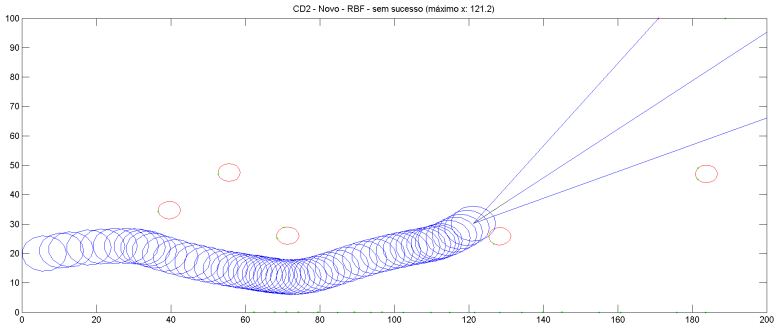
Figura 5.7: Simulações utilizando os controladores selecionados, com posição inicial $x_r = 6$, $y_r = 20$ e $\phi = 15$, no cenário 2.



(a)



(b)



(c)

Figura 5.8: Simulações utilizando os controladores selecionados, com posição inicial $x_r = 6$, $y_r = 20$ e $\phi = 15$, no cenário 2.

6 CONCLUSÃO

6.1 Considerações Finais

Após a conclusão deste trabalho, levando em conta os resultados exibidos e a análise realizada no Capítulo 5, podemos dizer que fomos bem sucedidos em alcançar nossos objetivos.

Os resultados mostram que, com a definição dos novos sensores da maneira como foi feita, estamos caminhando numa boa direção. Além disso, a utilização do pré-processamento, que permite armazenar os pontos já conhecidos e passar informações mais relevantes ao módulo controlador, certamente fez diferença na obtenção de bons resultados.

Uma vez tendo os novos sensores definidos, pudemos construir o novo controlador Mamdani. Com conjuntos e regras voltados ao novo esquema de entradas, o sistema obteve ótimos resultados, sendo melhor que o antigo controlador em três das quatro métricas utilizadas para comparação, incluindo a principal destas. Assim, atingimos o objetivo de criar um novo controlador utilizando novos sensores (mais fáceis de serem implementados no mundo real), e ainda melhorando os resultados em relação

ao antigo controlador.

Ao treinarmos os outros modelos a partir do novo controlador Mamdani, obtivemos excelentes resultados com o modelo Sugeno, aumentando ainda mais a métrica de número de simulações terminadas com sucesso. Contudo, vimos que este ganho foi alcançado, dentre outras coisas, a partir do aumento da sensibilidade do controlador, o que fez com que suas métricas de número de passos e soma de ângulos de rodas também aumentassem.

O modelo RBF, por sua vez, mostrou-se bastante promissor, pois parece não sofrer da mesma deficiência de sensibilidade que o modelo Sugeno. Pelo contrário, por vezes sua falta de sensibilidade foi decisiva para que colidisse mais vezes que os modelos Mamdani e Sugeno, de modo que devemos tentar refiná-lo para que seja de fato utilizado.

Já os modelos MLP e SVM tiverem desempenho bem abaixo do esperado. Sabemos, porém, que a explicação para isto está no fato de que estes modelos possuem parâmetros cuja correta adaptação é bem mais complexa do que a dos parâmetros dos modelos Sugeno e RBF, simplificados em apenas uma camada. E isto nos confirma que, caso tenhamos à diposição informação suficiente para montarmos este tipo de estrutura mais simples, devemos utilizá-la, pois certamente o treinamento convergirá mais rápido e mais facilmente e, provavelmente, trará melhores resultados.

Além disso, vimos também que nem sempre uma maior quantidade de amostras leva necessariamente a um aprendizado mais eficiente e a melhores resultados.

De qualquer modo, podemos dizer que alcançamos o objetivo de, a partir de amostras do novo controlador Mamdani, suavizar sua superfície de saída, visando a corrigir alguns casos em que as propriedades do sistema Mamdani faziam com que o robô

colidisse com obstáculos.

Por fim, devemos dizer que falhamos ao tentar utilizar o conhecimento do antigo controlador na construção de um novo, provavelmente por não conseguirmos mapear corretamente as entradas do antigo controlador para entradas do novo.

Apesar de não termos atingido este objetivo, acreditamos que o trabalho trouxe ótimas contribuições, tanto pela definição dos novos sensores e a construção de novos controladores, que permitirão a implemetanção de um robô real, quanto pela melhora das métricas em relação ao antigo controlador.

6.2 Trabalhos Futuros

Como trabalhos futuros, sugerimos alguns tópicos que podem ser abordados para complementar os estudos realizados por este trabalho, bem como tópicos decorrentes da análise dos resultados deste:

- Tentar refinar o modelo Sugeno, visando a melhorar a métrica de número de passos para as simulações com sucesso.
- Tentar refinar o modelo RBF, em especial as gaussianas de sua camada escondida, de modo a obter resultados comparáveis aos do modelo Sugeno para a métrica número de simulações com sucesso, observando se isto levaria ao detrimento das outras métricas.
- Utilizar Algoritmos Genéticos (GOLDBERG, 1989; MICHALEWICZ, 1994) para a definição dos parâmetros dos modelos (MOTA, 2007; TUNSTEL; JAMSHIDI, 1996; HOMAIFAR; MCCORMICK, 1995; XIAO et al., 1997), avaliando os indivíduos a partir de métricas obtidas através de simulações.

- Fazer com que as entradas do controlador não sejam os mínimos das distâncias em cada faixa, mas sim avaliações sobre as condições de cada faixa, como por exemplo considerar se estas estão cheias ou vazias.

- Posicionar sensores de distância de modo a simular as entradas do antigo controlador de (MORATORI, 2006), tentando aproveitar o conhecimento já disponível neste.

- Verificar o desempenho dos controladores em ambientes ligeiramente diferentes do utilizado neste trabalho, como ambientes com mais obstáculos ou nos quais os obstáculos são móveis.

- Relizar a análise de robustez do robô, inserindo erros artificiais durante as simulações e verificando como o robô se comportaria num ambiente mais próximo ao real.

- Elaboração das especificações de *software* e *hardware* para um robô real baseado no robô virtual construído neste trabalho, e posterior inserção do controlador naquele robô real.

REFERÊNCIAS

ASIMOV, I. (Ed.). **Histórias de Robôs**. 1.ed. [S.l.]: L&PM, 2005. v.1, 2 e 3.

BEOM, H. R.; CHO, H. S. A Sensor-Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement Learning. **IEEE Transactions on Systems, Man and Cybernetics**, [S.l.], v.25, n.3, p.464–477, 1995.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A Training Algorithm for Optimal Margin Classifiers. In: FIFTH ANNUAL WORKSHOP ON COMPUTATIONAL LEARNING THEORY, 1992, Nova Iorque, NY, EUA. **Proceedings...** ACM Press New York, 1992. p.144–152.

BRAGA, A. P.; CARVALHO, A. C. P. L. F.; LUDERMIR, T. B. **Fundamentos de Redes Neurais Artificiais**. Rio de Janeiro, RJ, Brasil: LTC, 2000. 262p.

BURGES, C. J. C. A Tutorial on Support Vector Machines for Pattern Recognition. **Data Mining and Knowledge Discovery**, Holanda, v.2, n.2, p.121–167, 1998.

CALVO, R. **Arquitetura Híbrida Inteligente para Navegação Autônoma de Robôs**. 2007. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação, USP, São Carlos, SP, Brasil.

CANU, S.; GRANDVALET, Y.; RAKOTOMAMONJY, A. SVM and Kernel Methods Matlab Toolbox. **Perception Systmes et Information**, Rouen, França, 2003.

CORTES, C.; VAPNIK, V. N. Support-Vector Networks. **Machine Learning**, Holanda, v.20, n.3, p.273–297, 1995.

CRUZ, A. J. O. **Lógica Nebulosa**. Disponível em: <`http://equipe.nce.ufrj.br/adriano/fuzzy/apostila.pdf`>. Acesso em Abril de 2010.

CYBENKO, G. V. Approximation by Superpositions of a Sigmoid Function. **Mathematics of Control, Signals and Systems**, Londres, Reino Unido, v.2, n.4, p.303–314, 1989.

DRIANKOV, D.; HELLENDOORN, H.; RENFRANK, M. **An Introduction to Fuzzy Control**. [S.l.]: Springer-Verlag, 1993.

DRUCKER, H.; BURGES, C. J. C.; KAUFMAN, L.; SMOLA, A.; VAPNIK, V. Support Vector Regression Machines. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 9, 1997, Cambridge, MA, EUA. **Anais...** MIT Press, 1997. p.155–161.

DUBOIS, D.; WIDROW, B. **Fuzzy Sets and Systems**. Orlando, E.U.A.: Academic Press, 1980.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Reading, PA, E.U.A.: Addison-Wesley, 1989.

GUINGO, B. C.; RODRIGUES, R. J.; THOMé, A. C. G. Automatic Identification for Automotives Vehicles Plates. In: WSEAS INTERNATIONAL CONFERENCE ON NEURAL NETWORKS AND APPLICATIONS (NNA'02), 3., 2002. **Anais...** [S.l.: s.n.], 2002. v.2, p.15–21.

HEINEN, M. R. **Controle Inteligente do Caminhar de Robôs Móveis Simulados**. 2007. Dissertação (Mestrado) — Universidade do Vale do Rio dos Sinos, São Leopoldo, RS, Brasil.

HOMAIFAR, A.; MCCORMICK, E. Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers Using Genetic Algorithms. **IEEE Transactions on Fuzzy Systems**, [S.l.], v.3, n.2, p.129–139, Maio 1995.

HUNG, K.-T.; LIU, J.-S.; CHANG, Y.-Z. A Comparative Study of Smooth Path Planning for a Mobile Robot by Evolutionary Multi-Objective Optimization. **International Symposium on Computational Intelligence in Robotics and Automation**, [S.l.], p.254–259, 2007.

JANG, J. S. R. ANFIS: adaptive-network-based fuzzy inference system. **IEEE Transactions on Systems, Man and Cybernetics**, [S.l.], v.23, n.3, p.665, 1993.

KONG, S.-G.; KOSKO, B. Adaptive Fuzzy Systems for Backing Up a Truck-and-Trailer. **IEEE Transactions on Neural Networks**, [S.l.], v.3, n.2, p.211–223, Mar 1992.

LUENBERGER, D. **Linear and Nonlinear Programming**. Reading, MA, EUA: Addison-Wesley, 1984.

MAMDANI, E. Application of Fuzzy Algorithms for Control of Simple Dynamic Plant. In: IEEE TRANSACTIONS ON COMPUTERS, 1974. **Proceedings...** [S.l.: s.n.], 1974. v.121, p.298–316.

MCCULLOCH, W. S.; PITTS, W. A Logical Calculus of the Ideas Immanent in Nervous Activity. **Bulletin of Mathematical Biophysics**, New York, NY, E.U.A., v.5, n.4, p.115–133, 1943.

MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**. 2.ed. Berlim, Alemanha: Springer-Verlag, 1994. 340p.

MINSKY, M.; PAPERT, S. **Perceptrons – An Introduction to Computational Geometry**. Cambridge, MA, U.S.A.: MIT Press, 1969.

MORATORI, P. B. **Análise da Estabilidade e Robustez de Controladores Nebulosos – Aplicação ao Controle de Trajetória de Robos**. 2006. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática, UFRJ, Rio de Janeiro, RJ, Brasil.

MORATORI, P. B.; CRUZ, A. J. O.; PEDRO, M.; FERREIRA, E.; MANHAES, M.; ANDRADE, L.; LIMA, J. C. M. Analysis of Stability of a Fuzzy Control System Developed to control a simulated robot. In: IEEE INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS (FUZZ-IEEE 2005), 2005. **Proceedings...** [S.l.: s.n.], 2005. p.726–730.

MOTA, T. C. **Aplicação de Algoritmos Genéticos à Definição da Arquitetura e ao Treinamento de Redes Neurais MLP**. 2007. Monografia de Projeto Final de Curso — Instituto de Matemática, UFRJ, Rio de Janeiro, RJ, Brasil.

MOTA, T. C.; THOMé, A. C. G. One-Against-All-Based Multiclass SVM Strategies Applied to Vehicle Plate Character Recognition. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN'09), 2009, Atlanta, GA, E.U.A. **Proceedings...** [S.l.: s.n.], 2009. p.2153–2159.

NEARCHOU, A. C. Path Planning of a Mobile Robot Using Genetic Heuristics. **Robotica**, [S.l.], v.16, p.575–588, 1998.

ROSENBLATT, F. The Perceptron – A Probabilistic Model for Information Storage and Organization in the Brain. **Psychological Review**, Washington, DC, E.U.A., v.65, n.6, p.386–408, 1958.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning Representations by Back-propagating Errors. **Nature**, Londres, Reino Unido, n.323, p.533–536, 1986.

SANDRI, S.; CORREA, C. Lógica Nebulosa. In: V ESCOLA DE REDES NEU-

RAIS, 1999, São José dos Campos, SP, Brasil. **Anais...** [S.l.: s.n.], 1999. p.c073–c090.

SHI, D.; DUNLAP, D.; COLLINS, E. G. A Comparison Between a Fuzzy Behavioral Algorithm and a Vector Polar Histogram Algorithm for Mobile Robot Navigation. In: INTERNATIONAL SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE IN ROBOTICS AND AUTOMATION, 2007. **Anais...** [S.l.: s.n.], 2007. p.260–265.

SILVA, E. **Reconhecimento Inteligente de Caracteres Manuscritos**. 2002. Dissertação (Mestrado) — Instituto Militar de Engenharia, Rio de Janeiro, RJ, Brasil.

SMOLA, A. J.; SCHöLKOPF, B. A Tutorial on Support Vector Regression. **Statistics and Computing**, Hingham, MA, EUA, v.14, n.3, p.199–222, 2004.

TAKAGI, T.; SUGENO, M. Fuzzy Identification of Systems and its Applications to Modeling and Control. **IEEE Transactions on Systems, Man and Cybernetics**, [S.l.], v.15, p.116–132, 1985.

TUNSTEL, E.; JAMSHIDI, M. On Genetic Programming of Fuzzy Rule-Based Systems for Intelligent Control. **International Journal of Intelligent Automation & Soft Computing**, [S.l.], v.2, n.3, p.273–284, 1996.

USHIZIMA, D. M.; LORENA, A. C.; CARVALHO, A. C. P. L. F. de. Support Vector Machines Applied to White Blood Cell Recognition. In: FIFTH INTERNATIONAL CONFERENCE ON HYBRID INTELLIGENT SYSTEMS (HIS'05), 2005. **Anais...** [S.l.: s.n.], 2005. p.379–384.

WANG, L. X.; MENDEL, J. M. Generating Fuzzy Rules by Learning from Examples. **IEEE Transactions on Systems, Man and Cybernetics**, [S.l.], v.22, n.6, p.1414, 1992.

WIDROW, B.; HOFF, M. E. Adaptative Switching Circuits. In: WESTERN ELECTRONIC SHOW AND CONVENTION, 1960, New York, NY, E.U.A. **Anais...** Institute of Radio Engineers, 1960.

XIAO, J.; MICHALEWICZ, Z.; ZHANG, L.; TROJANOWSKI, K. Adaptive Evolutionary Planner/Navigator for Mobile Robots. **IEEE Transactions on Evolutionary Computation**, [S.l.], v.1, n.1, p.18–28, 1997.

ZADEH, L. A. Fuzzy Sets. **Information Control**, [S.l.], v.8, p.338–353, 1965.